ABSTRACT

SYSTEMS INTEGRATION WITH DLSI

by
Tarun Preet S. Bedi

Systems integration aims to provide a homogenous view to a system consisting of many heterogeneous systems. Systems integration can be achieved by many means out of which the middleware approach is the most popular. This is because middleware can be built around existing systems thereby reducing the amount of changes needed for the integration. Nowadays many middleware technologies are available to integrate heterogeneous systems. One can chose among them depending upon the requirements. But due to the many technologies available for Middleware, a muddle of middleware exists. To reduce this muddle and to integrate the systems already implemented using different middleware technologies, Web Services plays a key role. Web Services is an XML-based technology which uses known protocols to communicate between applications, thus making them platform and technology independent.

Digital Library Service Integration (DLSI) provides a systematic approach in integrating the digital library collections and services. Amazon's website www.amazon.com is integrated with DLSI such that users see the same web pages as they see when they are logged in normally, but these pages will be augmented with link anchors. Wrappers will parse the web pages to look for elements of interests like book title, author name, etc., and provide them with link anchors which will connect the user to the library at New Jersey Institute of Technology where he or she can search for books or request them through interlibrary loan. The thesis proposes a new architecture for the DLSI as a web service, which can provide this functionality.

SYSTEMS INTEGRATION WITH DLSI

by
Tarun Preet S. Bedi

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

May 2004

APPROVAL PAGE

SYSTEMS INTEGRATION WITH DLSI

Tarun Preet S Bedi

Dr. Michael Bieber, Thesis Advisor — Date
Associate Professor of Information Systems, NJIT

Dr. Il Im, Committee Member — Date
Assistant Professor of Information Systems, NJIT

Dr. Stéphane Gagnon, Committee Member — Date
Assistant Professor of School of Management, NJIT

BIOGRAPHICAL SKETCH

Author: Tarun Preet S. Bedi

Degree: Master of Science

Date: May 2004

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2004

- Bachelor of Science in Mechanical Engineering,
  Pune University, Pune, India, 2000

Major: Computer Science

I dedicate this thesis to my brother Monty, who is not in this world to share this moment with me and to my parents whose constant motivation made all this possible.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Michael Bieber, who not only served as my research supervisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. Il Im and Dr. Dr. Stéphane Gagnon, for actively participating in my committee.

I would also like to thank Anirban Bhaumik and Nkechi Nnadi for their support throughout this task.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

CHAPTER 1

SYSTEMS INTEGRATION

1.1 Introduction to Systems Integration

Systems Integration is a very broad term generally used to describe the activities that are meant to access systems which are in a heterogeneous environment to attain a common goal. In the following pages the author has concentrated upon various systems integration (SI) techniques which are available.

To put in technical terms Integration is the mechanism of combining data and available functions in a system into a cohesive set (Heiler et al., 1991). Systems integration can be achieved by building the systems as an integrated entity or as an after thought in systems, which have already been built. But, it is relatively easy to have the concept of integration built in right from the word go into a system, than to try and integrate the existing systems which were never built with an intention of being integrated.

1.2 Types of Systems Integration

Systems Integration can be done at database level, user level, at semantic level or even at the process level. The types of approaches in systems integration can be classified as:

1) **Database Integration:** Usually when one talks about integrating the system at the database level, usually one means providing a central database. The schema of the database should be available in detail to the applications that need to access it.
2) **User Integration:** Integration from the User's (user of the system) point of view can be classified as either application (or functional) based or task based (Nilsson

et al., 1990). Application oriented systems are those where the user sees many applications which run on different screens (connected to different remote processes) but on the same machine. The task based systems are those where the user sees one integrated screen and the information needed for processing the user's tasks is achieved by the programs or processes that run behind the scenes.

3) **Semantic Integration:** The semantic integration provides a way for applications to exchange metadata across applications. Primarily, according to the definition given by W3C, the worldwide consortium for web, Resource Description Framework (RDF) is the basis for semantic web. It follows the pattern that each entity has a predicate associated with it to correlate into an object, i.e., the entities in two or more heterogeneous applications are associated on the basis of semantics.

4) **Process Integration:** Process to process integration allows the heterogeneous systems to initiate function calls on other systems and also respond to such calls. This is the basis of the middleware technologies such as DCOM, CORBA, JAVA RMI, etc. This gives a flexible solution to the problem of systems integration. Even if the systems were not originally intended to be integrated can be integrated by these middleware technologies.

The next chapter will discuss more about middleware as an approach of systems integration.

