VIDI:
A LIGHTWEIGHT PROTOCOL BETWEEN
VISUALIZATION SYSTEMS AND DIGITAL LIBRARIES

by
JUN WANG

A Thesis Presented to the Faculty of
the Virginia Polytechnic Institute and State University
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in
Computer Science

Prof. Edward A. Fox – Chairman
Prof. Chris North
Prof. Clifford A. Shaffer

May 10, 2002
Blacksburg, Virginia

Keywords: Digital Libraries, Visualization, Protocol, VIDI, Interoperability, OAI

Copyright © 2002 Jun Wang
VIDI:
A LIGHTWEIGHT PROTOCOL BETWEEN VISUALIZATION SYSTEMS AND DIGITAL LIBRARIES

by
JUN WANG

Committee Chairman: Dr. Edward A. Fox
Computer Science

(Abstract)

Achieving interoperability between digital libraries and visualization tools is a difficult problem. To solve this problem, a version of the Open Archives Initiative (OAI) Protocol for Metadata Harvesting called VIDI is proposed. It is a lightweight protocol, which contains only 5 request verbs -- Identify, ListMetadataFormats, ListVisdataFormats, ListTransformers, and RequestResultSet. It is extended from the OAI protocol, which enables its simplicity and wider acceptability. It is flexible, which avoids a rigid architecture in implementation. It is general, so it can apply to all kinds of Visualization Systems and Digital Libraries. But most importantly, it reaches our goal of enabling operability between Visualization Systems and Digital Libraries. The protocol design and implementation details are given. Two prototype systems are implemented to demonstrate the above features. Implementation details are given about ENVISION-ODL and ENVISION-MARIAN. Analysis, evaluation, and conclusions reinforce the discussion of the benefits of VIDI.
To:

The Ones I Love Deeply

And the Ones Who Love Me Deeply

In My Whole Life...
Acknowledgement

I would like to express my appreciation to my advisor, Professor Edward A. Fox, for the great amount of time, effort, support, and guidance from him during the past year. I am deeply grateful for his understanding and patience during the research, the thesis writing, and the information storage & retrieval course last semester.

I extend my sincere thanks to Dr. North for his guidance, help, and comments for the research on ENVISION and other visualization issues. The information visualization course he taught last semester is one of the most unforgettable courses I have ever taken. Thanks go to Professor Shaffer for his willingness and promptness in answering all my questions and setting my defense date. I also thank all professors who taught me in the last two years.

I specially thank Abhishek Agrawal and Wensi Xi for helping me in correcting grammatical errors, and for all kinds of other help in the most urgent phase of my thesis writing. Thanks go to my dear Digital Library Research Lab colleagues Hussein Suleman, Marcos A. Gonçalves, Rao Shen, and Ryan Richardson for their kind offer of help in their specific areas. Thanks go to all my other project buddies on ENVISION and MARIAN, Anil Bazaz, Supriya Angle, Ye Zhou, Ming Luo, Liang Xu, and all other lab members for their supporting in my thesis writing, spending the whole last year with me, and sharing with me their valuable experiences. I will miss you all always.

Special thanks go with Robert France, a great project leader, who had spent the last couple of months in his life with me on the MARIAN project.

I owe thanks for all the people who have been supporting me or caring for me in my way of learning, and for support by NLM and VT. This material is based in part upon work supported by the NSF through grants: CDA-9312611; DUE-0121741, 0136690, 0121679; IIS-0080748, 0086227, 0002935, and 9986089.
# Table of Contents

## CHAPTER I INTRODUCTION ................................................................. 1
  1.1 Problem Statement ........................................................................... 1
  1.2 Solution Outline ............................................................................... 4
  1.2 Solution Outline ............................................................................... 5
  1.3 Thesis Organization ......................................................................... 7

## CHAPTER II REVIEW OF LITERATURE .............................................. 8
  2.1 Terminology and Conventions ......................................................... 10
    2.1.1 DL ................................................................................................. 10
    2.1.2 VIS ................................................................................................. 10
    2.1.3 OAI ................................................................................................. 10
  2.2 Digital Libraries ................................................................................ 12
    2.2.1 MARIAN ......................................................................................... 13
    2.2.2 ODL ................................................................................................. 13
    2.2.3 Others ............................................................................................ 14
  2.3 Visualization Systems ....................................................................... 15
    2.3.1 A Classification of Visualization Systems ......................................... 15
    2.3.2 Infosphere Visualization .................................................................. 15
    2.3.3 Workspace Visualization .................................................................. 16
    2.3.4 Visual Knowledge Tools ................................................................... 18
    2.3.5 Visual Object Visualization ............................................................. 22
  2.4 Observations ..................................................................................... 24

## CHAPTER III DESIGN ........................................................................ 25
  3.1 Problems and Proposed Solutions .................................................... 25
    3.1.1 Problems .......................................................................................... 25
    3.1.2 Proposed Solutions .......................................................................... 26
  3.2 VIDI Protocol Design Concerns ....................................................... 28
    3.2.1 Use of XML, XSLT and schema ....................................................... 28
    3.2.2 Use of HTTP Request and Response ................................................ 30
  3.3 VIDI Protocol .................................................................................... 32
    3.3.1 Protocol Entities Definitions ............................................................ 32
    3.3.2 Protocol Features ............................................................................ 36
    3.3.3 Protocol Commands Summary ......................................................... 41
    3.3.4 Protocol Commands Breakdown ....................................................... 42
    3.3.5 Protocol Commands Scenarios ......................................................... 55
  3.4 Solutions to Problem Scenarios with VIDI Protocol ............................ 62
    3.4.1 Terminology used below .................................................................. 62
    3.4.2 Solutions for Scenarios .................................................................... 62

## CHAPTER IV IMPLEMENTATION ......................................................... 65
  4.1 Important Implementation Decisions ................................................ 65
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The two common models for a user to access DLs</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Visiting multiple DLs using model 2</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>New model</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Visiting multiple DLs and VISs using model 3. VIS$_1$ supports DL$_1$ and DL$_2$; VIS$_2$ supports DL$_2$ and DL$_3$; VIS$_3$ supports only DL$_3$</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>MARIAN system architecture</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Map of Internet</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>WebBook and Web Forger</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Data Mountain</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>The ENVISION architecture</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>HomeFinder</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>FilmFinder</td>
<td>21</td>
</tr>
<tr>
<td>2.8</td>
<td>Table Lens</td>
<td>22</td>
</tr>
<tr>
<td>2.9</td>
<td>Visible Human</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>Updating + General Scenario</td>
<td>60</td>
</tr>
<tr>
<td>4.1</td>
<td>Hierarchical structure of XML documents and DOM tree</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>Metadata part in Dublin Core Schema</td>
<td>68</td>
</tr>
<tr>
<td>4.3</td>
<td>MARIAN nlmServlet process user request to Webgate</td>
<td>69</td>
</tr>
<tr>
<td>4.4</td>
<td>Original Envision Request format. The character behind ENVA is actually a binary value, which is equal to the length of the query string</td>
<td>73</td>
</tr>
<tr>
<td>4.5</td>
<td>Portion of ENVISION Result File</td>
<td>74</td>
</tr>
<tr>
<td>4.6</td>
<td>ODL metadata format</td>
<td>77</td>
</tr>
<tr>
<td>4.7</td>
<td>Flow model in prototype implementation</td>
<td>79</td>
</tr>
<tr>
<td>4.8</td>
<td>Sample VIDI request string for command RequestResultSet</td>
<td>84</td>
</tr>
<tr>
<td>4.9</td>
<td>XML document in DIRLINE metadata format</td>
<td>93</td>
</tr>
<tr>
<td>4.10</td>
<td>XML document portion in DIRVIS visdata format</td>
<td>93</td>
</tr>
<tr>
<td>4.11</td>
<td>ENVISION-MARIAN VIDI process</td>
<td>95</td>
</tr>
<tr>
<td>4.12</td>
<td>Marian nlmServlet handles user’s request and talks to Webgate</td>
<td>97</td>
</tr>
<tr>
<td>5.1</td>
<td>SOM frontend interface</td>
<td>100</td>
</tr>
<tr>
<td>5.2</td>
<td>SOM internal structure</td>
<td>101</td>
</tr>
<tr>
<td>5.3</td>
<td>SOM applying VIDI</td>
<td>103</td>
</tr>
<tr>
<td>5.4</td>
<td>UVA interface</td>
<td>104</td>
</tr>
<tr>
<td>5.5</td>
<td>User contact DL to get information to provide to Spotfire</td>
<td>108</td>
</tr>
<tr>
<td>5.6</td>
<td>Spotfire provides more support for user to access DL</td>
<td>108</td>
</tr>
</tbody>
</table>
Table of Tables

Table 3.1 Field names for different commands.................................................40

Table 4.1 Mapping between Dublin Core metadata format and Enva visdata format.....76
Table 4.2 URL encoding table.................................................................81
Table 4.3 Fields in DIRLINE metadata format, DIRVIS visdata format, meanings and mapping.................................................................90
CHAPTER I
INTRODUCTION

1.1 Problem Statement

The emergence of digital libraries enables easier on-line access over more diverse formats of digital information, and varied functions for handling the information retrieved from them. At the same time, common users with different expertise, who have only a basic knowledge of information systems and the ways to access information, may face difficulties and perceive unmanageable complexity. One approach to these problems calls for nontraditional methods of information retrieval. Visualization (VIS) can transform the unseen internal semantic representations of data (e.g., metadata) into visible geometric displays. It has the potential to help searchers explore new retrieval options and to bring recognition capacities for users in either discovery or display of the information. More and more research is now being done in information visualization because of its relevance and its apparent advantages in the field of digital libraries.

There are two common models existing for users to access digital libraries. One is the classical model: a user directly fills in queries inside the specified fields on a simple interface provided by the digital library. Almost all digital libraries support this kind of retrieval service (e.g., IEEE Xplore, ACM Digital Portal). The retrieved result contents (for text documents or other formats, e.g., multimedia) are presented to users, usually in the form of a sorted list. The sorted list is useful when a user browses and judges the relevance of the results, but its simple linear presentation hardly reflects any sophisticated semantic and logical relationships among the results. Users cannot get any useful information from a retrieved results list to understand the relationships among entries, which in turn might aid them to refine their search and get more useful information from the results.
The second model applies information visualization and human-computer interaction theories and techniques. A visualization system becomes the major application connecting a user and a digital library. It communicates with the user to form the request and sends it to the back-end digital library. Then it retrieves the results and presents them to the user through some visualization techniques the specific tool is employing. Some digital libraries are adopting this method to help users who are analyzing some specific format of result records. In this way, the benefits of visualization representation of information are utilized to a large extent. With the help of visualization systems, users require less time to understand the results of a search and are likely to make more correct decisions on the selection of relevant and useful documents.

However, in the second model, all the details of a DL have been hidden from the user by a VIS tool. See Figure 1.1. The relation between DL and VIS is fixed from the very beginning. They transfer query and result data in some format that both sides can recognize. Similarly, people were used to the homogeneous situation in which an information system was supporting some specific type of information. Also, the users of the system came from a specified group or groups whose interests are similar. Although much broader in scope than many information systems, DLs are based almost exclusively on this design. Regarding visualization, however, there are two types of VIS tools. One is free format; tools in this category can take any type of data as input and visualize them in general ways, but have weaker visualization effects. The second type is composed of tightly coupled visualization tools, which are very powerful in presenting some specific formats, but cannot easily be adapted to different structures. Presently, almost all visualization tools for digital libraries fall into the second category: the guideline for designing these tools is very specific and the tools have been optimized to the largest degree based on the designated information resource. They have done well with regard to this purpose and they usually can render results to users in an amazingly elegant and efficient way. But the requirement for specificity is so dominant that it affects everything in the system design, including choice of algorithm, selection of visualization space, and details of code. The visualization tool operating in this way is totally useless in situations falling aside of the context of the data structure it supports.
The larger the system is, the more it suffers from increasing complexity and diversity of information. Similarly the users that come to that large system may have different needs and hence are expected to use varied visualization tools. Digital libraries themselves often are huge, with sizes still growing. It is fundamentally important to find a way to pass the data available into various visualization tools. However, it is impractical and unnecessary to generate a visualization tool that supports all advanced visualization techniques and also all kinds of data formats, especially when both technologies are advancing quickly. So, naturally, multiple visualization tools should be used. Further, nowadays users want to have access to more and more information. It is very common that users are searching in multiple systems concurrently, and they want to be able to analyze the results together or in a unified way. Thus, the old one-to-one digital-library-to-visualization-tool model is not applicable any more. Further, the second model we presented above shows its clumsiness and rigidity in dealing with this situation, as is illustrated in Figure 1.2.

![Model 1](image1)

![Model 2](image2)

Figure 1.1. The two common models for a user to access DLs
Figure 1.2. Visiting multiple DLs using model 2
1.2 Solution Outline

Research on joining digital libraries together offers another approach that could help in solving the interoperability issue we are facing. The OAI Protocol [1] is a good example in this direction. (Please refer to Chapter 2 for more details.) However, joining service can only work for digital objects that are very similar or the same. Also there is the critical problem of assigning weights (or estimated relevance value) consistently to digital objects in varied collections or digital libraries. In this thesis we present a new approach that does not conflict with, but rather complements our work on interoperability. It is a model that actually combines the two former models we discussed above, and at the same time provides more functionality. Then we propose a lightweight protocol that is an extension of OAI and can be used inside this model to enable interoperability and scalability.

In the new model (see Figure 1.3) the digital libraries and the visualization systems are looked upon as counterparts. The users can access both DL & VIS synchronously and freely (see Figure 1.4), as long as both sides support our VIDI protocol. Interoperability and scalability are the most important attributes we consider here. Interactions are dynamically invoked in accord with a user’s will, and data transfer methods can be dynamically generated and agreed upon by both parties. Communication between them not only includes data transfer, but also identification, recognition, and agreement on communication methods.

![Figure 1.3. New model](image-url)
Figure 1.4. Visiting multiple DLs and VISs using model 3. VIS₁ supports DL₁ and DL₂; VIS₂ supports DL₂ and DL₃; VIS₃ supports only DL₃.
1.3 Thesis Organization

The remainder of this thesis details the new protocol that enables well-organized communication between digital libraries and visualization tools. It consists of the following:

Chapter 2 provides background materials that are necessary to obtain a better understanding of digital libraries and visualization tools. It also presents a survey of recent work in these two fields.

Chapter 3 details the methodology and technologies related to the protocol design.

Chapter 4 discusses the implementation of the protocol in the context of two specific case studies: ENVISION-ODL and ENVISION-MARIAN.

Chapter 5 presents an analysis of the protocol’s possible implementation based on multiple types of visualization systems, and discusses related design and implementation issues.

Chapter 6 presents concluding remarks, summarizes contributions, and offers recommendations for future work.
CHAPTER II
REVIEW OF LITERATURE

The origins of modern digital libraries can be traced back to the information retrieval systems of the 1960s and the hypertext systems of the 1980s. Digital computers and digital storage are the key technologies that have made digital libraries feasible and viable. The World Wide Web (WWW) facilitated the birth of the modern digital library technologies.

The WWW is frequently thought of as the technology that revolutionized computer networking by effectively breaking down the barriers between the providers of content and the users of that content. However, it also created information management problems for which simple solutions did not exist. One such problem is that of persistence: how can one guarantee that a digital object on the WWW will always exist? Another question is authority: how can people trust the authenticity of a source of digital objects? These and other concerns led some individuals and organizations to begin creating managed repositories of digital information, nowadays called Digital Libraries (DLs), with additional and specialized services to enhance the users’ experiences beyond what the WWW can offer.

Since the mid 1990s, Digital Libraries have emerged as one of the central and most compelling applications in the world; numerous Digital Library research projects are underway. In 1994, six large-scale pilot projects were funded jointly by ARPA, NASA, and NSF jump-starting research in the field. Digital Libraries is now a vibrant research area, and also a field in which considerable commercial development is taking place (presaging the future economic importance of Digital Library technology to the United States). Many new or parallel technologies therefore will start to integrate in or link with Digital Library research as a result of this flowering of activities, among which visualization is one of the most useful and promising aids.
Research activities concerning visualization technologies, Human-Computer Interaction (HCI), and usability & use studies have been popular since the 1980s. Related studies and scientific analyses have shown that visualization can amplify users’ cognition and enhance users’ information access experiences in the following ways:

- **Increased Resources:** First, the human visual system has the highest bandwidth among all the channels available for a human to perceive the outside world. Second, it can help a user to process information in a parallel approach. Lastly, it saves a user from overburdened memory attributed to its direct presentation of information.