1.3 Thesis Overview

The objective of this thesis is to give an overview about systems integration techniques. It focuses particularly on the middleware as a solution to systems integration. In middleware the emphasis is given on web services as an emerging technology in integrating heterogeneous systems. The thesis then goes on to describe the architecture of DLSI and gives an insight of the components involved in the making of DLSI architecture. The thesis gives out the details of integrating www.amazon.com and the library at New Jersey Institute of Technology with the DLSI. The details include the Java classes and code snippets. The last chapter of the thesis proposes a new architecture for the DLSI. The proposal is to implement the existing DLSI as a web service.

CHAPTER 2

MIDDLEWARE

2.1 Introduction to Middleware

This section's primary focus is on the role of middleware in System's Integration. Let us see first why one needs middleware services. The current trend in information systems is of the fact that the computing systems are growing day by day. The older systems need to operate in conjunction with newer systems. It is rather difficult to obsolete the older systems because of time and cost constraints. So, these legacy systems should be able to integrate with newer systems. Moreover, these legacy systems can have different hardware and operating systems. This adds to the heterogeneity in the systems. Steve Vinoski, in his paper on Middleware (Steve Vinoski, 2002) states that the above causes as well as mergers, reorganizations, leadership changes, e-business, etc., add to the heterogeneity in the existing systems. Now, because of this heterogeneity the information available to applications is distributed over different systems and the applications should be able to communicate with these systems.

2.2 Classification of Middleware

Now with the background of current trends of heterogeneity and distribution in information systems, a common solution needs to be identified, which comes in the form of Middleware services. Middleware services sit in a layer above the operating system and network software but below the industry specific applications. They provide the standard programming interfaces and protocols that mask the complexity of networks and

lower level protocols (Bernstein, 1996). Bernstein states that it is rather difficult to describe the middleware in a specific way. However, the following properties, when taken together, can be used to describe middleware:-

- They are generic across applications and industries
- They run on multiple platforms
- They are distributed
- And support standard protocols and interfaces.

Emmerich (Emmerich, 2000) in his paper has described middleware service as something that aims at providing application engineers with high level primitives that simplify distributed system construction. Also, Emmerich has provided an overview of current middleware technologies which are as follows:-

1) **Transactional Middleware:** The examples for these middleware services are IBM's CICS and BEA's Tuxedo. It uses two phase commit to implement distributed transactions. Two phase commit is the action of committing a transaction atomically, i.e., instantaneously and indivisibly. Client and server components can reside on different hosts and the requests can be transported over the network. Data heterogeneity is not very well supported because the middleware does not provide primitives to express complex data structures that could be used as service request parameters. Another drawback is the fact that marshalling and unmarshalling between data structures used by client and the parameters that service require need to be done manually. Marshalling is the process of transforming data structures (used by the client or server) into a format

that can be transmitted over the network and unmarshalling is the reverse of marshalling.

2) **Message-Oriented Middleware (MOM):** The examples are IBM's MQSeries and Sun's Java Message Queue. The communication between distributed systems components is facilitated by message exchange. The client sends a message across the network to the server by inserting the message in the server's message queue. The messages can be synchronous or asynchronous. The server responds to the client request with a reply message containing the result of execution. Like Transactional Middleware MOM requires the application engineers to write the code for marshalling. It is good for event notifications because when an event occurs, the client application hands off to the messaging middleware application the responsibility of notifying a server that some action needs to be taken and the client can then simply continue with its processing.

3) **Procedural Middleware:** Examples are JAVA RMI. Remote Procedure Calls (RPCs) support the definition of server components as RPC programs. An RPC program exports the procedures and associated parameter types. Clients that reside on other hosts can then invoke these procedures across network. RPCs support synchronous interactions between exactly one client and one server. Also, the RPC manages the procedure calls by automatically marshalling and unmarshalling the request and response. The major disadvantage is that the interface is not reflexive, i.e., procedures exported by one RPC program cannot return another RPC program (one RPC program cannot return another as a result of execution).

4) **Object and Component Middleware:** Examples are CORBA, COM and EJB. The client object requests the execution of an operation from a server object that may reside on another host. Client object has to have an object reference to the server object. Interface Definition is used to create the stubs which do the marshalling and unmarshalling. Object middleware support different activation policies which include whether the server objects are available all the time or they are activated on demand. CORBA supports multiple programming languages and uses a standardized data representation.

2.3 Challenges in Middleware

Current Middleware technologies have no doubt eased the way of integrating applications. But, now there is a problem where in we have ended up with too much middleware. With so many technologies available and so many systems being developed with these distinct technologies there is a muddle of middleware.