- **Reduced Search:** The locality attribute of visualization reduces the time necessary for a user to process information.

- **Enhanced Recognition of Patterns:** Recognition is easier than recall. The usage of visualization techniques such as abstraction and aggregation simplifies and organizes information. A user is more inclined to find patterns through vision activities.

- **Perceptual Inference and Monitoring:** Visualization can enable a large number of computations and monitoring techniques that can help humans understand.

- **Manipulable Medium:** It aids the user to directly operate on the values of the data of interest.

In general, it is helpful to personalize Digital Libraries through exploration of information spaces [32]. Over time, more complex representation techniques of data presentation have been pursued and proved to be useful. The term "digital object" is intended to describe all such types of information objects. Digital objects in a digital library can have all kinds of formats -- textual, audio, video, numeric, computer programs, or multimedia composites of such components. Different attributes of digital objects demand different visualization techniques to be used to investigate data. Consequently a variety of schemes have been developed to help users work with search results sets, browse, or accomplish other tasks.
Advanced Digital Libraries demand more research into the related visualization area to satisfy the growing needs of users. Thus, developing novel interfaces to DLs has been a popular line of HCI research, building upon the related field of user interface design [34]. Also, Digital Libraries provide opportunities and challenges for exploring all novel interfaces – complex features, data volumes, formats, and other aspects provide many difficult and interesting problems for HCI experts. The cooperation from both areas has been proven efficient and fruitful. [41-49]

2.1 Terminology and Conventions

2.1.1 DL

DL is an abbreviation for “Digital Library”. The D-Lib Working Group on Digital Library Metrics scopes the definition of digital libraries as, “The collection of services and the collection of information objects and their organization, structure, and presentation that support users in dealing with information objects available directly or indirectly via electronic/digital means.” (http://www.dlib.org/metrics/public/papers/dlib-scope.html)

2.1.2 VIS

VIS is an abbreviation for “Visualization System”. It represents the system that makes use of computer-supported, interactive, visual representations of abstract data to amplify cognition. [20]

2.1.3 OAI

In October of 1999 the Open Archives Initiative (OAI) [31] was launched in an attempt to address interoperability issues among the many existing and independent DLs. OAI develops and promotes interoperability standards that aim to facilitate the efficient dissemination of content. It has roots in an effort to enhance access to e-print archives as a means of increasing the availability of scholarly communication, but later was extended to all content providers [37]. The focus was on high-level communication among
systems and simplicity of protocols. The OAI has since received considerable attention in the DL community and, primarily because of the simplicity of its standards, has attracted many early adopters.

The OAI Protocol for Metadata Harvesting [1] supports an application-independent interoperability framework that can be used by a variety of communities who are engaged in publishing content on the Web. In the protocol there are two classes of participants: data providers and service providers. The data provider provides metadata about its content through the OAI protocol, while the service provider asks the data provider for metadata and uses the returned metadata as a basis for building value-added services.

Examples of some DL systems that are connected with the Open Archives Initiative are: Computer Science Teaching Center (CSTC, 2001), Computing and Information Technology Interactive Digital Education Library (CITIDEL, 2001), Los Alamos e-Print archive (LANL, 2001), the American Memory Project (Library of Congress, 2001), and the National Science, Technology, Engineering and Mathematics Education Digital Library (NSF, 2001). Hussein Suleman is developing a Digital Library toolkit framework with the name of “Open Digital Libraries”. [4] It applies and extends the OAI-PMH to achieve interoperability through metadata exchange among digital library components.
2.2 Digital Libraries

The DL definition we saw above is only one broad conception among many other similar definitions. Terms such as "electronic library" and "virtual library" are often used synonymously. The elements that have been identified as common to these definitions are:

- The digital library is not a single entity.
- The digital library requires technology to link the resources of many.
- The linkages between the many digital libraries and information services are transparent to the end users.
- Universal access to digital libraries and information services is a goal.

(http://sunsite.berkeley.edu/ARL/definition.html)

According to the definition above and our own understanding, the key features of a Digital Library are:

- **Multiple formats**
  The presentation methods of DL contents have become more and more complex and different. These methods include hypertext, virtual reality, graphics, multimedia, document interchange, word processing, desktop publishing, and scholarly publishing.

- **Public availability**
  The ability of the user to access, reorganize, and utilize this repository is enriched by the capabilities of digital technology. The comprehensiveness and value of the collection accessible through a Digital Library system can be strengthened by the ability to integrate materials in digital formats that have not been well-represented, easy to access, or effectively usable in traditional library collections, such as multimedia, geospatial data, or numerical datasets. Because of the digitalized feature, all this information is urgently waiting to be better utilized and represented to users.

- **Increasing usage**
Nowadays, users increasingly have access to various types of digital collections and information systems, including personal information resources; workgroup and organizational information collections; collaboration environments; and more public digital libraries. Users also have been using diverse visualization systems.

Next, we consider selected available DL systems for detailed analysis.

### 2.2.1 MARIAN

MARIAN is an indexing, search, and retrieval system optimized and extended for digital libraries. It was developed at the Virginia Tech Computing Center, with development continuing at the DLRL. Figure 2.1 describes the structure of the MARIAN system.

![MARIAN System Architecture](image)

**Figure 2.1. MARIAN system architecture**

MARIAN is an indexing, search, and retrieval system optimized and extended for digital libraries. It was developed at the Virginia Tech Computing Center, with development continuing at the DLRL. Figure 2.1 describes the structure of the MARIAN system.

### 2.2.2 ODL
An Open Digital Library (ODL) is a network of extended Open Archives that work together to supply services required by information seekers. Each node within this network is an ODL service, designed to conform to a set of principles for maximum reusability at the levels of design, implementation, and information sharing.

The Open Digital Library design framework allows building digital libraries in a systematic manner based on current principles of good software engineering, with state of the art digital library services and networked information provision. The ODL design framework specifies strategies for the construction of modular DLs in a new information society where we can no longer afford to consider issues such as interoperability to be optional afterthoughts.

2.2.3 Others

Further analyzing DL systems, we observe the following:

- They are built in isolation as a response to the needs of a particular community.
- The underlying program logic varies vastly among systems.
- Most modern DLs have WWW interfaces – thus the user interfaces and process flows are fashioned to resemble the way people use the WWW, which itself changes with time.
- DLs, by the very nature of having to provide responses to user needs, can be arbitrarily complex. This complexity will continue to increase over time.
2.3 Visualization Systems

All kinds of visualization techniques are used in VIS systems. Keim [30] lists the following prevailing visual data exploration techniques:

- **Geometric techniques**: Examples include scatterplots, landscapes, projection pursuit, posection views, hyperslice, parallel coordinates, …
- **Icon-based techniques**: Famous ones are Chemoff faces, stick figures, shape-coding, color icons, TileBars, …
- **Pixel-oriented techniques**: Example include recursive pattern technique, circle segments technique, spiral axes techniques, …
- **Hierarchical techniques**: We note familiar names like dimensional stacking, Worlds-within-Worlds, treemap, cone trees, infoCube, …
- **Graph-based techniques**: Some are basic graphs (straight-line, polyline, curved-line, ...); others can be specific graphs (e.g., DAG, Symmetric, cluster); or even systems (e.g., Tom Sawyer, Hy, SeeNet, Narcissus, …)

In actual systems, one of the techniques mentioned above normally dominates. Sometimes several techniques are combined, i.e., applying hybrid techniques. All these techniques provide a rich variety of representations in which information elements are conveyed to the users, and help the users obtain insight from the data.

### 2.3.1 A Classification of Visualization Systems

A clear classification of visualization systems will be very helpful in the design of a communication protocol between digital libraries and visualization systems. Visualization systems have at least four levels of usage [21]: (1) visualization of the infosphere, (2) visualization of an information workspace, (3) visual knowledge tools, and (4) visual objects.

### 2.3.2 Infosphere Visualization
The infosphere (or cyberspace) is the reachable space of information sources, e.g., the World Wide Web, or a specific organizational document collection or digital libraries. The applications falling into this category include Bray’s map of 1995’s Internet [22]. See Figure 2.2. This type of application describes the whole space that contains all the documents, or it could be more abstract. It can take the form of a hierarchy to reveal more details by a top-down approach.

![Map of Internet](http://www5conf.inria.fr/fich_html/papers/P9/Overview.html)

**Figure 2.2. Map of Internet**

### 2.3.3 Workspace Visualization

The workspace is a collection of objects whose access is arranged to make relevant tasks efficient. A range of applications support this type of visualization. For example, Card’s WebBook & Web Forager [23] and Data Mountain [24], illustrated in Figure 2.3 and 2.4, support book-title or web pages collections.
Figure 2.3. WebBook and Web Forager
http://www.acm.org/sigs/sigchi/chi96/proceedings/papers/Card/skc1txt.html

Figure 2.4. Data Mountain
2.3.4 Visual Knowledge Tools

Most visualization systems fall in this category. They may arrange information to reveal patterns, allow the manipulation of information for finding patterns, and allow visual calculations. Among them are ENVISION [9], Ahlberg and Shneiderman’s dynamic homeFinder and filmFinder query systems [25], Table Lens [27], and Spotfire.

2.3.4.1 ENVISION

ENVISION provides query functions over fields of authors, titles and both fields. It allows users to retrieve prior search results by saving query history. It also supports hierarchical queries – new queries can be created by modifying original queries, and more refined results can be retrieved accordingly. However, ENVISION is normally considered as a visualization system because of its dominant visual representation features. It uses multiple representation formats, such as icons, colors, shapes, and locations, to convey attributes of a document. When a user inputs a query, he or she can...
choose to view results in a three-window suite – ENVISION graphic view, ENVISION query window, and ENVISION result window. The result documents could be arranged on the graphic view based on values for different attributes, such as the growth in the literature over time, the documents clustered by relevance (similarity to query), and the distribution of relevant documents by subject. Consequently a user can easily evaluate many related issues by trying different combinations, e.g., on the X-axis documents are ordered by the author values and on the Y-axis by date values, with document types represented through shape.

ENVISION has gone through several phases of development. During 1991-1995, it was connected to a C/C++ version of the MARIAN search engine and provided the graphic view presentations of academic library data in MARC format from the Virginia Tech Library. Beginning in 1998, the MARIAN system has been reengineered into Java; consequently ENVISION has been rewritten from C into Java as well during students’ course study and independent study. In 2001, the ENVISION interface was refurbished with some important visualization techniques, such as aggregation to show overview [26]. The new interface is more flexible than the original rigid matrix view, applying a variable-width algorithm, and a show-icons-inside-cluster presentation method. Also it has utilized new tools such as Java Swing. The Java version will replace the old C/C++ MARIAN and provide campus support for library catalog records.

2.3.4.2 HomeFinder and FilmFinder

The dynamic HomeFinder and FilmFinder query system [25] is a tight coupling of dynamic query filters with starfield displays. Its major feature is using dynamic queries to improve performance and acquire high levels of user satisfaction. By allowing rapid, incremental and reversible changes to query parameters, often simply by dragging a slider, users were able to explore and gain feedback from displays in a few tenths of a second.
The HomeFinder showed a map of Washington, DC and 1100 points of light indicating homes for sale. It enabled users to adjust upper and lower bounds on home prices and see available properties.

The FilmFinder is used to find and learn about a film from a library of 10,000 videotapes. The axes and fields are fixed in the interface. The X-axis represents time and the Y-axis a measure of popularity. The labels on the axes are updated as zooming occurs when the user limits the scope of search. When fewer than 25 films are visible, their titles are automatically displayed. The users can obtain more information about a particular element of the query results by clicking on that element and getting desired details-on-demand.

Figure 2.6. HomeFinder
http://www.cs.umd.edu/hcil/spotfire/
Both of them apply tight coupling to process the dynamic queries. A user’s operation on controlled elements directly invokes queries to a back-end database and gets continuous display of results for the user to explore.

2.3.4.3 Table Lens and Others

Table Lens [27] (refer to Figure 2.8) looks like an Excel spreadsheet which applies focus + context. It is very interesting from a visualization viewpoint and may reveal some pattern from provided information. However, it can only visualize numerical or enumerable information.

Spotfire [15] is one of the most popular visualization tools to handle general types of data. The Spotfire DecisionSite Browser 7.0 can be downloaded from http://www.bioinfo.utmb.edu/spotfire/Spotfire-DecisionSite-Browser.exe, with some film data inside as a demo.
2.3.5 Visual Object Visualization

This type of system presents visualization based on a preprocessed package of data, or *visually enhanced objects* to improve object interaction with users. A good sample of this type of object is a human body or a book. Visual data are directly bonded to the information inside the package. Figure 2.9 is an example system using human anatomic data packaged as a visualization [14]. In this package, all the information itself is an image of a slice of a human body. The visualization tool provides the user with the input of the height (using the mouse to decide). The corresponding image will be shown with the user’s input on the right upper corner.

Figure 2.8. Table Lens
Figure 2.9. Visible Human
http://www.cs.umd.edu/hcil/visible-human/
2.4 Observations

Based on our literature review, we can make the following observations:

- There are more and more concerns about interoperability in the field of DLs.
- To find proper systems to visualize DL content is of growing concern.
- The underlying program logic varies vastly among DL systems.
- The users’ needs to access various type of DL systems are increasing every day.

These observations lead me to think that visualization systems and digital libraries should be weakly coupled, so that more free interaction in all suitable directions would be possible.

Furthermore, since there are many DL systems and many visualization systems, it would be nice if a general approach were designed so that any DL and any VIS could be connected, preferably with minimal new programming effort. This goal is the focus of subsequent chapters.
CHAPTER III
DESIGN

3.1 Problems and Proposed Solutions

Defining interoperability is difficult. It is clear that this is still a research problem on its own, and one that merits continued attention. There is a broad spectrum of views on how interoperability should be achieved between Digital Libraries and Visualization systems. At one end of the spectrum is relying almost entirely on human intelligence to provide any coherence of content they need to exchange. The relationship between tightly coupled dynamic-query systems and their corresponding databases can be viewed as this. At the opposite end of the spectrum is deep semantic interoperability [40]. The precise definition of deep semantic interoperability is the subject of some debate. It deals with the ability of a user to access, consistently and coherently, similar (though autonomously defined and managed) classes of digital objects and services, distributed across heterogeneous repositories, with federating or mediating software compensating for site-by-site variations. It also extends beyond passive digital objects to actual services offered by specific digital library systems. Deep semantic interoperability is a "grand challenge" research problem; it is extraordinarily difficult, but of transcendent importance, if digital libraries are to live up to their long-term potential. An intermediate position between these two extremes advocates primarily syntactic interoperability (the interchange of metadata and the use of digital object transmission protocols and formats based on this metadata rather than simply common navigation, query, and viewing interfaces) as a means of providing limited coherence of content, supplemented by human interpretation.

As previously discussed, the decoupling of Digital Libraries and Visualization Systems is never an easy or trivial problem to solve.

3.1.1 Problems
Let’s analyze our problem in a more human way. Suppose there is a huge unknown world of DLs and VIS’. What are the possible scenarios that can happen?

**Scenario 1**

A very successful VIS is supporting some visdata format beautifully. I am a DL, and want to be able to use this VIS to visualize some of my data... Can I? How can I?

**Scenario 2**

A very successful DL is supporting multiple public metadata formats at ease. I am a VIS, and want to be able to provide some visualization services about its data... Can I? How can I?

**Scenario 3**

Some very familiar DLs and some very familiar VIS tools are available; I am a user, and I want to choose at my will to use some VIS tools to support some data from some DL… Can I? How can I?

These three scenarios present the desperate needs in making the VIS and DL systems accessible and understandable to each other. However, there seems to be no viable or clear solution existing for this problem, currently.

### 3.1.2 Proposed Solutions

In our design, the proposed protocol should achieve or present the following features in order to be successful.

- It should enable interoperability.
  
  This is the essential part we are facing. This feature is achieved if the protocol can answer the questions listed in the three scenarios above.
- It should be lightweight.
We don’t want to have a heavy and complex protocol. A suitable lightweight protocol, which provides enough functions and enables the communication process, will be our first choice as a solution to this problem. Also a lightweight solution enables flexible implementation, which means that new and advanced technologies can be added freely or utilized with their emergence. Lastly a lightweight solution usually expects minimum requirements from the systems.

- It should be an extension to the OAI protocol for metadata harvesting.

This prerequisite is set from practical considerations. The OAI protocol is a very successful protocol and widely used among DL systems to enable the exchange of digital objects. If this new protocol can be an extension to OAI then it would be easier for it to gain widespread acceptance. Also, it would be easy to generate a standard to talk to an OAI-compliant DL using the new protocol, because now the request methods to DLs are homogeneous.

The VIDI protocol fulfills all the above prerequisites, and is the lightweight protocol proposed to solve the interoperability problem commonly existing between the VIS and DL systems. Besides, it has some other unique design features we will discuss, when we provide more details.
3.2 VIDI Protocol Design Concerns

3.2.1 Use of XML, XSLT and schema

The VIDI protocol utilizes XML and its related technologies in its design. To understand why this decision is made, we have to first understand some concepts and answer the following questions:

What is XML?
XML stands for eXtensible Markup Language [34]. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. Since the introduction of XML by the World Wide Web Consortium (W3C), it has emerged as the new standard in data communication. Even though it is still in its infancy, XML is being embraced by major corporations the world over.

Why use XML?
Why is there such a tremendous draw towards XML and why are we applying it in our protocol? The beauty of XML is the simplicity of how it organizes data. It does so using tags. Each tag name identifies the data within it. A tag may be unique, or it may be a single entity in a collection. Each XML file has a "start" and an "end" tag; the data between the start and the end tags may be text or may contain other tags. A tag may have one or more attributes to either provide uniqueness or additional data.

[http://www.devx.com/premier/mgznarch/javapro/2001/01dec01/cs0112/cs0112-1.asp]

There are many different technologies that have generated a lot of “hype”, but only a handful bring serious attention, and seldom can any compete with XML on overall abilities. Its flexible user-defined document types make up for the flaw of HTML, which is but a single type, and has already been much abused for tasks it was never designed for. XML itself is very simple, but it allows you to describe any information in a platform- and vendor-neutral way, and then easily share this information with anyone. Major corporations, like Microsoft, IBM, and Sun, all put XML on the focal position of their
business. It enables the further interoperability of data exchange between applications. There is no question that, because of the goals of the VIDI protocol, we should choose XML to be the format for our data exchanges.

The price of the flexibility of XML is learning all of the associated technologies that make XML truly powerful.

What is XSLT?
XSLT is a language for transforming XML documents into other formats of documents [39]. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another document that uses the formatting vocabulary.

Why use XSLT?
The base function of the VIDI protocol is to provide communication between Digital Libraries and Visualization Systems. The exchange of both system information and data information requires the transformation between different formats of data. To enable the interoperability between DL systems and VIS systems, XSLT is our first choice to do some lightweight transforming.

If the transformation is not so simple, we may use some other channels to enable it, such as HTTP request and response to an external site, which accepts the input data, does the transforming internally, and then outputs the transformed data for further use.

What is Schema?
XML Schemas define structures and constraints of XML documents [28]. They express shared vocabularies and allow machines to carry out rules made by people. Requirements are for constraints on how the component parts of an application fit together, the document structure, attributes, data type, and so on.
Why use Schema?
There is the construct of Document Type Definition (DTD) that has similar function as schema, to declare constraints on the use of markup and enable automated processing of XML documents. Also there are new standards for defining constraints emerging every day, for example, RELAX NG [29], which claims to be the next generation schema language for XML. We use Schema because of its public acceptance and prevailing support.

3.2.2 Use of HTTP Request and Response

HTTP is the protocol used between Web browsers and servers. It is an application layer protocol where the underlying layers provide reliable pipe connections (like TCP). It is also a client-server based protocol. The client sends an HTTP request to the server, specifying which file to get. After receiving the request, the server finds the file and sends it to the client in an HTTP response. Version 1.1 of the protocol provides features like persistent connections, cache/proxy support, and so forth. Readers can find a detailed description of HTTP version 1.1 in [8].

Due to the popularity of the World Wide Web, almost all online information systems (including DL systems) use HTTP to provide their user interfaces through browsers to interact with their users. An important feature of HTTP in those systems is that HTTP is a “stateless” protocol. No relationship can be assumed between different requests even if they are sent from the same client to the same server. This can be an obstacle since most online searching systems are context sensitive and allow users to complete a search function over the course of multiple interactions with the system. Systems now have to use some mechanism to maintain state information on top of HTTP between different client requests. However this is usually deemed as a benefit, because it makes the communication process as simple and neat as possible.

The VIDI protocol chooses to use HTTP request and response as its major communication channel. This can to the largest extent reutilize the existing DL systems’
resources of functionality such as supporting users to directly contact a server by HTTP. This asks for the least modification and exerts the least burden for fulfilling our task of achieving interoperability. Consequently, as is required for the lightweight VIDI protocol, the implementing software of this protocol must support HTTP communication protocols. Other modes of data service would be supported by talking to other network sockets or network services, or by calling the data server executable by a different name. We will explain HTTP request and response in more detail in the protocol specification that follows.
3.3 VIDI Protocol

“VENI, VIDI, VICI.” (I Came; I Saw; I Conquered.)

-- Gaius Julius Caesar, 48 BC

The term VIDI originates from taking the initial two letters from both visualization systems and digital libraries, and concatenating them together. In ancient ROME, it also has the meaning of “I saw”, which vividly represents our goal of “see the digital libraries”.

The VIDI protocol described in this document has an interoperability and scalability framework with two classes of participants:

Data Providers administer DLs that support the VIDI protocol as a means of exposing metadata about the content in their systems and based on which they generate new metadata formats (called visdata formats being used in visualization tools) as needed;

Visualization Providers issue VIDI protocol requests to the systems of data providers and use the returned visdata as a basis for visualizing and presenting results to users.

3.3.1 Protocol Entities Definitions

Metadata Format

Metadata is a machine understandable representation of data that we can retrieve from a DL system. The manifestation format of the metadata is called the metadata format. For one item inside a DL system, there might be multiple metadata formats, which means the DL can present the item in multiple ways. In a word, metadata format is a format in which a DL system presents its data.
Visdata Format

*Visdata* is the machine understandable representation of data that we can visualize in a VIS system. The manifestation format of the visdata is called the *visdata format*. In a word, *visdata format* is a format a VIS system understands regarding how to present the data.

Transformer

A *transformer* is essentially a tool to handle the transformation of data from *metadata format* to *visdata format*. A transformer can be as simple as an XSLT stylesheet that would transform some defined specific *metadata format* in XML that a DL supports to some *visdata format* in XML that a VIS supports. An XSLT stylesheet can be a very common transformer because usually the *visdata format* is a subset of the corresponding *metadata format*. Also the case exists that some simple operations need to be done in the stylesheet to get some desired attribute. A transformer also can be as complex as an application that accepts the data in some specifically defined *metadata format* as input and returns the data as output in some defined specific *visdata format*. VIDI transformers – referred to for the remainder of this document as transformers – are organized into the following parts:

- *metadataFormat*. The metadata format. All the metadata formats that are supported by one DL system can be listed by the *ListMetadataFormats* command without any parameter.
- *visdataFormat*. The visdata format. All the visdata formats that are supported by one VIS system can be listed by the *ListVisdataFormats* command without any parameter.
- *baseURL*. The location of a transforming tool, either an XSLT stylesheet or a URL of an online accessible application. The DL can access this tool and use it to transform DL items with the specified metadata format designated in the *metadataFormat* field into the desired VIS items with the specified visdata format designated in the *visdataFormat* field.
Result Set

A *result set* is an XML-encoded byte stream that is returned by a DL, in response to a VIDI protocol request for *visdata formats* of a set of *records* from that DL. It is a set of *result records* indicated by a *result set identifier*. *VIDI result sets* – referred to as *result sets* for the remainder of this document – are organized into the following parts:

- **resultSetID**: The *result set identifier* sent by the VIS request. Preserving this can help the VIS system to match the returned *result set* with the corresponding query.
- **numRecords**: The number of all the *records* that are sent back in this *result set*. One function is to check if the data transferring is OK; the other is to let the VIS system know in advance about the scale of the whole *result set*. Many VIS systems deem such kind of information as critical in the visualization process, so it’s better to keep it here rather than counting every time.
- **record**: A list of *result records* described as below.

Result Record (or Record)

A *result record* is an XML-encoded byte stream that represents one transformed record from the DL. *VIDI result record* – referred to as *result record* or *record* for the remainder of this document – is in some *visdata format*.

Result Set Identifier (or ResultSetID)

A *result set identifier* is an XML-encoded byte stream that is sent by a VIS as a VIDI protocol request to retrieve a *result set* in a DL. *VIDI result set identifiers* – referred to as *result set identifiers* for the remainder of this document – are organized into the following parts:

- **uniqueID**: A unique string to retrieve a set of items inside digital libraries. This part can be as simple as a serial number that a DL returns to a user as a ticket to retrieve some ready *result set*, or can be as complex as a query string in some format that is predetermined by the user or the VIS system and the DL system.
• userAuthentication. Commonly designed to include the username and password to connect to the DL. The format of user authentication is defined inside the response from DL’s Identify.

Unique Identifier (or Identifier)

A unique identifier is a key for extracting the information from a data provider or a visualization provider. The VIDI protocol requires that a unique identifier should identify a provider to describe all the necessary information for any later communication purpose. VIDI unique identifiers – referred to for the remainder of this document as identifiers – are organized into the following parts:

• name: Either the DL or the VIS system will have a human-recognizable name so that the human users or administrators or operators can understand.

• baseURL. A path that is specified by the respective HTTP server as the service provider for VIDI protocol requests. As a minimum requirement, the baseURL specifies the Internet host and port of an HTTP server so that either the DL or the VIS system can find the other party it wants to communicate with from the value of this field.

• lastModified. To indicate the last modified time for the data formats (the metadata formats on the DL, or the visdata formats on the VIS). This field is critical to decide if it is possible to skip the process of exchanging some system information. (For details of usage please refer to the updating scenarios below.)

• protocolVersion. The version of the protocol supported by both the VIS system and the DL system.

• adminEmail: the email address of the administrator of this system.

• userAuthenticationFormat: The path of the schema file which specifies the format that the user will use to contact the DL system. When the user wants to view the result set through the VIS system, the VIS has to send the request with the userAuthentication field filled with the value conforming to the format defined by the schema. When this field is empty, it means the user can contact the DL without any authentication information.
• *encryptionMethod:* The DL system might require all information to be sent in encrypted mode. When this field is empty, it means the user can contact the DL with a plain request and retrieve a plain response. If this field is not empty, the VIS should follow its rule to encrypt requests and decrypt the responses.

• *description:* In addition to all the fields above, the system can choose to have extra description containers to convey more information necessary to be known.

### 3.3.2 Protocol Features

Besides the features defined for the proposed solution (refer to section 3.1.2) and the design concerns of the protocol (refer to section 3.3.2), the VIDI protocol also has the following features:

#### HTTP Embedding of VIDI Requests

The VIDI protocol requests are expressed as HTTP requests. A typical implementation uses a standard Web server, such as Apache, that is configured to dispatch the VIDI requests to the software handling these requests. The remainder of this section describes the aspects of the protocol that are specific to the HTTP embedding.

**HTTP Request Format**

VIDI protocol requests may be expressed using either the HTTP GET or POST methods. There is a single base URL, *baseUrl*, for all requests. The *baseUrl* should specify the Internet host and port of an HTTP server acting as a DL or VIS. Typically, the *baseUrl* will also specify a *path* that is specified by the respective HTTP server as the handler for VIDI protocol requests. Note that the composition of this path and its presence or absence is determined by the configuration of the system's HTTP server.

In addition to the *baseUrl*, all requests contain a list of *keyword arguments*, which take the form of key=value pairs. Arguments may appear in any order and multiple arguments should be separated by ampersands (&). Minimally each VIDI protocol
request has one key=value pair that specifies the VIDI protocol request issued by the client:

key is the string 'verb';
value is one of the defined VIDI protocol requests.

The number and nature of additional key=value pairs is dependent on the arguments for the individual protocol request.

Libraries such as libwww-perl will decode GET and POST requests seamlessly; the application does not even need to know which method was used. The advantage of POST is that there are no limitations on the length of the arguments. The “POST” method is recommended for the VIDI protocol implementation.

HTTP Response Format

Responses to protocol requests are formatted as HTTP responses, with appropriate HTTP header fields. Every VIDI protocol request returns a Content-Type of text/xml. Similarly, the success or error status of each protocol request is returned via the HTTP Status-Code.

Dates and Times

Both the dates and times used in the VIDI protocol are following standards.

The date used in the VIDI protocol is encoded using the "Complete date" variant of ISO8601 (http://www.w3.org/TR/NOTE-datetime). That format is YYYY-MM-DD where YYYY is the year, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 28 or 29 or 30 or 31, depending on the length of the month and whether it is a leap year. For example: 1957-03-20 is March 20th 1957.

The time used in the VIDI protocol is encoded using the "complete date plus hours, minutes and seconds" variant of ISO8601 (http://www.w3.org/TR/NOTE-
datetime). That format is YYYY-MM-DDTh:mm:ssTZD, with the following meanings:

- YYYY, MMM, DD are the same as for date above.
- T is the character "T",
- hh are the hours (between 00 and 23), mm are the minutes (between 00 and 59) and ss the seconds (between 00 and 59).
- TZD is an indication of the time zone of the repository, expressed as an offset from UTC. A time zone offset of "+hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes ahead of UTC. A time zone offset of "-hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes behind UTC.

For instance, 1957-03-20T20:30:00+00:00 is UTC 8:30 PM on March 20, 1957.

**Prefix and Schema**

There are two types of data structures: one is the *metadata format* on the DL system; the other is the *visdata format* on the VIS system. The *Visdata format* is normally seen as a subset of the *metadata format*, in which case the records in the visdata format can be transformed from the records in the metadata format using an XSLT stylesheet. The two types of data structures have very similar functions although they are on the different systems. They are discussed without differentiation below as *data format*. The terms related to them are discussed without differentiation below with the prefix of *data*.

Each *data format* that is used in the transformation process (either as the input or as the output) defined by the VIDI protocol, is identified within the protocol requests to a DL system or a VIS system by a *data prefix* and across multiple DLs or VISs by the URL of a *data schema*.

The *data prefix* must be a non-space embedded string consisting of alphanumeric characters or an underscore [_] character. All the *data prefixes* are preceded by the string “VIDI” (for example “VIDI_Identify”) in this document for the purpose of clarity.
The data schema is an XML Schema that may be used for providing structure constraints for the data included in the record. It is highly recommended that the schema include comments that can aid in human understandability.

Certain communities may adopt guidelines for sharing of data prefixes and/or data schema throughout the community. Such guidelines are outside of the scope of the VIDI protocol.

**Dual Usages of Commands**

All of the commands defined in the protocol, except the `RequestResultSet` command (refer to section 3.3.3), have options related to the two ways of implementation:

- They can be on the server side, providing the information only when requested by the other party; or
- They can be on the client side, on their own initiative providing the information to the other party.

This feature enables even more flexible system implementation – the system now can itself trade off between simplicity and functionality. The server side case is the normal case, which is common in many other protocols. This can be achieved by implementing VIS as a counterpart server side, to help with some of the functionality. It can provide information if there is anything missing in the processing of some request, by adding a little bit of complexity to provide server services. However, the availability of the second case allows us to implement as purely a client side routine to achieve simplicity. This client side implementation results in loss of some of the functionality because now if there is any system information missing, it can only get the notification of an error message. In our prototype implementation, I chose to implement in client-server structure. However, if the VIS system is more complex, standalone, and more powerful, it will prefer a server-server implementation.
The VIDI protocol achieves this client side command feature by defining the following rules when a command is to be implemented as a client side command:

- The requester will put the expected XML response as a value into one field. The expected XML response means the response the current requester will return when it acts as a server and is requested by this command. E.g., normally a VIS is requested by a ListTransformers command from a DL, and will return some XML response with transformers contained. However, this command also can be sent from the VIS, which acts as a requester now, and sends ListTransformers directly to the DL with transformers as a parameter. The parameter names are different depending on the different commands. See Table 3.1 for a listing. The only difference between a parameter sent in the client side command and an expected XML response for the server side command is that now the <responseTime> element value in the parameter is set to the timestamp when the request is sent out (maybe actionTime will be more appropriate).