Vinoski, in his paper on middleware (Vinoski S, April 2002), states that the very abstractions and simplifications that allow middleware to address integration issues can cause problems between middleware systems, as eventually, due to the changes in the industry, there will be a need to integrate two or more systems that use different middleware. He also states that most significant challenge that today's middleware is facilitating Internet-scale application-to-application integration. With internet becoming the ubiquitous norm to support business interactions, the next goal is to extend the application integration from the intranet to the internet.

Emmerich (Emmerich, 2000) identifies the limitations of middleware. The current middleware systems do not scale beyond Local Area Networks and they do not respond well to the changing requirements. Most of the middleware technologies use naming for component identification. This requires that the client must resolve name binding in order to obtain a reference to the server component to the server component. This results to architectures which are inflexible where client components cannot dynamically adapt to better service providers as they become available. He states that marshalling and unmarshalling, which is done by the middleware, can sometimes be overhead. So, the author suggests the use of middleware and markup-languages as XML. The XML documents should be used to transfer data between components as uninterpreted byte strings. This requirement is fulfilled currently by Web Services that utilize SOAP/HTTP for message passing.

Now considering the challenges that the middleware systems are dealing with nowadays, let us look at the Web Services as a means of ironing out these challenges.

CHAPTER 3

WEB SERVICES

3.1 What is a Web Service?

According to W3C "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"

Web services use light weight interoperable protocols such as HTTP, SMTP, etc., to support the distribution of components. There are several definitions of web services available, let us look at a few of them.

Web services can be described as autonomous, modular applications, which can be run over the internet towards performing a specific business task. These services use specific norms so that they can be operated over the WWW (Gergic et al., 2002).

Web services can also be defined as "internet based applications fulfilling a specific task or a set of tasks that can be combined with other web services to maintain workflow or business transactions" (Aoyama et al., 2002).

One can describe Web services as:

- An Integration technology
- Based upon open standards (SOAP, WSDL, UDDI)
- Communication is in standardized XML messaging

- Uses standard protocols such as HTTP, SMTP for communication.

Web services unify two styles of integration: request–response and messaging. Typically, integration approaches have selected either tightly coupled request-response or the loosely coupled one-way messaging. The main advantage a web service is that it uses HTTP for transportation of SOAP messages, which means that web services can be invoked over firewalls. Web services can be described as the middleware for other middleware because rather than replacing the existing middleware, one can just integrate and expand their services via web services (Vinoski, 2002).

3.2 Components of Web Service

The following are the components or technologies involved in the concept of web service:

1) **SOAP:** SOAP stands for Simple Object Access Protocol. SOAP is the communication protocol for exchanging XML data over the web (Roy et al., 2001). SOAP is an XML based protocol for RPCs and messaging which works on existing transport protocols HTTP, SMTP, etc., (Cubera et al., 2002). Other middleware like COM, CORBA, Java RMI etc also provide similar functionality, but the best part about SOAP is that the messages are all written in XML and hence they are platform and language independent.

   A typical SOAP message consists of a SOAP envelope. This envelope is an XML element, which defines the rules, with two sub-elements; one is the header and other the body.

SOAP header consists of information pertaining to the credentials of the sender and other correlated information (Cubera et al., 2002). The details of the header are not bounded by many rules. However, the protocol specifies two attributes: `actor` attribute and `MustUnderstand` attribute. The `actor` attribute specifies the recipient of the SOAP Header. The `MustUnderstand` attribute tells whether the header element is mandatory or not. If it is set to true the recipient of the message has to process the SOAP header.

The body element is mandatory for all SOAP messages. The body element includes the requests and the responses. The information meant to be conveyed from one end to another is carried in the body. The data-types used to convey the requests or responses are defined in the XML schema specification given by W3C (www.w3.org/XML/Schema) (Cubera et al., 2002).