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify</td>
<td>identifier</td>
</tr>
<tr>
<td>ListMetadataFormats</td>
<td>metadataFormats</td>
</tr>
<tr>
<td>ListVisdataFormats</td>
<td>visdataFormats</td>
</tr>
<tr>
<td>ListTransformers</td>
<td>transformers</td>
</tr>
</tbody>
</table>

Table 3.1 Field names for different commands

- Besides the fields listed above, the request needs to attach the sender’s name and baseURL address. This is mandatory, because if there is some system information missing, the other party (the receiver) can still try to make requests for the missing data by using this name and baseURL.

Refer to section 4.2 for an example of the ListTransformers command implemented as a client side command in the ENVISION-ODL prototype.
3.3.3 Protocol Commands Summary

Five commands are identified and defined in the VIDI protocol. They are listed below with a brief summary and usage notes.

**Identify**

This verb is used to retrieve system information about a DL or a VIS necessary for the protocol communication, including the administrative, identity, security, and update control information. VIS systems often are simpler and usually do not provide security information. For a DL system, Identify specifies requirements for later conversations.

**ListMetadataFormats**

This verb is used to retrieve the *metadata formats* available from a DL.

**ListVisdataFormats**

This verb is used to retrieve the *visdata formats* available from a VIS. It is the counterpart of the ListMetadataFormats request on the DL side.

**ListTransformers**

This verb is used to request the *transformers* that a VIS system supports to transform the *metadata format* into the *visdata format*. Some optional arguments are available to restrict the response *transformers* for specified use. If all arguments are omitted, by default the response returns all the *transformers* that the VIS system supports.

**RequestResultSet**

This verb is used by a VIS system to request the *result set* from a DL system, using an argument of *resultSetID*. This required argument restricts the command to return the *result set* corresponding to this *resultSetID*. 
3.3.4 Protocol Commands Breakdown

This section lists the protocol requests – verbs – defined in the VIDI protocol. There are two sides – the DL systems and the VIS systems – that implement these requests. Some requests are suitable for both sides, i.e., Identify, while others are only suitable for one specified side. This will be indicated when explaining each request type. The documentation for each protocol request is organized as follows:

- Each section header corresponds to the token used to specify the request as the required verb argument to an HTTP protocol request.
- The list of additional arguments for the request. Arguments are of two types:
  - required, the argument must be included with the request (the verb argument is always REQUIRED).
  - optional, the argument may be included with the request.
- The format of the response defined by means of an XML schema.
- One or more example requests and the corresponding responses, with explanatory notes if appropriate.

3.3.4.1 Identify

Arguments:
None

Response Format:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:vidi="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify"
    targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

<annotation>
  <documentation>
    Schema to verify validity of responses to Identify VIDI-protocol request.
  </documentation>
</annotation>
```

- 42 -
<element name="Identify" type="vidi:IdentifyType"/>

<!-- response to Identify-request -->

<complexType name="IdentifyType">
  <sequence>
    <element name="responseTime" minOccurs="1" maxOccurs="1" type="dateTime"/>
    <element name="requestURL" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="name" minOccurs="1" maxOccurs="1" type="string"/>
    <element name="baseURL" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="lastModified" minOccurs="1" maxOccurs="1" type="dateTime"/>
    <element name="protocolVersion" minOccurs="1" maxOccurs="1" type="string"/>
    <element name="adminEmail" minOccurs="1" maxOccurs="1" type="string"/>
    <element name="userAuthenticationFormat" minOccurs="0" maxOccurs="1"
      type="vidi:userAuthenticationFormatType"/>
    <element name="encryptionMethod" minOccurs="0" maxOccurs="1" type="string"/>
    <element name="description" minOccurs="0" maxOccurs="unbounded"
      type="vidi:descriptionType"/>
  </sequence>
</complexType>

<complexType name="descriptionType">
  <sequence>
    <any namespace="##other" processContents="lax"/>
  </sequence>
</complexType>

<complexType name="userAuthenticationFormatType">
  <sequence>
    <any namespace="##other" processContents="lax"/>
  </sequence>
</complexType>

</schema>

This schema is available at
http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify.xsd

Example:

- Request:

- Response:
The response to this protocol request includes the following elements that must be provided:

- **name**: a human readable name for the DL or VIS, for example “ODL”.
- **baseURL**: the BASE_URL for the DL or VIS, only have this so the other party can send it a request.
- **lastModified**: the timestamp for the last time the data structure changed (corresponding to metadata formats on DL side, and visdata formats on VIS side); this field is used to notify others of the changes.
- **protocolVersion**: the version supported of the VIDI protocol.
- **adminEmail**: the email address of the administrator of the system, so that problems can be reported.

There are optional fields to provide user logging information and encryption information:

- **userAuthenticationFormat**: this field provides the necessary information to prompt the VIS system on how to use its service. The normal case is when the DL asks the VIS to provide a user name and password.
- **encryptionMethod**: the encryption method that both parties agree to use for the request and response for data and user information.

In addition, the response may contain a list of description containers, which provide an extensible mechanism for communities to describe their repositories. The description container, could -- for instance -- be used to include collection-level metadata in the response to the Identify request. Each description container must be accompanied by the URL of an XML schema, which provides the semantics of the container.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Identify
    xsi:schemaLocation="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_Identify">
```
3.3.4.2 ListMetadataFormats

Arguments:
None

Response Format:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:vidi="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListMetadataFormats"
   targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListMetadataFormats"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <annotation>
     <documentation>
       Schema to verify validity of responses to ListMetadataFormats VIDI-protocol request.
     </documentation>
   </annotation>

   <element name="ListMetadataFormats" type="ListMetadataType"/>

   <!-- response to ListMetadataFormats-request -->

   <complexType name="ListMetadataType">
     <sequence>
       <element name="responseTime" minOccurs="1" maxOccurs="1" type="dateTime"/>
       <element name="requestURL" minOccurs="1" maxOccurs="1" type="anyURI"/>
       <element name="lastModified" minOccurs="1" maxOccurs="1" type="dateTime"/>
       <element name="metadataFormat" minOccurs="0" maxOccurs="unbounded"
         type="vidi:metadataFormatType"/>
     </sequence>
   </complexType>

   <complexType name="metadataFormatType">
     <sequence>
     </sequence>
   </complexType>

   <complexType name="ListMetadataFormats" type="ListMetadataType"/>

   <!-- response to ListMetadataFormats-request -->
```

```
This schema is available at

http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListMetadataFormats.xsd

Example:

Request:


Response:

The response to this protocol request includes the following elements that must be provided:

- **lastModified**: the timestamp for the last time the metadata structure changed

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ListMetadataFormats
  xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListMetadataFormats">
  <responseTime>2002-03-24T15:53:00EST</responseTime>
  <lastModified>2000-01-01T11:00:00-05:00</lastModified>
  <metadataPrefix>visdc</metadataPrefix>
  <schema>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/OAI_Record.xsd</schema>
  <metadataNamespace>http://www.openarchives.org/OAI/1.0/OAI_Record</metadataNamespace>
</ListMetadataFormats>
```
3.3.4.3 ListTransformers

Arguments:

- **identifier**: an *OPTIONAL* argument that restricts the response transformers to those that will work for the DL with the specified identifier. This identifier is expected to have the identical format as an Identify response that would be returned by this DL, with minor value changes such as requestURL set to the DL URL, and the responseTime set to the request time of this ListMetadataFormats protocol request. So it has an XML header and conforms to the VIDI_Identify.xsd schema described above. Since this identifier contains information such as DL name, DL URL, and the lastModified field, the VIS that handles this request can do necessary handshaking or commence with an update procedure. Refer to section 3.3.4.4 to read more about the response-format parameters.

- **metadataFormat**: an *OPTIONAL* argument that restricts the transformers to those that transform only from the unique metadata format with the specified name.

- **visdataFormat**: an *OPTIONAL* argument that restricts the transformers to those that transform only to the unique visdata format with the specified name.

Response Format:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:vidi="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListTransformers"
   targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListTransformers"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

<annotation>
   <documentation>
   Schema to verify validity of responses to ListTransformers VIDI-protocol request.
   </documentation>
</annotation>

<element name="ListTransformers" type="vidi:ListTransformerType"/>
```
<!-- response to ListTransformers-request -->

<complexType name="ListTransformerType">
  <sequence>
    <element name="responseTime" minOccurs="1" maxOccurs="1" type="dateTime"/>
    <element name="requestURL" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="transformer" minOccurs="0" maxOccurs="unbounded"
      type="vidi:transformerType"/>
  </sequence>
</complexType>

<complexType name="transformerType">
  <sequence>
    <element name="metadataFormat" minOccurs="1" maxOccurs="1"
      type="vidi:metadataFormatType"/>
    <element name="visdataFormat" minOccurs="1" maxOccurs="1"
      type="vidi:visdataFormatType"/>
    <element name="xslt" minOccurs="1" maxOccurs="1" type="anyURI"/>
  </sequence>
</complexType>

<complexType name="metadataFormatType">
  <sequence>
    <element name="metadataPrefix" minOccurs="1" maxOccurs="1"
      type="vidi:dataPrefixType"/>
    <element name="schema" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="metadataNamespace" minOccurs="0" maxOccurs="1"
      type="anyURI"/>
  </sequence>
</complexType>

<complexType name="visdataFormatType">
  <sequence>
    <element name="visdataPrefix" minOccurs="1" maxOccurs="1"
      type="vidi:dataPrefixType"/>
    <element name="schema" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="visdataNamespace" minOccurs="0" maxOccurs="1"
      type="anyURI"/>
  </sequence>
</complexType>

<complexType name="dataPrefixType">
  <restriction base="string">
    <pattern value="[a-zA-Z0-9_]"/>
  </restriction>
</complexType>
Example:

Request:

Response:

<?xml version="1.0" encoding="utf-8" ?>
<ListTransformers
  xsi:schemaLocation="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListTransformers.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListTransformers">
  <responseTime>2002-03-25T18:19:13EST</responseTime>
  <requestURL>http://tuppence.dlib.vt.edu:8080/ENVISION/servlet/VIS?verb=listtransformers</requestURL>
  <transformer>
    <metadataFormat>
      <metadataPrefix>visdc</metadataPrefix>
      <schema>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/OAI_Record.xsd</schema>
      <metadataNamespace>http://www.openarchives.org/OAI/1.0/OAI_Record</metadataNamespace>
    </metadataFormat>
    <visdataFormat>
      <visdataPrefix>enva</visdataPrefix>
      <schema>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva.xsd</schema>
      <visdataNamespace>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva</visdataNamespace>
    </visdataFormat>
    <xslt>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/recordtoenva.xsl</xslt>
  </transformer>
</ListTransformers>
3.3.4.4 RequestResultSet

Arguments:

- **baseURL**: a *REQUIRED* argument that specifies the request URL for the VIS which sends the request. This field becomes necessary if there is something wrong in this request, e.g., the DL does not have records about this VIS. In this case, the DL can retrieve this field from the request, and sends out the request for Identify and any other related information.

- **name**: an *OPTIONAL* argument that specifies the name for the VIS which sends the request. If DL has the VIS on record, name might be the optimal way for it to retrieve information about it. Also it suits for the case that VIS acts totally as a client and does not really need a baseURL.

- **visFormat**: a *REQUIRED* argument to indicate in which visdata format the VIS wants the result set to be sent from the DL. DL will base on this value retrieve transformers to transform raw sets into desired formats. The value of this field is corresponding to the visdataPrefix field inside the visdataFormat.

- **transactionID**: an *OPTIONAL* argument that specifies the transaction this request is in. This works for DLs that need maintaining user logging information.

- **resultSetID**: a *REQUIRED* argument that specifies the unique identifier for which the result set is being requested. The returned result set will be transformed and sent out as response after the DL has all necessary information from VIS.