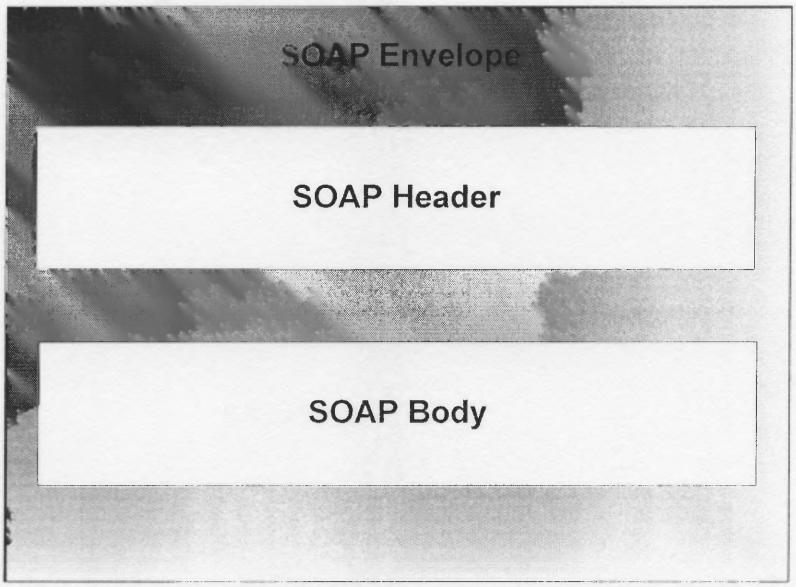

Figure 3.1 A SOAP envelope showing the encapsulation of the header and body, which forms a SOAP message.

2) **WSDL:** Web Services Description Language (WSDL), is the XML specification which defines the description of web services. Curbera (Curbera et all, 2002), define WSDL as an XML format used to describe Web services as collections of communication between the end points that exchange the SOAP messages, i.e., it describes a Web Service's interface. It is analogous to the interface definitions in RMI. But, it is platform and language independent because it is written in XML. The clients locate a web service from its definition in the WSDL and then they can invoke any published method of the web service.

The following is the specification of the elements in WSDL:

`definitions`

The definitions element is the root element of any WSDL document. Basically any WSDL document is simply a set of definitions for element at root and definitions inside. It defines the name of the web service and contains the elements which follow.

`types`

The types is a container for data type definitions used between the client and the server. It is needed only when the communication between client and server requires simple data types.

`message`

Message element describes operations like request-response, one-way, etc. It contains the message name and has one or more message part elements, which can refer to the message parameters.

portType

A portType is a collection of operations supported by one or more end points. It combines multiple message elements to complete one-way or on round trip operation.

binding

The binding describes which communication protocol to be used and how the service will interact over this protocol (Curbera et al., 2002).

service

A port element describes the address for invoking the specified service, i.e., the URL where the service can be invoked.

The following code snippet taken as an example from W3C's website describes an entire WSDL (http://www.w3.org/TR/2001/NOTE-wsdl-20010315).

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Figure 3.2 A sample WSDL message showing the key components marked in bold letters.

3) **UDDI:** UDDI stands for Universal Description, Definition and Integration. UDDI is like an online telephone directory of web services which presents an automated, unified and systematic way of discovering service providers (Curbera et al., 2002). The core component of UDDI is the UDDI Business Registry which is essentially an XML file used to describe a business entity and its web services. UDDI contains three types of information about Web Services:

   a) **White Pages:** they contain generic information about name and contact details of the business which owns a particular web service.

   b) **Yellow Pages:** these provide the information on the basis of business and service.

   c) **Green Pages:** These contain the technical information about the web services.

Curbera (Curbera et al., 2002), has described the UDDI registry as being organized around two entities that describe the businesses and the services that they provide. The business's information consisting of identifiers, contact information and description is provided by the `businessEntity` element. The services' information is provided by the `businessService` element, which can occur one or more times in a `businessEntity` element. Each data entity be it businesses or services is identified by a Universal Unique Key, which are globally unique. These Unique keys can be cross referenced between the businesses and services. These keys are a long hexadecimal strings assigned when an entity is registered. For example, `businessKey` attribute uniquely identifies a business and `serviceKey` attribute identifies a service. The technical information about accessing a web service is given in the `bindingTemplates`

entity. The binding template represents an access point to the service; this is so because the same service can be provided at different endpoints. Another important identity is the `tModelInstanceDetails` which provides the technical description, i.e., green pages. It contains the list of references to the technical specification with which the service complies. The technical specification of a service is represented in the UDDI by an entity known as `tModel`. The WSDL document for a service is registered as a `tModel`. The idea of the `tModel` is to be able to reference specification whose type and format cannot and should not be anticipated, i.e., reference some external specification for the service.

CHAPTER 4

DLSI ARCHITECTURE

4.1 Overview

The Digital Library Service Integration (DLSI) infrastructure will provide a systematic approach for integrating digital library collections and services. Digital libraries will be able to share relevant services within a seamless, integrated interface. Users will see a totally integrated environment. They will use their digital library system just as before. But in addition, they will see additional link anchors, and when they click on one, they will be presented with a list of relevant links. DLSI also will filter and rank order this set of generated links to a specific user's preferences and current task.

4.2 DLSI Architecture

The DLSI architecture consists of four levels:

1) User interface

2) The Integration Manager,

3) Collection and service wrappers, and

4) Independent collections and services.

The Figure 5.1 shows all the levels.

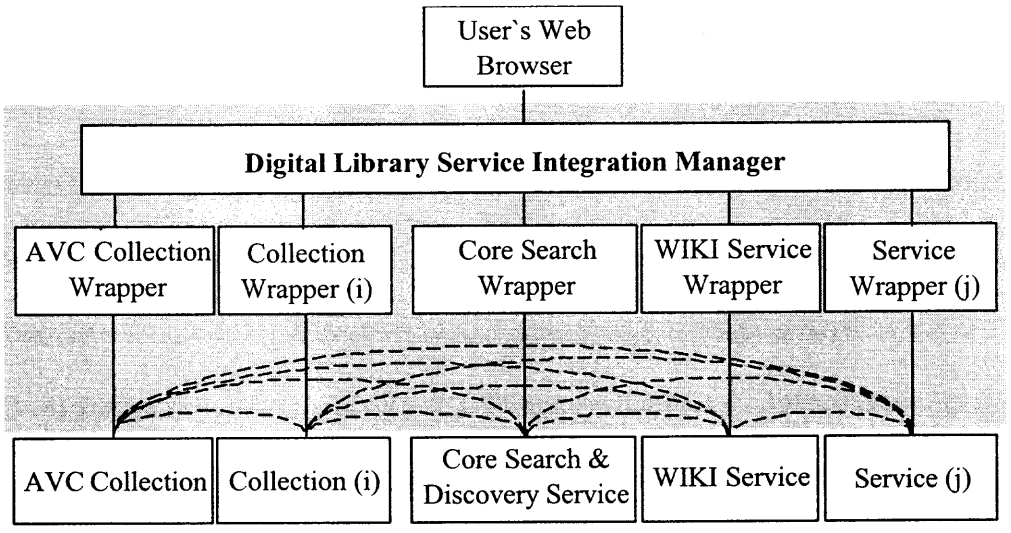

Figure 4.1 Design view of DLSI architecture showing the integration manager and the wrappers.

Briefly, the functionality of the DLSI is:

The wrapper's parse the display screens of the user's browser to identify the elements of interest such that the DLSI will make them into link anchors. Wrappers will parse the display based on an understanding of the structure of its content. Relationship rules specify which links DLSI will generate when the user clicks on a link anchor. Now, let us look at the DLSI architecture at a detailed level.

4.2.1 Metainformation Engine

The Integration Manager and the wrappers together constitute the Metainformation Engine (ME). The ME is a loosely coupled system, where the various components communicate with each other using messages that conform to a standard internal XML format.

The following figure shows the message flow in the ME. The components will be described later on