Response Format:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:vidi="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_RequestResultSet"
    targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_RequestResultSet"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <annotation>
        <documentation>
            Schema to verify validity of responses to RequestResultSet VIDI-protocol request.
        </documentation>
    </annotation>
</schema>
```
<element name="RequestResultSet" type="vidi:RequestResultSetType"/>

<!-- response to RequestResultSet-request -->

<complexType name="RequestResultSetType">
  <sequence>
    <element name="responseTime" minOccurs="1" maxOccurs="1" type="dateTime"/>
    <element name="requestURL" minOccurs="1" maxOccurs="1" type="anyURL"/>
    <element name="resultSetID" minOccurs="1" maxOccurs="1" type="string"/>
    <element name="numRecords" minOccurs="1" maxOccurs="1" type="decimal"/>
    <element name="record" minOccurs="0" maxOccurs="unbounded"
      type="vidi:recordType"/>
  </sequence>
</complexType>

<complexType name="recordType">
  <sequence>
    <any namespace="##other" processContents="lax"/>
  </sequence>
</complexType>

</schema>

This schema is available at
http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_RequestResultSet.xsd

Example:

Request:
(for better readability here, it is not shown in appropriate URL encoded format)

Response:

  <?xml version="1.0" encoding="utf-8" ?>
  <!RequestResultSet
  xsi:schemaLocation="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_RequestResultSet.xsd"
  xmlns:dcl="http://purl.org/dc/elements/1.1/"
  xmlns:oai="http://www.openarchives.org/OAI/1.0/OAI_ListRecords"
The Effects of Humor on Cognitive Learning in a Computer-Based Environment

Author: Whisonant, Robert Dowling

Year: 1998

Categorization: Curriculum and Instruction
3.3.4.5 ListVisdataFormats

Arguments:
None

Response Format:
<schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:vidi="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListVisdataFormats"
       targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListVisdataFormats"
       elementFormDefault="qualified"
       attributeFormDefault="unqualified">

   <annotation>
   <documentation>
       Schema to verify validity of responses to ListVisdataFormats VIDI-protocol request.
   </documentation>
   </annotation>

   <element name="ListVisdataFormats" type="ListVisdataType"/>

   <!-- response to ListMetadataFormats-request -->

   <complexType name="ListVisdataType">
   <sequence>
       <element name="responseTime" minOccurs="1" maxOccurs="1" type="dateTime"/>
       <element name="requestURL" minOccurs="1" maxOccurs="1" type="anyURI"/>
       <element name="lastModified" minOccurs="1" maxOccurs="1" type="dateTime"/>
       <element name="visdataFormat" minOccurs="0" maxOccurs="unbounded" type="vidi:visdataFormatType"/>
   </sequence>
   </complexType>
<complexType name="visdataFormatType">
  <sequence>
    <element name="visdataPrefix" minOccurs="1" maxOccurs="1" type="dataPrefixType"/>
    <element name="schema" minOccurs="1" maxOccurs="1" type="anyURI"/>
    <element name="visdataNamespace" minOccurs="0" maxOccurs="1" type="anyURI"/>
  </sequence>
</complexType>

This schema is available at
http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListVisdataFormats.xsd

Example:

Request:
http://tuppence.dlib.vt.edu:8080/envision/servlet/vis?verb=listvisdataformats

Response:
The response to this protocol request includes the following element that must be provided:

• lastModified: the timestamp for the last time the visdata structure changed.

<?xml version="1.0" encoding="utf-8" ?>
<ListVisdataFormats
  xsi:schemaLocation="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListVisdataFormats.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ListVisdataFormats">
  <responseTime>2002-03-24T15:53:00EST</responseTime>
  <requestURL>http://tuppence.dlib.vt.edu:8080/envision/servlet/vis?verb=listvisdataformats</requestURL>
  <lastModified>2000-01-01T11:00:00-05:00</lastModified>
  <visdataFormat>
    <visdataPrefix>enva</visdataPrefix>
    <schema>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva.xsd</schema>
    <visdataNamespace>http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva</visdataNamespace>
  </visdataFormat>
</ListVisdataFormats>
3.3.5 Protocol Commands Scenarios

Not only the data transfer requests and responses, but also the system information exchange requests and responses are defined in the VIDI protocol. Among the five commands defined in the VIDI protocol, only one command `RequestResultSet` is used for query data transferring. All others are system information exchange commands, which fall into two categories:

- Mandatory system information exchanging:
  
  **Identify**.

  This command is mandatory because whenever a session between a VIS system and a DL system is set up, this command needs to be issued. Depending on the implementation structure – Client-Server structure, or Server-Server structure – the **Identify** from one side or both sides will be called. The two commands decide whether the optional system information exchanging commands (see below) need to be issued.

- Optional system information exchanging:
  
  **ListMetadataFormats, ListVisdataFormats**, and **ListTransformers**.

  These commands are optional and their usages depend on the process of the mandatory system information exchanges. Please refer to the “Updating Scenarios” Section below for more details.

The basic reason to separate them into two categories is to skip unnecessary system information exchanges. Identifiers and all other system information from either side can be saved in the history. An **identifier** will indicate if modification of data structures (i.e., the **metadata formats** on the DL side or the **visdata formats** on the VIS side) has happened. Only due to modification will optional system information exchange occur, i.e., extra communication and information updating.
When the relationship between the VIS and the DL system is not purely a Client-Server relationship, either party can request of each other the specific information it needs.

Two scenarios are listed below to help developers understand how the VIDI protocol can be employed.

### 3.3.5.1 Updating Scenarios

This is the scenario to decide if the extra system information exchanges are necessary or not.

The **Identify** for both the VIS system and the DL system will return an identifier with a “last modified (abbr. to l-m)” field to indicate the need to update information kept in the history of both sides. For the DL system, the “l-m” field corresponds to the last time it changes its *metadata formats* (possible reasons include: support new *metadata formats*; stop supporting some *metadata formats*; modifying the schemas of some *metadata formats*; etc.). For the VIS system, it is for the “*visdata formats*”. Just as we mentioned in Chapter 3.3.2, two structures are supported in implementation, both Client-Server and Server-Server. Next we will explain the updating scenarios separately on these two structures.

1. **Client-Server Structure:**
   
   This structure enables simplicity. In this structure the VIS system acts purely as a client. It only sends requests to the DL system, but does not accept any requests from the DL system. In this case, the system information exchange scenario can be like this:
   
   (i) In the beginning of the process, the VIS sends **Identify** request to the DL and gets the response.

   (ii) The VIS figures out from the “l-m” field in the response that the metadata formats on the DL side have already changed since last access, by comparing the value with the stored copy, or it figures out
that it does not have the history of the last access to this DL system: It will do the following steps of updating:

First, VIS requests \textit{ListMetadataFormats} from DL;
Second, VIS sends \textit{ListTransformers} to DL.

(iii) When the \textit{visdata formats} on the VIS side are changed, the VIS has to clean all the history, so that the next time any DL accesses to it, the transformers will be generated once to reflect the changes of \textit{visdata formats}. This is because this structure causes the DL has no channel to see the changes from the VIS side. If the VIS system is very small (can be restarted easily), or not easy to change, this structure will be quite suitable. If during the process the \textit{metadata formats} on the DL side have changed, because of the Client-Server structure, the DL cannot notify the VIS about the changes, so the only thing it can do is to wait till next time.

(2) Server-Server Structure:

This structure enables functionality. In this structure, either system has complete knowledge about the other system, and can request necessary system information when it is missing. Usually in this case the VIS system is very powerful, and can support multiple \textit{visdata formats}. In this case, it is beneficial for the DL to remember the VIS’s information, so that the DL can initiate the system information update process, without the need of VIS to clean up all its history. The updating process will be as follows:

(i) The VIS sends out the \textit{Identify} request to the DL system and gets the response from the DL. Meanwhile, the DL finds out the VIS is trying to contact it, so sends out the \textit{Identify} request to this VIS, and gets the response from the VIS.

(ii) The VIS figures out from the l-m field in the DL’s response that the \textit{metadata formats} on the DL have already changed since last access by comparing the value with the stored copy, or it figures out that it does
not have the history of the last access to this DL system. It will do the following steps of updating:
First, VIS requests `ListMetadataFormats` from DL;
Second, VIS sends `ListTransformers` to DL.

(iii) The DL figures out from the l-m field in the VIS’s response that the visdata formats on the VIS have already changed since last access by comparing the value with the stored copy, or it figures out that it does not have the history of the last access to this VIS system. It will do the following steps of updating:
First, DL requests `ListVisdataFormats` from VIS;
Second, DL requests `ListTransformers` from VIS.

(iv) When either the metadata formats on the DL side, or the visdata formats on the VIS side have changed in the process, the changed party can immediately send out the `ListMetadataFormats/ListVisdataFormats` and the `ListTransformers` request to the other party to update the system information. Of course the response to `Identify` request has to be changed accordingly. The connection between is to the largest extent maintained.

We can see that the `ListTransformers` might be called twice if both metadata formats and visdata formats have changed. But since transformers are lightweight components (besides the description of the corresponding metadata format and visdata format, there is only the baseURL description about this transformer), and the possibility that both sides change at the same time is very small, so this is acceptable. Also this guarantees that the last `ListTransformers` call will bring to the DL the newest version of transformers that would be responsive to any changes.

`ListMetadataFormats`: By using this command the DL describes its supported metadata formats. If a DL has modified its metadata formats, the “l-m” timestamp value in the `Identify` response will be the time when it most recently updated those formats. The VIS
will compare the value it gets from the response with the stored copy of the DL *Identify*
response’s corresponding field. If it’s newer or there is no history, the VIS will request
the DL to *ListMetadataFormats* so that the VIS can update its *metadata formats* copy.
Also if the DL changes in the process, it can send this command to the VIS to indicate the
changes.

**ListVisdataFormats**: By using this command the VIS describes its supported *visdata
formats*. If a VIS has modified its *visdata formats*, the “l-m” timestamp value in the
*Identify* response will be the time when it most recently updated those formats. The DL
will compare the value it gets from the response with the stored copy of the VIS *Identify*
response’s corresponding field. If it’s newer or there is no history, the DL will request
the VIS to *ListVisdataFormats* so that the DL can update its *visdata formats* copy. Also
if the VIS changes in the process, it can send this command to the DL to indicate the
changes.

**ListTransformers**: This command is used to pass the *transformers* from the VIS side to
the DL side. These can transform DL results into the *visdata formats* that the VIS can
support. Each *transformer* corresponds to one *metadata format* and one *visdata format*.
In both cases (either the DL has changed its *metadata formats* or the VIS has changed its
*visdata formats*) this command needs to be called to ensure the DL always has the newest
version of the *transformers*.

### 3.3.5.2 General Scenario

Typically, the user passes the URL of the DL to the VIS. The VIS asks the URL for
*Identify* and gets back the DL information, including address – IP, port – of service, DL
name, user authentication format, encryption method, and last modified time as well as
the collections’ administrative, protocol version, and optional community-specific
information.

After all these conversations described in the “Updating Scenario” Section, the VIS can
send *RequestResultSet* to the DL with the *resultSetID* the user has passed to it, together
with some other authentication information received from the user. DL retrieves the set based on this information and uses the transformers to transform each item in the result set to some visdata format. It then gives the transformed result set (which can be directly used by VIS) back to the VIS. The VIS can present the result to the user. To illustrate this, Figure 3.1 is shown below.

The command `ListVisdataFormats` is only used in the Server-Server structure, in which the DL can initiate the handshaking. Also it is useful in some other situations, e.g., it will facilitate the design of a new VIS. Then it is useful to know which visdata formats and/or transformers are normal for the VIS and so might be borrowed. In another situation, the DL might want to know the formats the VIS supports so that it can store data better suited for transformation or so it can select a more appropriate metadata format.

Figure 3.1. Updating + General Scenario

See the above graph to understand more about the communication process. There are 4 scenarios described in this graph, which are separated by the vertical separators. They are, from left to right, 1) the scenario using most commands in Server-Server structure; 2)
the scenario using least commands in Server-Server structure; 3) the scenario using most commands in Client-Server structure; and 4) the scenario using least commands in Client-Server structure. The numbers labeled in the graph are the order of this command in the scenarios. The “s” behind the numbers means in Server-Server structure. The “c” behind the numbers means in Client-Server structure. The “x” means in this specific scenario, this command is unused. The arrow only indicates the data flow direction (the phrases inside the parentheses are the real meaningful data that are passing between), and has nothing to do with deciding who is the requester, who is the responder.
3.4 Solutions to Problem Scenarios with VIDI Protocol

With the help of the VIDI protocol, the questions in section 3.3.1 can be answered. In other words, from the analysis, the VIDI protocol can support interoperability between the DL and the VIS systems as expected. Before we come to the real solution, we first explain some terminology used in the solution:

3.4.1 Terminology used below

- Protocol [some-command-name]: In this step, this command is used to make the communication.
- Human-[some-system]: Here the system can be DL or VIS. It means that in this step the human (programmer or administrator) needs to do this part of the work.
- Program-[some-system]: Here the system is the same as above – can be DL or VIS. It means that in this step, the program (code) needs to be changed to do this part of the work.

Prerequisite
The initial DL or VIS has implemented VIDI protocol.

3.4.2 Solutions for Scenarios

Solutions are as below for the three scenarios:

Solution for scenario 1
0: The initial DL approaches VIS, checking if it's possible to access. (Protocol Identify)
1: DL asks VIS to list its visdata format(s). (Protocol ListVisdataFormats)
2: If DL finds familiar visdata format(s) it knows how to support, it can directly asks the VIS to provide the transformers for those. (Protocol ListTransformers)
3: Otherwise if DL doesn’t find any recognizable visdata format(s), DL analyzes all the visdata formats and chooses one or several it wants to support, and writes transformers for them. (Human-DL)
3: The DL’s users now can use this VIS to visualize the DL. (Protocol RequestResultSet)
Solution for scenario 2
0: The initial VIS approaches DL, checking if it's possible to access, and if it’s possible, what is the access method. (Protocol Identify)
1: VIS asks DL to list its metadata format(s). (Protocol ListMetadataFormats)
2: VIS analyzes the metadata formats. If there are some metadata formats it can support, and selects transformers written for those metadata formats to transform into some visdata formats it knows how to support, then presents them to DL. (Protocol ListTransformers)
3: If the VIS cannot find any it supported before, it will choose one or several to support and write transformers for them, then presents them to DL. (Human-VIS)
4: VIS can now notify its users: new DL community member. Please visit using access method we obtained from step 0. (Program-VIS)
5: The VIS' users now can request some type of data from DL and VIS will visualize responses. (Protocol RequestResultSet)

Solution for scenario 3
There are two alternatives for this solution.

Solution a:

0a: User provides DL the URL of the VIS, "can I use this VIS to visualize some of your data?" (Program-DL)
1a: DL tries to identify the VIS. (Protocol Identify)
2a: DL asks for visdata formats from the VIS (Protocol ListVisdataFormats). Will go to 3a or 4a depending on visdata formats it receives.
3a: DL figures out it can support the VIS from the visdata formats; grants a dataSetID to user indicating the data set user asks for visualization. (Program-DL)
   3a_1: user gives the dataSetID to VIS. (Program-VIS)
   3a_2: VIS uses the dataSetID to retrieve data from DL. (protocol RequestResultSet)
   3a_3: VIS visualizes data. (Program-VIS)
4a: DL figures out it cannot support the VIS from the visdata formats; see solution for scenario 1.

**Solution b:**

0b: User provides VIS the URL of the DL, "can you access this DL"? (Program-VIS)

1b: VIS tries to identify the DL. (Protocol **Identify**)

2b: VIS asks for metadata formats from the DL. (Protocol **ListMetadataFormats**)

Will go to 3b or 4b depending on metadata formats it receives.

3b: VIS figures out it can support the DL from the metadata formats; pass the transformer to DL. (Program-VIS)

3b_1: VIS configures its interface properly if necessary, and asks the user to input a query on its interface and then constructs a dataSetID from it, or directly inputs dataSetID. (Program VIS)

3b_2: VIS uses the dataSetID, that the user provided or that it constructed from user's input, to retrieve data from DL (protocol **RequestResultSet**)

3b_3: VIS visualizes data (Program-VIS)

4b: VIS figures out it cannot support the DL from the metadata formats; see solution for scenario 2.
CHAPTER IV
IMPLEMENTATION

The main purpose of implementing the VIDI protocol (proposed in the last chapter), and some other efforts in this research work, is to prove that interoperability between the VIS and DL systems can be successfully achieved.

From the very beginning of the design, I kept in mind that the VIDI protocol should be flexible to suit both the VIS and the DL systems. This is guaranteed by the VIDI protocol since either side can initiate the hand-shaking communication process, and either side can choose to take more burden or less to make the communication process possible. In the VIDI prototype, even before the coding work starts, the task load that each side should take is predefined so that the implementation would be as simple as possible.

4.1 Important Implementation Decisions

The design of the VIDI protocol provides the solid foundation on which the goal of interoperability can be achieved. However, its flexible nature provides enough freedom to choose the advanced and proper, technologies and tools during the implementation. From an implementer’s perspective, there is still much room left for decision making. The key implementation choices are described in the following subsections.

4.1.1 The Role of ENVISION

First let’s analyze what can be the role of a VIS system while communicating with DL systems. From a VIS system's view, it can choose to support only one specific format of visdata as input, or it can choose to support multiple specific formats of visdata. If the VIS system decides only to support one specific format, then all the burden of making the communication possible is left to the DL systems. As I analyzed in Scenario 1 in Chapter
3, under these circumstances, a DL system will get the *visdata format*, and apply
transformations to fit DL’s data into the VIS' requirement for data. This becomes
mandatory if an existing VIS system has a very inflexible internal programming and data
structure so it is not possible for that system to support multiple *visdata formats*. But this
is of limited use, and very probably the DL cannot convert any of the data types it
supports to the specific type the VIS is asking for. It will be very helpful if the VIS can
be more flexible – in finding a good middle ground between a specific VIS and a general
purpose VIS, if it can choose the way of "Semantic Specific". It can enable a user to
access multiple DL systems through the VIDI protocol and can provide visualization
depending not only on the string type or the numeric type of data, but also on the
meaning that is conveyed by the data structure. As I discussed in Chapter 2, VIS systems
can be powerful because every VIS system uses one or more visualization techniques to
invoke better cognition from the users. These visualization techniques suit best for
presenting some features of the data. VIS should be able to support as much data with
those features as possible without failing because of some trivial limitations such as:
specific data source settings, some fields having multiple values, or some missing fields
that could not fit into the format the VIS has chosen. This is believed to be the trend in
VIS system design as well.

ENVISION is a successful dynamic-query VIS system, and it has the potential to be more
flexible than its current state. I strive to make it the first specific purpose visualization
system that can help users gain access to multiple DL systems. Being able to configure
ENVISION depending on the data formats, I am able to achieve this.

### 4.1.2 Using SAX or DOM

Implementing the VIDI protocol needs a lot of related knowledge of XML and related
technologies. While dealing with XML documents and schemas, I cannot avoid the use
of XML parsers. There are two types of XML parsers – SAX (Simple API for XML) and
DOM (Document Object Model) – designed to allow programmers to access their
information without having to write a parser in Java and also some other programming
language options (C++, Perl, Python, etc.). Both SAX and DOM were created to serve the same purpose, which is to give access to the information stored in XML documents using any programming language (and a parser for that language). However, both of them take very different approaches to do this.

DOM enables access to information stored in the XML document through a hierarchical object model. DOM creates a tree of nodes based on the structure and information and the interaction can be made with this tree of nodes. Figure 4.1 shows a simplified representation of the model. It is simplified because actually elements can contain both text data and other subelements, so even a text node is one of an element node’s children, i.e., in reality, there exists a text node with the value of “Programming for poet” as one child of the element node “title” in the above example.

Alternatively, SAX reads the parsing XML document and fires a bunch of events depending on what tags it encounters in the XML document. The parsing process can be achieved by interpreting these events with a user-defined XML document handler class, which will make sense of all the tag events and create objects in a user-defined object...
model. Its merits over DOM include that it is fast – unlike the DOM that parses everything and builds a DOM tree inside the memory before giving you the handler for access, SAX leaves you the option of selecting only the related tags to process. However, its programming interface is not as friendly as DOM. By using DOM, an XML document can be turned into a tree in three lines of code. Once there is a tree, the programmer can walk or traverse it back and forth.

For my prototype purpose, I could choose either of them. In the beginning it seemed that SAX was a more natural choice, because my XML parsing is the conversion from file stream parsing and SAX is suitable for sequentially parsing the structured data. But after some more analysis I found that the XML document I need to parse is a large XML file that contains some elements following some other schema, and this schema is dynamically fitted into the XML document and defining the structure of those elements. If I were to use SAX, then I would have to invoke some other XML document handler class towards these elements. For example, when the ODL is returning a response with records in the Dublin-Core metadata format corresponding to the ListRecord OAI request, the metadata part of the schema for its response is (note the parts namespace = “##any” and processContents = “lax”):