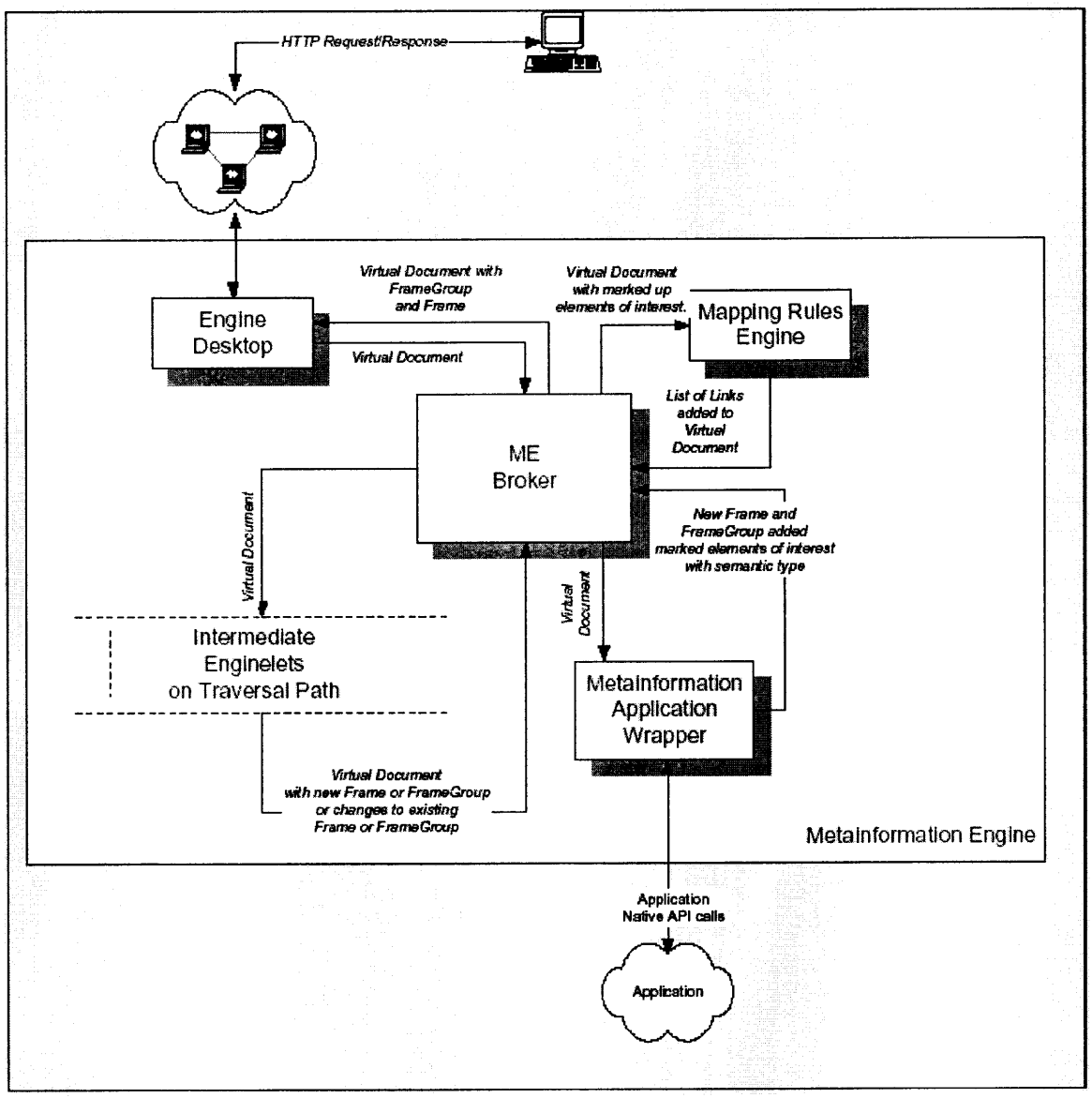

Figure 4.2 Message flow in the ME.

The following is the sequence of events which happen during the processing of a request:

- The user browser issues a HTTP request
- The Engine Desktop component generates a Virtual Document simulating the user's request.
- The ME Broker reads the source of the document, which is the Engine Desktop, and determines the path and destination of the document, which is an enginelet that will process the document. The traversal path is looked from the Traversal Path Manager (TPM) which gives out the route the Document follows, which might include several enginelets.
- The intermediate enginelets process the Virtual Document in some manner.
- After all the intermediate enginelets have processed the Virtual Document, the Document is sent to the destination enginelet or the Metainformation Application Wrapper (MAW). The MAW marks up the Elements of Interest in the application's output and adds the output document to a Frame and FrameGroup.
- This Virtual Document is then sent back to the MEB, which again looks up the traversal path. But now the source is the MAW and the destination is the Engine Desktop. Here also the intermediate enginelets process the document.
- When all the intermediate enginelets are done with their processing the document goes to the Mapping Rules Engine.
- Based on the Elements of Interest marked by the MAW, the Mapping Rules Engine looks up the mapping rules for each element of interest generates a list of links which are added as Relationship Objects in the Frames.
- The Virtual Document is returned to the MEB, which then returns it back to its source desktop.
- The desktop then displays the Document to the user.

4.2.2 Components of ME

1. **Virtual Document:** The Virtual Document is the communication protocol between the various engine components. It has an Id, Source, Destination, etc., fields which control the flow of the document across the components.

2. **ME Broker:** The MEB enables and manages the communication between the different components. MEB routes the document based on a registry of enginelets, which it uses to route the messages. It uses the Traversal Path Manager (TPM) to find the next enginelet.
3. **TPM:** TPM is a component of the MEB that determines the intermediate enginelets that must process a document before it reaches the destination enginelet.
4. **Mapping Rules Engine:** It identifies a class of elements of interest and provides hyperlinks for each instance of that class. It maintains a registry to generate the links. A mapping rule takes an object from one domain and links it to another one. For example, here the book name taken from Amazon is mapped with that of NJIT library.
5. **Engine Desktop:** The desktop enables the communication between the User Interface and the ME.
6. **Metainformation Application Wrappers (MAW):** The MAW's are the components that communicate with the underlying applications, identify the elements of interest and mark them up. The ME relies on the MAWs for parsing the document. MAWs are the components that an Application Developer needs to develop.

4.3 DLSI's Middleware Approach

The Metainformation Engine uses the Virtual Document as a means of communication between the various components. This is quite similar to being a Message Oriented Middleware. Just like messaging the ME sends an XML document with the address of the

source and destination in the message, giving rise to a system which can be integrated by writing new MAWs.

To develop and integrate with DLSI a developer writes a wrapper. The developer needs to write up the code to initiate the communication between collection service and the wrapper. The enginelet or wrapper has to be registered with the MEB with a URL qualifier so that the requests containing that URL can be routed to the enginelet. The author of the enginelet has to provide routine to parse the document to locate and mark the element(s) of interest. The mapping rules registry has to be modified to define relationship rules for generating links.

CHAPTER 5

INTEGRATION OF AMAZON WITH DLSI

5.1 Overview of Integration Work

The objective of this integration project is to demonstrate the ability of the DLSI to seamlessly integrate Amazon's huge databases of books with that of NJIT Library. The users will see the same pages as they see when they give searches on Amazon, but these web pages will be augmented with additional links which will connect them with NJIT's Library database. These additional links will provide the users with the supplemental links for the integrated system.

Amazon.com is world's largest online retailer of various products ranging from electronics to books. The integrated system will provide the user with a transparent navigation between the two systems.

5.2 Accessing Amazon with DLSI

The process flow of accessing Amazon's website with the DLSI architecture is:

- User enters the DLSI's website through the web browser.
- User clicks on Search Amazon's link to be taken to Amazon's search book's website routed through the DLSI engine.
- The requests for search are handled by an enginelet that calls the custom wrapper, which retrieves the search results from Amazon.
- The custom wrapper parses the document hence retrieved to locate the elements of interest based on predefined rules and marks them by adding link anchors.
- The document which has now been augmented with the link anchors is then sent to DLSI engine for display.

- The user can then navigate links generated to search for the book at NJIT library or submit a request to borrow the book via Interlibrary Loan.

5.3 Development of Enginelet

An enginelet is a component which is capable of processing the messages sent by the Metainformation Engine (ME). The application wrappers are the enginelets. The enginelet also parses a document and identifies the elements of interest to be mapped with the hyperlinks. The elements of interest are defined as the documents, objects or concepts for which other collections may provide information or other service can process (DLSI Developer Guide).

5.3.1 Changing the `enginelet.xml` File

The ME broker sends the request to the enginelet based on the entry in the enginelet.xml file, i.e., an enginelet must have an entry in the enginelet file so that it can be invoked. The information about the enginelet is contained in the EngineletData element of the XML document. The core attributes are:

- className: It's the entry point to an enginelet which is `com.dmi.apps.amazon.AMAZON`.
- engineletName: A unique name given to an enginelet which is `amazon`.

The following code snippet shows the added entry:

```
<EngineLetData ClassName="com.dmi.apps.amazon.AMAZON"
StartupOrder="12" NumInPool="1" EngineLetName="amazon"/>
```

5.3.2 URLs Used in Enginelet

Based on the URL of the request the ME determines which enginelet will process the request. The URLs used were designated with the "/" followed by the enginelet name followed by a traversal path. The document is passed on to the enginelet which is specified the traversal path by the main enginelet.

Here the author had used:

`/amazon`: to access the default search page of Amazon

`/amazon/dl`: will invoke the enginelet meant for parsing the results from Amazon

A code snippet of the same is:

```
public void init() throws EngineStateException
{
        Log.debug(this,"AMAZON module inited");
        setDefaultHandler(new AmazonQueryForm());
        add("dl", new AmazonLinkProxy());
```

The class which proxies and displays the Amazon's search page is `AmazonQueryForm`. Now when the user submits the search, the results are retrieved by the AmazonLinkProxy class and augmented with link anchors to display the results page to the user. The functionality of classes is explained in a later section.

5.3.3 Class Descriptions

All the classes implemented were packaged under the `amazon` package. The classes packaged are as follows:

AMAZON

This is the main class which serves as the entry point to the enginelet. This class extends the `EngineLetImpl` class. The MEB forwards all the URL requests starting

with /amazon to this class. This class specifies page handlers in the `init()` method The following code shows the classes which handle pages based on the second qualifier which follows the /amazon.

```
public void init() throws EngineStateException
{
      Log.debug(this,"AMAZON module inited");
      setDefaultHandler(new AmazonQueryForm());
      add("dl", new AmazonLinkProxy());
      add("njit1",new NJITLibLoan());
      add("in",new NJITLibLoanSub());
```