```xml
...<element name="metadata" minOccurs="0" maxOccurs="1"
type="oai:metadataType"/>
...
<complexType name="metadataType">
  <sequence>
    <any namespace="##any" processContents="lax"/>
  </sequence>
</complexType>
...
```

Figure 4.2. Metadata part in Dublin Core Schema

Then in the XML document, the corresponding metadata part is as below:
Figure 4.3. Metadata part in response XML

We can see that there is another element with the name “dc” inside the “metadata” element and that it is using its own schema to define each field under “dc”. Since SAX is event-driven, it is very complex to define another handler inside one handler to deal with this situation. But by using DOM, that will build a tree structure for each XML file in order to transverse much more freely from element to element, the only thing I need do is to pass different parameters to access it in my document object model. There is no need to write different document models for different formats and load different ones to do parsing one inside the other each time. I can use the most interesting new feature from DOM [http://www.onjava.com/pub/a/onjava/2002/03/06/topten.html] in my implementation as well: I can ask from a DOM tree a nodelist of specified-named elements, so that I can deal with the elements one after another iteratively in the nodelist.

4.1.3 Using CGI or Servlet

In the implementation of the VIDI protocol, it is necessary to provide some server services to deal with the HTTP requests and return responses. CGI is widely used in many online searching systems. However, it suffers from a very important feature that after the output is sent to the browser the CGI program will “die”. It will be run again by the Web server the next time a user clicks its name from the browser. This is one of the
reasons that users observe slower speed when communicating with CGI programs than when directly accessing HTML files.

Servlet performs better than CGI. The biggest performance feature of Servlets is that they do not require the creation of a new process for each request. Servlets support threads, so there can be one servlet invocation to support multiple clients. In the web server environment, Servlets can run in parallel within the same process as the server, which provides significant performance advantages over CGI. This is because Servlets only require lightweight thread context switches.

Also Servlet gains platform independence and extensibility because of its Java language feature. It can run on any platform that supports JVM (Java Virtual Machine), and it can utilize Java code from any source. Because of all these listed advantages, I use Servlet instead of CGI in my prototypes.
4.2 ENVISION and ODL

The first prototype to convey the feasibility of my design and VIDI protocol is to connect ENVISION with Open Digital Library (ODL).

As I discussed in Chapter 2, the ODL is a network of extended Open Archives Initiative (OAI) components that work together to supply the services required by information seekers. It provides a comparatively simple OAI interface to do searches and return responses. That’s one of the major reasons that I am choosing it to be my first case study DL system. Without loss of generality, I can choose the “etdunion” server site among the set of data service providers as the one that would provide the search results for my visualization purpose.

4.2.1 Analyzing Formats on ODL and ENVISION

The first working step was to understand the format of the data that could be provided by the DL side. All the OAI-compliant DLs can support the Dublin Core metadata format. Currently ODL only supports the Dublin Core format in its implementation, so I am using this type of metadata to talk to ENVISION. The schema for the Dublin Core metadata format is listed as below:

```xml
<schema xmlns=http://www.w3.org/2001/XMLSchema
   xmlns:dc=http://purl.org/dc/elements/1.1/
   targetNamespace=http://purl.org/dc/elements/1.1/
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <annotation>
       <documentation>
           Schema for Dublin Core metadata format.
           The Open Archives Initiative. 2000.
           Schema validated at http://www.w3.org/2001/03/webdata/xsv on 05-09-2001
       </documentation>
   </annotation>
</schema>
```
Next I analyzed the format of data that ENVISION could support as a basis for visualization. ENVISION was talking to MARC documents from the VT Library Catalog. Its requests and responses were directly reading or writing through socket streams to the old C/C++ version MARIAN system. Following is a sample request:
Figure 4.4. Original Envision Request format.

The character behind ENVA is actually a binary value, which is equal to the length of the query string.

From the above, I can get some sketchy idea about how to send a request and what type of response should be dealt with. A request contains the information of the SESSION ID and QUERY ID which were granted by ENVISION to keep context sensitive information and store it in the history. The FILENAME is generated by ENVISION to indicate where to save the query results. The AUTHOR, TITLE, and CONTENT contain the values the users have filled into the fields of the Query Window. Some other options also are important and are included to exert more requirements about searching, such as the total number of results it is expecting – NUMBERREQUESTED, or the way of searching – AUTHOR / TITLE / CONTENT MATCH, in this case they are all using the “ALL” option.

The response back from MARIAN also was examined as below:
RESULT
SESSIONID: "976896523"
QUERYID: "1"
FILENAME: "976896523.1.result"
RESULTTYPE: RESULTSET
STATUS: OK
RESULTSETSIZE: 25
RESULTSETID: 1
DOCUMENTID: "63545760"
URL: "(null pointer)"
RELEVANCE: 838
AUTHOR: "<tnm>Mitchell, Margaret</tnm>"
TITLE: "Gone with the wind."
YEAR: "1936"
CRCATEGORY: "United States"
FORMAT: HTML
DOCUMENTSIZE: 0
TIMESCITED: 0
DOCUMENTCITES: 0
DOCTYPE: 27
DOCUMENTID: "65300460"
URL: "(null pointer)"
RELEVANCE: 838
AUTHOR: "<tnm>Gable, Clark</tnm>";
"<tnm>Leigh, Vivien</tnm>";
"<tnm>Howard, Leslie</tnm>";
"<tnm>De Havilland, Olivia</tnm>";
"<tnm>Fleming, Victor</tnm>";
"<tnm>Mitchell, Margaret</tnm>";
"<tnm>Selznick International Pictures.</tnm>"
TITLE: "Gone with the wind"
YEAR: "1985"
CRCATEGORY: "Feature films";
"United States"
FORMAT: HTML
DOCUMENTSIZE: 0
TIMESCITED: 0
DOCUMENTCITES: 0
DOCTYPE: 9

Figure 4.5 Portion of ENVISION Result File
The result file has a very rigid structure, and so does the original parser inside ENVISION. The parser takes out lines one by one, and checks the value’s validity, then puts them into the correct format for an internal structure. Some fields can allow multiple values, such as AUTHOR, CRCATEGORY, and so on. Some fields can allow only a choice of possible values, such as FORMAT and DOCTYPE.

Based on the above analysis, I extracted the schema for the XML document that ENVISION would support, and called it the “enva” visdata format:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva"
   xmlns:enva="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/enva"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

<annotation>
<documentation>
For historical reasons, following the format the old Envision system was using and converting into XML instead of plain text
</documentation>
</annotation>

<element name="enva" type="enva:envaType"/>

<complexType name="envaType">
<choice minOccurs="0" maxOccurs="unbounded">
<element name="documentid" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="url" minOccurs="0" maxOccurs="unbounded" type="anyURL"/>
<element name="relevance" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="author" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="title" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="year" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="crcategory" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="format" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="documentsize" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="timecited" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="documentcites" minOccurs="0" maxOccurs="unbounded" type="string"/>
<element name="doctype" minOccurs="0" maxOccurs="unbounded" type="string"/>
```
Next I did the mapping between the metadata format and the visdata format. I analyzed each field’s feature on each side, and decided which pair should be the best match. In the Envision visdata format some of the most important fields that are heavily used in the interface are: Document ID, Relevance, Author, Index Terms (corresponding to ‘Category’), Item Types (corresponding to ‘Format’), Pub Year. The mapping result between Dublin Core metadata format and ENVA visdata format is listed in Table 4.1:

<table>
<thead>
<tr>
<th>Dublin Core metadata format</th>
<th>Envision visdata format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>DocumentID</td>
</tr>
<tr>
<td>Creator</td>
<td>URL</td>
</tr>
<tr>
<td>Subject</td>
<td>Relevance</td>
</tr>
<tr>
<td>Description</td>
<td>Author</td>
</tr>
<tr>
<td>Publisher</td>
<td>Title</td>
</tr>
<tr>
<td>Contributor</td>
<td>Year</td>
</tr>
<tr>
<td>Date</td>
<td>Category</td>
</tr>
<tr>
<td>Type</td>
<td>Format</td>
</tr>
<tr>
<td>Format</td>
<td>DocumentSize</td>
</tr>
<tr>
<td>Identifier</td>
<td>Timescited</td>
</tr>
<tr>
<td>Source</td>
<td>DocumentCites</td>
</tr>
<tr>
<td>Language</td>
<td>DocType</td>
</tr>
<tr>
<td>Relation</td>
<td></td>
</tr>
<tr>
<td>Coverage</td>
<td></td>
</tr>
<tr>
<td>Rights</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1. Mapping between Dublin Core metadata format and Enva visdata format

Although the Dublin Core format does not provide a “relevance” field, the ODL search engine does provide this field, which I suppose most search engines will do. They always have some method to do weighting. The interoperability between these metadata formats itself is another tricky problem that is worth deep investigation and thought. Currently I only take these at face value without going any further. We follow how ODL provides its relevance value. (The <metadata> field is holding the real Dublin Core metadata.) For example:
Because of the special importance of some fields such as “relevance” as pointed above, I find that the pure metadata format that was used between DL systems to exchange digital objects is not suitable directly for transforming without any modification. A new schema is written based on the combination of Dublin Core schema and ODL response schema, and is used as my metadata format.

4.2.2 Transforming Between Formats

After analyzing the information structures on both sides and generating the schemas, it is time to write the transformer to do the transformation. As can be seen from the design of the VIDI protocol, a transformer plays an important role in the smooth communication of the two parties in this protocol. Based on the mappings I do, I found that an XSLT stylesheet was perfectly suitable for my need. The transformation is converting one record of the visdata format into one corresponding record of the metadata format. In another words, it is not used for generating the whole response for some protocol command. However, it generates the most important part to be used in the communication – the transformed data. The recordtoenva.xsl stylesheet is attached below to fulfill the task of transformation for this situation:

```xml
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/VIDI_ResultSetSet"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dcl="http://purl.org/dc/elements/1.1/"
    xmlns:oai="http://www.openarchives.org/OAI/1.0/OAI_ListRecords">
```
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes" />

<xsl:template match="/">
  <numRecords>
    <xsl:number value="count(/oai:/oai:record)" format="1" />
  </numRecords>
</xsl:template>

<xsl:template match="oai:record">
  <record>
      <documentid>
        <xsl:value-of select="oai:header/oai:identifier" />
      </documentid>

      <xsl:for-each select="oai:metadata/dcl:dc/dcl:source">
        <url>
          <xsl:value-of select="." />
        </url>
      </xsl:for-each>

      <relevance>
        <xsl:value-of select="oai:about/oai:relevance" />
      </relevance>

      <xsl:for-each select="oai:metadata/dcl:dc/dcl:creator">
        <author>
          <xsl:value-of select="." />
        </author>
      </xsl:for-each>

      <xsl:for-each select="oai:metadata/dcl:dc/dcl:title">
        <title>
          <xsl:value-of select="." />
        </title>
      </xsl:for-each>

      <xsl:for-each select="oai:metadata/dcl:dc/dcl:date">
        <year>
          <xsl:value-of select="substring-before(string(),'-')" />
        </year>
      </xsl:for-each>

      <xsl:for-each select="oai:metadata/dcl:dc/dcl:subject">
        <xsl:value-of select="." />
      </xsl:for-each>
    </enva>
  </record>
</xsl:template>
4.2.3 Protocol Flow Graph (Abstract)

As we can see, this is the same as the 3rd scenario, e.g., the most commands in Client-Server structure, in Section 3.3.5.2 General Scenario.

4.2.4 Protocol Commands Implementation on VIS Side
In this prototype, the ENVISION’s role is that of a pure client. Consequently, there are no server services, such as the command `ListVisdataFormats` defined by this protocol, being implemented on the VIS side. However, there is a command `ListTransformers`, which normally should be presented as a server service, being implemented here as a client service. (Refer to Chapter 3 for a detailed explanation about why and how this feature is enabled in the VIDI protocol.) We can see that in the protocol flow graph above, the VIS system is merely sending requests and receiving responses, while the DL system accepts requests and sends back responses. On the VIS side, the sending requests and retrieving responses part of the protocol requires the following four commands: `RequestResultSet`, `Identify`, `ListMetadataFormats`, and `ListTransformers`. Because the VIS and the DL system can be deemed as counterparts from some perspectives, the implementation of the server services can refer to those that have been implemented on the DL side.

The protocol handling part in the ENVISION system is encapsulated as a new component separate from the original VIS system, which is referred to as “VISComponent”. For connection to one specific DL, the VIS will initialize one VISComponent with the DL’s name and DL’s URL. For each command there is a corresponding function to send the command’s request and retrieve the command’s result.

The normal processing for each command is as follows: First, the function constructs the request conforming to the VIDI protocol defined format by using a StringBuffer and input parameters (or public variable values). Second, the content in the StringBuffer is converted to a URL and the URL is sent to the specified DL URL in the form of an HTTP request. Lastly the function returns the HTTP response directly to the caller – which is the VIS system.

The commands `ListTransformers` and `RequestResultSet` need data preparation. This preparation phase can be totally moved out from the protocol part. But in my prototype, I just left them in the VISComponent in order to do away with unnecessary classes. The private function of GetTransformers() and GetResultSetID() is for data preparation. The
GetTransformers() function takes in a parameter of metadata format and returns only transformers that will be able to convert this type of metadata format to some format which is supported by that VIS. The GetResultSetID() function takes the user’s query string and converts it to the correct resultSetID that the DL can recognize.

Another thing that needs to be paid attention is that in an HTTP request URL encoding is a MUST – i.e., special characters that appear in the value field need to be converted accordingly before writing to a HTTP connection stream, otherwise it is not a correct HTTP request. See Table 4.2 below for the right syntax rules of encoding.

<table>
<thead>
<tr>
<th>Character</th>
<th>URI Role</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Path Component Separator</td>
<td>%2F</td>
</tr>
<tr>
<td>?</td>
<td>Query Component Separator</td>
<td>%3F</td>
</tr>
<tr>
<td>#</td>
<td>Fragment Identifier</td>
<td>%23</td>
</tr>
<tr>
<td>=</td>
<td>Name/Value Separator</td>
<td>%3D</td>
</tr>
<tr>
<td>&amp;</td>
<td>Argument Separator in Query Component</td>
<td>%26</td>
</tr>
<tr>
<td>:</td>
<td>Host Port Separator</td>
<td>%3A</td>
</tr>
<tr>
<td>:</td>
<td>Authority Namespace Separator</td>
<td>%3B</td>
</tr>
<tr>
<td>''</td>
<td>Space Character</td>
<td>%20</td>
</tr>
<tr>
<td>%</td>
<td>Escape Indicator</td>
<td>%25</td>
</tr>
<tr>
<td>+</td>
<td>Escaped Space</td>
<td>%2B</td>
</tr>
</tbody>
</table>

Table 4.2. URL encoding table

4.2.5 Other Changes on VIS Side

The protocol part of the implementation is static and simple. However, all the other handling, such as maintaining the state information about DLs, collecting information for requests, and doing operations based on responses, will be much harder. More interoperability issues will be encountered and discussed in details in the next prototype.
implementation. In this ENVISION-ODL prototype I am implementing the minimum requirements and I believe this can be a good template for other VIS systems:

- Rewrite the parser. The original parser is parsing on the result file with a strict format as shown in Section 4.2.1. However, to enable flexible data type support, I have decided during the design phase to use XML as the data transmission format. So I rewrote the parser to deal with the new “enva” visdata format XML result. Also, whenever a new visdata format is to be supported in a VIS system, this step needs to be redone.

- The user should be able to choose some DL’s URL when he or she wants to visualize data into the ENVISION interface. A simple workaround for this problem is to take some URL as a command line parameter when the ENVISION system is starting up. If no URL is passed in, the default DL can be invoked.

- Maintain the session information. We want to keep the contacted DL information in the history as long as possible, until the DL’s supported metadata formats are changed for some reason. This can guarantee the necessary interaction between the VIS and DL systems will be as little as possible. We achieve this by the following solution:
  - For each new DL URL a user inputs, a directory corresponding to this DL’s name is created. Under this directory, we store all the key information that we need to know from the DL, such as Identify response, ListMetadataFormats response, etc. In the Identify response there is information of the URL of the DL, the last modified information, the user authentication information, the encoding information, and so on. The ListMetadataFormats response has all the metadata formats the DL supports as were known at last access.
  - The user can input only DL’s name if he or she is sure of having connected to this DL before. The VIS will check the DL name’s validity (if the same named folder exists).

  (a) If it’s in the history, this approach saves the user from inputting a long URL, also it saves the need of sending metadataformats and transformers between the VIS and DL systems again. The Identify request
will still be sent and the response will be compared with the saved copy, to make sure everything is normal since the last access.

(b) If it’s not, nothing is saved. Necessary interactions are made, and information is saved for next time’s reference.

More issues about interoperability will be discussed regarding connecting ENVISION to the second DL system – MARIAN.

4.2.6 Protocol Commands Implementation on DL Side

A Servlet on the ODL side is set up to handle the VIDI protocol requests and responses, as well as the transformation. This least impacts on ODL’s other major functions. As I noted before, it is better to implement the Post method in an HTTP request instead of a Get method. But for the purpose of simplicity and because of the belief that it should suffice for a demo of protocol, in my design I am still following the Get method, although it has a length limitation usually of 1024 Bytes.

This Servlet is relatively lightweight. It only implements a OnGet() method, in which it accepts a request and dispatches a response. The responses for Identify and ListMetadataFormats are static and predetermined, except the dynamic value from some fields such as responseTime, etc. For the ListTransformers command we return only an empty response (except required fields for every response), but will do some history keeping work as described in the next section. For the RequestResultSet command, the Servlet retrieves the resultSetID value, which is already in the OAI request format. So the Servlet sends out the OAI request and retrieves the OAI response, then it uses the transformer the VIS provides to do the transformation. Later the Servlet constructs all these into a VIDI protocol-compliant response and sends it back to the VIS.

To understand the RequestResultSet and resultSetID more clearly, we see the whole trace of them here in the following:

From the ENVISION’s old Query format (refer to section 4.2.1), ENVISION wraps information into a VIDI request:
Figure 4.8. Sample VIDI request string for command RequestResultSet

We can focus on the *resultSetId* part, which starts from the middle of the third line and is the encoded URL format of:

```
```

This string is directly used by the ODL Servlet to send out an OAI request to the ODL Server and retrieve result documents.

### 4.2.7 Other Changes on DL Side

Similarly the ODL side saves VIS side information to decrease redundant interaction. The same approach is applied as the one in the VIS side: for each new VIS a new directory is set as its home directory, under which the transformers the VIS sends is stored.

ODL was not giving “relevance” information in its responses. Because of the apparent importance of this field in the ENVISION visualization, the response format includes the “relevance” field in the `<about>` elements. (Refer to Section 4.2.1 for more explanation.)
4.3 ENVISION and MARIAN

The second prototype connects ENVISION with MARIAN.

As was explained in Chapter 2, MARIAN is a mature Digital Library. It is complex and provides many features for information seekers and many challenges for implementation. We planned to connect ENVISION with DIRLINE collection in the beginning, and all the following analysis is based on this. However, the DIRLINE collection is about medical organization record, which is very different from what the ENVISION previously supports. The ENVISION is not flexible enough currently. We believe it can be reengineered to be reconfigurable based on the input schemas later. Currently we only connect it to the NDLTD collection, which is in Dublin Core format, and we are not planning to say much about it (see last Section for details). The following analysis will be very helpful for the later implementers to successfully connect ENVISION with DIRLINE data.

4.3.1 Analyzing Formats on MARIAN and ENVISION

The first step is to understand the format of the DIRLINE data. It is the major collection running on MARIAN now. Also the size of this collection is still expanding day by day – the resources (the records in the collection) can be dynamically suggested to be included into the collection. The schema for DIRLINE metadata format is very complex, as we can see below:

```
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirline"
    xmlns:dirline="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirline"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

<annotation>
    <documentation xml:lang="en"
        XML Schema for expressing National Library of Medicine DIRLINE data in XML - 2001-10-23
        Created by Jun Wang
    </documentation>

```
Based on a sample collection of DirLine data from Tamas Doszkocs via Robert France

</documentation>
</annotation>

<xsd:element name="DOC">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="DOCNO" type="xsd:string"/>  <!-- AIDS/00017 -->
      <!-- Name -->
      <xsd:element name="na" type="xsd:string" maxOccurs="unbounded"/>  <!-- Name -->
      <xsd:element name="ac" type="xsd:string" minOccurs="0" minOccurs="0"/>  <!-- abbreviated Name -->

      <!-- Mailing address including ad, city, sts, zp and cy -->
      <xsd:element name="ad" type="xsd:string" minOccurs="0" minOccurs="0" maxOccurs="unbounded"/>  <!-- address line -->
      <xsd:element name="city" type="xsd:string" minOccurs="0"/>  <!-- city -->
      <xsd:element name="sts" type="xsd:string" minOccurs="0" maxOccurs="0"/>  <!-- state -->
      <xsd:element name="zp" type="xsd:string" minOccurs="0" maxOccurs="0"/>  <!-- zipcode -->
      <xsd:element name="cy" type="xsd:string" minOccurs="0" maxOccurs="0"/>  <!-- country US, CA -->

      <!-- (410) 685-1180 (Voice) -->  <!-- didn't handle case like <tel><x800>(800) AIDS-NYC</x800></tel> -->
      <xsd:element name="tel" type="xsd:string" minOccurs="0" maxOccurs="0" maxOccurs="unbounded"/>  <!-- phone number -->

      <!-- Internet access -->
      <xsd:element name="eml" type="xsd:string" minOccurs="0" maxOccurs="0" maxOccurs="unbounded"/>  <!-- website, email -->

      <!-- Person contacts -->
      <xsd:element name="ic" type="xsd:string" minOccurs="0" maxOccurs="0" maxOccurs="unbounded"/>  <!-- including name and position -->

      <!-- Description -->
      <xsd:element name="ab" type="xsd:string" minOccurs="0"/>  <!-- description -->
      <xsd:element name="pb" type="xsd:string" minOccurs="0" maxOccurs="0"/>  <!-- Publication Type -->
      <xsd:element name="gn" type="xsd:string" minOccurs="0"/>  <!-- garment -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
When the first prototype was implemented, the major issue we faced was to prove the protocol could work. Now by connecting to MARIAN, we are experiencing more issues, e.g., user authentication, and better proof of interoperability.

I analyzed each field in the schema to decide which of them should be supported in the ENVISION visualization. <DOCNO>, <na> (NAME), and <ac> (ABBREVIATED NAME) are critical fields to present one record, so I am going to support them. <ad> (ADDRESS), <city>, <sts> (STATE), <zp> (ZIPCODE), and <cy> (COUNTRY) all are
the features connecting with a location. If a VIS system is able to present geographic information about locations, then these information fields will be really helpful. But in ENVISION currently there is no efficient way to visualize this type of information, so they are not selected. So as applies to the <tel> (TELEPHONE) field: it would have been very interesting if the VIS system could support the visualization of a directory of telephone numbers. The <eml> (EMAIL AND INTERNET ACCESS) is similar to the original supported <url> field in ENVISION. <ic> (CONTACT PERSONNEL) is a very interesting field: it provides not only the name information but also the position information the person is holding. When doing the transformation I need to filter the position information and keep only the name information, which is similar to the originally supported <author> in ENVISION. <ab> (ABSTRACT) is too long to visualize. <pb> (PUBLICATION TYPE), <gn>, <lun>, <ho>, <sa>, <em>, and <dr> are too trivial field or fields with too random values which does not make sense in visualization. <kw> (KEYWORD) and <nt> (ORGANIZATION TYPE) should be important enough and specific enough for our visualization. <lr> is the set up time for the organization, which contains month, day, and year information. In the transformation only the year will be taken into account. <mesh> is a hierarchical field, which will be very interesting to be included in the implementation. <rg> (National Network of Libraries of Medicine Region) seems to have divided the locations of organizations into 8 regions, but I am not sure what each region represents. It would be nice to see if it’s represented the same way as relevance. Again the <relevance> is not given in metadata formats. I will get this field from the MARIAN search engine and add it in while applying a transformation.

After the analysis, our visdata format is listed below:

```xml
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirvis"
    xmlns:dirline="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirvis"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

  <annotation>
    <documentation>
      the visdata format corresponding to dirline data
    </documentation>
  </annotation>
</schema>
```
<xsd:element name="dirvis">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="DOCNO" type="xsd:string"/> <!-- one and only one -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
Following is the table that explains the meanings of each field in DIRLINE, and the mapping of fields into ENVISION visdata.

<table>
<thead>
<tr>
<th>Field in DIRLINE</th>
<th>Meaning</th>
<th>Field in DIRVIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC</td>
<td>Organization record</td>
<td>dirvis</td>
</tr>
<tr>
<td>DOCNO</td>
<td>Record identifier</td>
<td>DOCNO</td>
</tr>
<tr>
<td>na</td>
<td>Name</td>
<td>name</td>
</tr>
<tr>
<td>ac</td>
<td>Acronym name</td>
<td>acro</td>
</tr>
<tr>
<td>ad</td>
<td>Mailing address line</td>
<td></td>
</tr>
<tr>
<td>city</td>
<td>City</td>
<td></td>
</tr>
<tr>
<td>sts</td>
<td>State</td>
<td></td>
</tr>
<tr>
<td>zp</td>
<td>Zip code</td>
<td></td>
</tr>
<tr>
<td>cy</td>
<td>Country</td>
<td></td>
</tr>
<tr>
<td>tel</td>
<td>Contact telephone number</td>
<td></td>
</tr>
<tr>
<td>eml</td>
<td>Email address or website</td>
<td>eml</td>
</tr>
<tr>
<td>ic</td>
<td>Contact personnel information</td>
<td>cname</td>
</tr>
<tr>
<td>ab</td>
<td>Abstract or description</td>
<td></td>
</tr>
<tr>
<td>pb</td>
<td>Publication type</td>
<td></td>
</tr>
<tr>
<td>gn</td>
<td>Some comments</td>
<td></td>
</tr>
<tr>
<td>ho</td>
<td>Some comments</td>
<td></td>
</tr>
<tr>
<td>lun</td>
<td>Comments about possible fee</td>
<td></td>
</tr>
<tr>
<td>site</td>
<td>Locations of sub organizations</td>
<td></td>
</tr>
<tr>
<td>fx</td>
<td>Super organizations</td>
<td></td>
</tr>
<tr>
<td>kw</td>
<td>Keyword</td>
<td>kword</td>
</tr>
<tr>
<td>nt</td>
<td>Organization Type</td>
<td>otype</td>
</tr>
<tr>
<td>sa</td>
<td>Agency</td>
<td></td>
</tr>
<tr>
<td>rg</td>
<td>Region Number 01-08</td>
<td>region</td>
</tr>
<tr>
<td>em</td>
<td>Some 4-digit number for internal use</td>
<td></td>
</tr>
<tr>
<td>dr</td>
<td>Some 4-digit number for internal use</td>
<td></td>
</tr>
<tr>
<td>lr</td>
<td>Organization foundation date</td>
<td>fyear</td>
</tr>
<tr>
<td>mh</td>
<td>Mesh headings</td>
<td>mesh</td>
</tr>
<tr>
<td>relevance</td>
<td>Relevance for searching</td>
<td>relev</td>
</tr>
</tbody>
</table>

Table 4.3. Fields in DIRLINE metadata format, DIRVIS visdata format, meanings and mapping

The fields are similar to the ones ENVISION used before, but there are at least two major differences:
• The field has different semantic meaning, e.g., the person’s name does not represent author name any more, but contact information; and there is no “title” field, but “name” and “acro” instead.

• Some of the ENVISION’s features will not be fully utilized. Earlier if the field’s values are specified to be chosen from 3 types of value, then this information can be shown not only with icon, but other things like images or shapes. But in the DIRLINE collection, there is no single field that has only a few specified values, so this feature will not be used in ENVISION. The only thing that is similar is “region” field, which has range of integers 1 to 8.

4.3.2 Transforming Between Formats

After the format analysis is completed, the XSLT stylesheet is written and the transform can be deployed. Note that the input for the transformation will have to be a complete XML file with correct root element, and the list of records of <DOC> documents obtained from MARIAN. The transformation stylesheet is shown as below:

```xml
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
    xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirvis"
    xmlns:dirline="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirline"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<xsl:output method="xml" indent="yes" omit-xml-declaration="no"/>
<xsl:template match="/">
    <numRecords>
        <xsl:number value="count(/dirline:*/dirline:record)" format="1" /> 
    </numRecords>
    <xsl:apply-templates select="/dirline:*/dirline:record" />
</xsl:template>
<xsl:template match="dirline:record">
    <record>
        <dirvis
            xmlns="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirvis"
            xsi:schemaLocation="http://tuppence.dlib.vt.edu/manual/VIDI/1.0/dirvis"
```
For example, this stylesheet is converting the record in metadata format dirline

```xml
<?xml version="1.0" ?>
<results>
  <record>
    <DOC> the dirline doc </DOC>
    <relevance> the relevance </relevance>
  </record>
  <record>
    <DOC> another doc </DOC>
    <relevance> somewhat less relevant </relevance>
  </record>
  ...
</results>
```

Figure 4.9. XML document in DIRLINE metadata format

into visdata format dirvis as below:

```xml
<numRecords> num of records </numRecords>
<record>
  <dirvis> dirvis format </dirvis>
</record>
<record>
  <dirvis> another dirvis </dirvis>
</record>
  ...
</record>
```

Figure 4.10. XML document portion in DIRVIS visdata format
4.3.4 Protocol Commands Implementation on VIS Side

I have some good ideas about how to construct a resultSetId that is used in this prototype. The real implementation can be flexible and suited to a specific DL and VIS pair. In the case of ENVISION, a dynamic query system, we reuse the input form of MARIAN (only the query part, since other parts like user registration are not relevant).

A VIS servlet is written to implement the VIDI protocol commands. It also has the functionality of accepting the user’s input query from a web page that we provide. The HTML page from MARIAN is modified for the purpose of providing the user input query only. All the input fields are kept the same as the ones being used in the MARIAN request web page; besides, the form’s action target is changed to the servlet on the VIS side, instead of the nlmServlet residing in the MARIAN. Two fields are added: user name and user password, to do the necessary authentication operation. All other user requests except sending queries, such as user registering, and check history, etc., are not handled. If the users want to do any of these, they should directly go to the MARIAN web page and request the nlmServlet to do that.

The information filled in the form is collected and put in the field of resultSetId when sending the request to the DL servlet. Since now the VIS side has its own servlet, the relationship of VIS and DL is peer-to-peer in this prototype implementation. In this mode both DL and VIS sides can send and receive VIDI requests. An ENVISION instance will be initiated when a user chooses a new query. The response will be shown through the ENVISION instance connected with the VIS servlet. See Figure 4.11 to understand how the parts cooperate.
4.3.5 Other Changes on VIS Side

How to make ENVISION reconfigure to be suitable for different structures, is the hardest question we are facing in this section.

We now have the dirvis schema in hand – the schema for the visdata format, which represents the fields that we want and could visualize. This dirvis is very different in semantics from the old MARC library record and the electronic copy of dissertation and theses. It is describing the record of organizations’ information: It has the name for the organization, the abbreviated format of the name, the contact person, the year it was founded, the MESH hierarchy terms, the relevance value, and some other fields. Now let’s think how to put this new format together with the original enva visdata format.

Data structures must be able to be automatically generated and accessed. A new module to deal with the schema file is added. After the structure is analyzed (either enva or dirvis currently), the data structures are generated. When the results have been received, the parser reads them through and assigns them to those automatically generated data structures. The interface access needs to refer to those data structures as well.

4.3.6 Protocol Commands Implementation on DL Side
This servlet is different from the original ODL one in two aspects. First, it is using a socket to connect with MARIAN Webgate (the DL), instead of the OAI request in HTTP format. Second, when it finds that some information from the VIS side needed for a protocol command is missing, it is able to ask the VIS servlet for them – the peer-to-peer structure enables more functionality. So the communication between the VIS and DL now is smoother, without the worries of data flow between two systems stopping suddenly in the middle.

The major function of this DL, however, is similar to the last servlet in the ODL prototype: It is able to implement all the protocol commands. It is able to handle the resultSetID and pass it to the DL and do real retrieval. It applies the transformation designated by the VIS side through the command ListTransformers on the results it gets from the DL, and wraps the result in the correct Response format and presents it to the VIS side.

### 4.3.7 Other Changes on DL Side

To understand how we can start modification to let MARIAN support our protocol, and return the right results when we request them, we have first to do a detailed analysis about how MARIAN behaves. MARIAN itself has an nlmServlet to handle all the requests from users through HTTP pages. Users have all kinds of requests to MARIAN. See Figure 4.12 regarding a simulation of a user input query from the webpage interface.
Figure 4.12. Marian nlmServlet handles user’s request and talks to Webgate

Because it directly wraps everything into MARIAN_CGI_Request and MARIAN_CGI_Response, in which they already handle the return information including HTML pages and error codes, it doesn’t suit for our use that much. A separate Servlet is written to implement all the protocol commands.

Both the request and the response part are different. In the response part, the results are directly in the format of a list of XML records, instead of a MARIAN_CGI_Response. After that the records are transformed in the DL Servlet using the newly written transformer discussed in Section 4.3.2, wrapped into a correct protocol response, and sent back to the VIS system (see Section 4.3.6 for details).

In the request part, the MARIAN_CGI_Request and Environment classes also are skipped. The content, obtained from the simple query input interface and passed over through the protocol field resultSetID, is wrapped into a StringPool. Now the DL servlet can exactly send the same query to the WebGate of MARIAN in StringPool format and get the result back in StringPool format as well.
However, originally the process regarding how MARIAN deals with queries is, putting all the results returned in a class called “Results”, and only returning the short description format which only contains the names to the web pages. On the web pages users can choose to see more detail in that record or ask for more. This is not directly suitable to our needs of full XML format document. To handle this problem, a new type of request with the name “vis_request” is added as well as the new functions to handle against this type. In this request-handling process, we are using not the short form, but the form that is in raw XML and is the long type. Also, we will only return the documents that are inside the batch_size the users specified.

Another problem is, MARIAN has to generate the result list with relevance and so the result list should be in the format of XML to be processed by XSLT. Also the result document under processing should follow the dirvis format as described in section 4.3.2 to fit into the response for RequestResultSet.

The relevance value was not collected in the document in MARIAN. For visualization purposes, I traced the ranking scheme in MARIAN, found out the tf-idf weight mechanism MARIAN is using, and passed the weight value to my servlet through its response. Also when handling against the “vis_request” request from the DL servlet, the Response directly returns the result documents in the dirvis format we want.
CHAPTER V
PROOF OF CONCEPT

5.1 Approach

We can prove that this VIDI approach will work in general using the following two strategies. This assumes that the class of DLs is relatively homogeneous. The first approach is by implementation, discussed in 5.2. The second approach is to cover how each class of VIS system would be supported by the protocol. This is explained in section 5.3.

5.2 Implementation Explanation

It is not feasible for me to do all possible implementations on these two systems to prove our point completely. From a theoretical perspective, however, we can argue from cases implemented that they are both realizable.

5.2.1 SOM

Kohonen self-organization map (SOM) [35] [36] is an adaptive 2-D Kohonen-based Self-organizing Map. The SOM allows user customization of the level of categorization the tool provides. The Cancer Map for CANCERLIT project in University of Arizona is such a graphical display of important cancer concepts and a document server supports guided browsing of concepts and documents. The map consists of two parts. On the left hand side, a user finds the classical hierarchical view containing the location of the documents in the categories. On the right hand side is a multi-layered map with concept labels to indicate the different categories. Clicking on the map region takes the user down a layer, or if no lower levels remain, shows the documents associated with that particular category. In Figure 5.1 is shown the interface for the Cancer Map of SOM.
The experiment with Internet entertainment homepages leads to the Dynamic SOM (DSOM). It creates a map of a document collection on the fly. It can accept a query from the user and do a search, then the DSOM categorizes the top 300 documents, and in a matter of seconds, a user can review the key topics in a retrieved document set and browse the content accordingly. In this case, SOM is a tightly coupled system. Its structure can be explained as shown in Figure 5.2.
A plain crawler crawls HTML pages from the Web. It produces a document list, in which each document contains name, file content, URL, and some other general information. A phraser reads this document list and produces a list of phrases, which becomes the input of the SOM backend module. The SOM backend module runs some algorithm to produce a cluster representation file, which contains all the necessary classification information. The SOM frontend can understand this format of cluster representation file; it does some analysis and produces a Kohenen’s map and visualizes the result to the user.

This whole process is now working well. However, we can think to decouple it using the VIDI protocol to achieve interoperability. For example, currently, the input data source is coming from a crawler, which crawls a bunch of HTML pages. We might think of visualizing some specific records from some DL system, instead of a random Web. Also, currently the “phraser + backend” transforming approach is working as a black box. It is
running some algorithm to transform the data obtained from the Web into some specific classification file that the SOM front end can recognize and can visualize. If we have some other classification algorithm when we are applying the visualization to another data source, we now have the option of doing another type of transformation, which will work for that type of DL only. In short the decoupling by VIDI protocol makes the whole structure appear even more clearly, plus ensures the reusability of the SOM visualization frontend.

Now let’s see how these can be achieved through VIDI. To accomplish the decoupling, we must carry out the following steps:

- The crawler will be the DL end. The crawler now has to implement the commands the VIDI protocol defines.
- The “phaser + backend” will be the transformer. To decouple it from the SOM visualization frontend, we can put it in a different place where it is accessed by HTTP request and response. The transformer defined in the VIDI protocol (refer to Chapter 3) has a field of baseURL. This baseURL field defines the location of the transformation tool. It can be an URL for an .xsl stylesheet, in which case the DL will retrieve the stylesheet and do the transformation locally. Or it can be a URL for an HTTP request server, e.g., a servlet. The DL now can send the request to this URL with all the data as POST input, and the transformer can perform the necessary transformation operation, returning the transformed results as the response. After the DL gets the information it can send it in a normal VIDI way back to the VIS system SOM.
- The SOM frontend now is the VIS end. To implement the VIDI protocol, what it needs to do is implement all the commands the VIDI protocol defines, including making the acceptable cluster representation file in XML format. This is very important because, if the SOM frontend is going to talk to many other DLs with different transformers later in the future, the machine-readable complex file format it now defines is not enough. If other DLs want to use SOM, the SOM has to provide a common interface that will facilitate others. XML is one of the very
good methods to do this. The SOM frontend will achieve the reusability and interoperability we analyzed above after this essential step.

The later SOM decoupled using VIDI will be as shown in Figure 5.3:

![Diagram of SOM applying VIDI protocol]

Figure 5.3. SOM applying VIDI protocol

5.2.2 UVA

UVA [33] is the visualization part of an initiative termed U-DL-A (University Digital Libraries for All) in Mexico. UVA (U-DL-A Visualization Aid) is aiming at the DLs that organize materials according to some widely accepted schemes such as the classification system of The Library of Congress of the United States. Its visualization is a 3D tree visualizing procedure extended from Cone Trees [34]. In this process it recursively traces the branches on the imaginary surface of a cone, calculating distances based on its height, radius, and generatrix. The click on the parent node can initiate the representation of its children. Its interface is shown in Figure 5.4:
In a paper [33] and a few other web pages I can find about it, there is not much more information except the visualization implementation detail. From the discussion of the visualization process, however, we can know it is still a dynamic-query type of visualization system. Currently it is tightly coupled to the DL system it is now running. However, we can decouple it using VIDI accordingly. The DL part and VIS part are there. How can we decouple them to make UVA really realize the goal of supporting any hierarchical DLs with specified classification information?

This still can be implemented using the VIDI protocol. Each time when the user clicks on a cone node, it can be deemed as seeking a more advanced (next level) of information from the DL using the parent information. The resultSetID field that is required for a requestResultSet needs to be considered. One solution to this is to define a description
container in the DL implementation of the Identify protocol command to describe the format of classification supported as well as the format of the resultSetID. In this case, an ID to indicate the parent node, and information on the hierarchical classification system this visualization is following, should be sufficient to retrieve all the next-layer information. After the information is retrieved through VIDI, the UVA can apply its algorithm and show them on the interface to the users.

In this implementation, the transformer part should not be that important. An XSL stylesheet should be sufficient if filtering the unimportant information from a record in DL. The Identify will need a little more effort before the DL can be successfully understood by the VIS system.

5.3 Case Studies Over Classification of VIS

In this section, I am trying to prove that the VIDI protocol is generally applicable by analyzing its work process and performance on all four types of VIS systems based on the classification we used in Chapter 2. After completing all this analysis, we should be able to conclude that VIDI is generally applicable and is able to help in enhancing the interoperability between VIS and DLs.

5.3.1 Infosphere Visualization

SOM can be put into this category. The input format (DL metadata format) can be random, while the output format (VIS visdata format) can be very specific. To think of the Web visualization, the information retrieved from the Web contains only general information such as HTML URL address, HTML title, HTML content, and so on. This metadata information is analyzed (for example, to extract anchor text from the HTML content to get a map of hierarchies) into visdata format (hierarchies of structures), and then visualized. If we separate the transformer part from the VIS, we can reach our goal of interoperability afterwards.
5.3.2 Workspace Visualization

This type of visualization corresponds to a very specific type of request and response. As we analyzed before, the interface works like a drawer or a folder, to save some specific records from the users, so that they can be retrieved any time they like: the workspace or toolkit are always there for the users. This can be achieved in the DL-VIS background by retrieving only the specific set from the DL. The user defines some specific set on a DL (of course the DL has to support this type of user request), and can request this set later by the specific setID. It can visualize the workspace on any VIS system that the user wants to do the visualization on, using the VIDI protocol.

5.3.3 Visual Knowledge Tools

We have already seen and discussed many VIS systems in this category. Besides those dynamic-query systems, there are some general formats of VIS systems (like Table Lens). Due to the simple format supported, usually they are not the most suitable VIS systems to present data from DLs. They still can utilize our protocol, by adding wrappers, and setting transformers always to be null (no transformation necessary). Here we analyze one specific example – connect Spotfire with a DL – to understand how our model and the protocol would work.

The VIDI protocol is to describe the behavior between the VIS system and the DL system. To make the whole scenario work, the user is playing an important role which is not covered in the discussion in the scope of my protocol. Now there is a DL system, and a Spotfire application. Assuming the schemas and transformers have been written beforehand, how can a user visualize data from the DL by using the Spotfire?

Firstly, the user needs to pass the DLs URL to the Spotfire, by direct input through a command line parameter or by choosing from a server list of all the DLs that Spotfire supports. Spotfire now starts to communicate with the DL, and exchange the necessary system information such as identifier, metadata formats, and transformers. If inside the
identifier the DL provides user authentication information, then corresponding information, such as user name, password, and so on, needs to be input by the user as well.

Since Spotfire is a very general application, it does not have any specific visdata format. But to be suitable for the data set the user is choosing to visualize, the user can specify one format he or she feels interested in (this also can be done by choosing from some list, or searching in a public depository storing metadata formats, visdata formats, and transformers) and get a corresponding transformer. The user even can choose the visdata format to be the same as the metadata format, which means the transformer will be empty.

Now both the DL side and the VIS side have the necessary information and it is the turn to pass data. The user needs to provide the Spotfire a resultSetID, which is decided between the user and the DL. For example, the user can directly access the HTML interface of the DL, search for some result, and corresponding to this set the DL provides some unique ID. Or, like what we did in the ENVISION prototype, the VIS provides some query window, simulating the user’s query on the DL, then the VIS converts the query to a query the DL supports, so that the VIS can directly retrieve the transformed result set. The last case can be possible when the VIS already knows the query format detail about this DL. The simulating interface and the converting query procedure can be loaded as a separate module when the user is choosing the DL. In either implementation case, Spotfire now uses this resultSetID (either a real ID, or a query) with the DL. The DL uses the transformer to transform it into the correct visdata format, then pass the set back to Spotfire to present to the user. The pictures below describe the process:
Figure 5.5. User contacts DL to get information to provide to Spotfire

Figure 5.6. Spotfire provides more support for user to access DL
5.3.4 Visual Object Visualization

There is no clear DL system in this case, because the package of visual data usually is already installed on the VIS system side. But if we deem the package as the DL (data source), we can apply the VIDI protocol. For example, in the Visible Human Project we mentioned in Chapter 2, when the user clicks on the interface at a specific height, it can be looked upon as the dynamic-query system that directly asks for some specific visual data instead of some records in a DL system. However, as we analyzed before, a DL can have all types of data, including video or audio records. Here all images can be viewed as one record in a DL system. In this case, we can simply do some minor transformation on the metadata around the image (such as the height – it might be an index in the DL side, or the resolution of the image, and so on), after which we will get the needed format for the VIS to represent.
CHAPTER VI
CONCLUSIONS

This thesis has proposed a VIDI protocol that enables the interoperability between Visualization Systems and Digital Library Systems. It has described the feasibility of the VIDI protocol and some other capacities that the VIDI protocol can provide during the implementation of the two models of existing systems. The thesis discussed the efforts and modifications made to the existing systems in order to reach the goal of effective communication. It has suggested some design issues when a new VIS system is designed. This research represents important steps toward the realization of a truly flexible, scalable and interoperable world of systems.

6.1 Contributions

This thesis addresses the concerns arising from the interoperability issue between Digital Libraries and Visualization Systems. The work also proposed some solution to this problem. Following are the contributions made by this thesis:

Cooperation of Visualization Systems and Digital Library Systems: The interoperability between the two kinds of systems has proved to be achievable through the solution proposed in this work. The benefits are enormous on both sides. New visualization techniques and retrieval techniques are encouraged through the cooperation to provide better and more efficiently services for users. Through this kind of cooperation, the huge amount of data currently stored in Digital Libraries can be better revealed and utilized through diverse Visualization Systems. New visdata formats and metadata formats can be created or extracted and then announced. Thus more and more systems will come to be supported.

New testbed for designing new VIS systems and/or DL systems: Some commands defined in the VIDI protocol are very useful. They provide sufficient information for
investigation and more abstraction. When we are designing a new system, if we keep in mind that it should be compatible with the VIDI protocol, then the system will come out as much more able to interoperate. For example, when designing a VIS system, to support the VIDI protocol, we will understand that the VIS system would be better if it can support multiple visdata formats as well as present its own visualizing features. So this VIS will not be so limited in use and yet can still be innovative, efficient, and effective VIS.

6.2 Recommendations for Future Work

The SOM and UVA VIS system can be decoupled to further demonstrate the VIDI protocol as explained in Chapter 5.

Furthermore, in-depth analysis and additional implementations need to be completed to thoroughly evaluate the proposal and realize the potential of the work. The implementation process also can be considered as leading to standardization.

This approach can be regarded as an ideal starting point for the investigation of more generic guidelines on how to achieve interoperability between related systems.

The ENVISION interface can be reconstructed to be more configurable. The interface should be able to change based on the provided schema files, which has the corresponding fields’ information and the attributes of those fields. Location or some other relationships, such as ‘refer to’, actually are very common attributes of a record from a DL, so we can think of enhancing the map quality of the ENVISION interface, to show the relative distance between records. For example, an address with details like state, city, and even street name can be converted to latitude and longitude information, so that can be projected on the map. In this case, ENVISION will not show two attributes on the X and Y-axes, but will present location better.
REFERENCES


VITA

Jun Wang was born in September 1979 in Wuhan, Hubei, People’s Republic of China.

Jun Wang earned a Bachelor of Science degree in the Computer Science & Technologies Department at Peking University, Beijing in May 2000. She started her graduate study in the Computer Science Department at Virginia Tech, Virginia in August 2000. She completed her Master of Science degree requirements in May 2002.

From May to August 2001, Jun Wang gained work experience as a software design engineer intern in the Microsoft Corporation, Washington. She will continue her programming and research life there from June 2002.