AmazonQueryForm

This class implements the `RequestHandler` interface. This class retrieves the amazon.com's search page for books and readies the search page to be submitted to the DLSI. This is done by changing the form tag in the html to be submitted to `/amazon/dl`. Following is a snippet wherein the class is requesting the document form Amazon's website:

```
public void handleRequest(VirtualDocument vDoc) throws
                                          EngineProcessorException {
try{
      CachedHttpClient clnt = CachedHttpClient.getInstance();
      HTTPConnection conn = new
            HTTPConnection(AmazonLinkList.HOST,AmazonLinkList.PORT);
            HtmlDocument doc = clnt.doConnect(conn,AmazonLinkList.HOST,
            AmazonLinkList.PORT,AmazonLinkList.MAIN_QUERY_FORM,
            "GET",null,null,0,false);
```

MyLinkFixer

`MyLinkFixer` class fixes the links retrieved from amazon.com's website so that they can be processed by the Metainformation engine. It has a static method called `FixLinks` which fixes the links (the relative links so that they point to their intended

destination when being displayed directly by Amazon), repairs the form tag (this is done because when the user submits a request for search, the communication should be handled by the enginelet) and locates the elements of interests (on the document where the system receives search results page based upon the search given by the user). It is invoked both by `AmazonQueryForm` and `AmazonLinkFixer` classes. The following code shows the location of the element of interest:

```
eofi = buf.indexOf("<td width=100% valign=top><font size=-1><b><a
                                                          href=");
while(eofi <= buf.length() && eofi > 0)
{
      eofi_text=0;
      end_eoi = 0;
      eofi_text = buf.indexOf(">",eofi+50);
      end_eoi = buf.indexOf("</a></b>",eofi_text);
      String interest = buf.substring(eofi_text+1,end_eoi);
      buf.insert(end_eoi,"<a href=\"javascript:listMappingRules(" +
                           count_eoi + ");\"> <img
                           src=\"/html/static/images/info.gif\"
                           border=\"0\" /></a>");
      ElementMI eMI = new ElementMI();
      String xpath = "/html/body";
      eMI.setLocator(xpath);
      Relationship rel = new Relationship();
      rel.addSemanticType("AMAZON.Book");
      rel.addDocumentParameter("Search_Arg",interest);
      eMI.addRelationship(rel);
      mi.addElementMI(eMI);
```

AmazonLinkProxy

When the user submits a request to search a book, the `AMAZON` class invokes this class to get the search results from Amazon's website. This class takes in the search parameters entered by the user and queries the Amazon's website for search request entered by the user. The results hence obtained are then parsed to locate the elements of interest and marks them with the link anchors.

NJITLibLoan

When the user selects the link "Borrow this book via interlibrary loan" this class retrieves the login page of NJIT library where the user puts the barcode and last name. It saves the name of the book that user had selected in a session variable which is then subsequently used to populate the request form.

```
HtmlDocument doc =
clnt.doConnect(AmazonLinkList.NJITLIBHOST,AmazonLinkList.PORT,
                        AmazonLinkList.NJITINTERURL,"GET",null,0);
String searchArgs =
vDoc.getRequestContext().getParamtable().getParamAsString("Search_Arg")
```

NJITLibraryLinkFixer

This class fixes up the links in the form tag to `/in` so that the login is routed to `NJITLibLoanSub` class.

NJITLibLoanSub

This class validates the user's ID and last name with NJIT's database. And displays the screen wherein the user can click on the book's button amongst other buttons.

NJITLibraryLinkFixerLast

This class changes the form tag on the page displayed by the `NJITLibLoan` class to `/amazon/last` so that the form is submitted to DLSI, which in turn handles the request to display the page where the user enters the details of the book and authors before he submits a request to NJIT.

NJITLibLoanSubLast

This class takes the user to the Interlibrary Loan Order Form. The book name is pre-populated in this form from the session variable saved in the NJITLibLoan class.

```
Paramtable pTable = vDoc.getRequestContext().getParamtable();
String searchArgs;
searchArgs =
(String)vDoc.getRequestContext().getSessionProperty("Search_Arg");
pTable.put("PublicationTitle",searchArgs);
HtmlDocument doc =
clnt.doConnect(AmazonLinkList.NJITLIBHOST,AmazonLinkList.PORT,
"/illweb/loan.cfm","POST",pTable,null,0);
```

5.3.4 Elements of Interest

The search results obtained from Amazon's website are parsed to obtain the elements of interests. The author has identified two parameters as the elements of interest: One is the Book Name and second, is the Author Name. The link anchors are placed besides these elements of interest in the display page. When the user clicks on these Link Anchors a pop-up widow displays the relationships between the elements of interest and links available for those elements. The easiest elements to identify are those that appear in a particular location in the page each time. For example, the elements of interest may be an item that always appears in the third column of a particular table on the page.

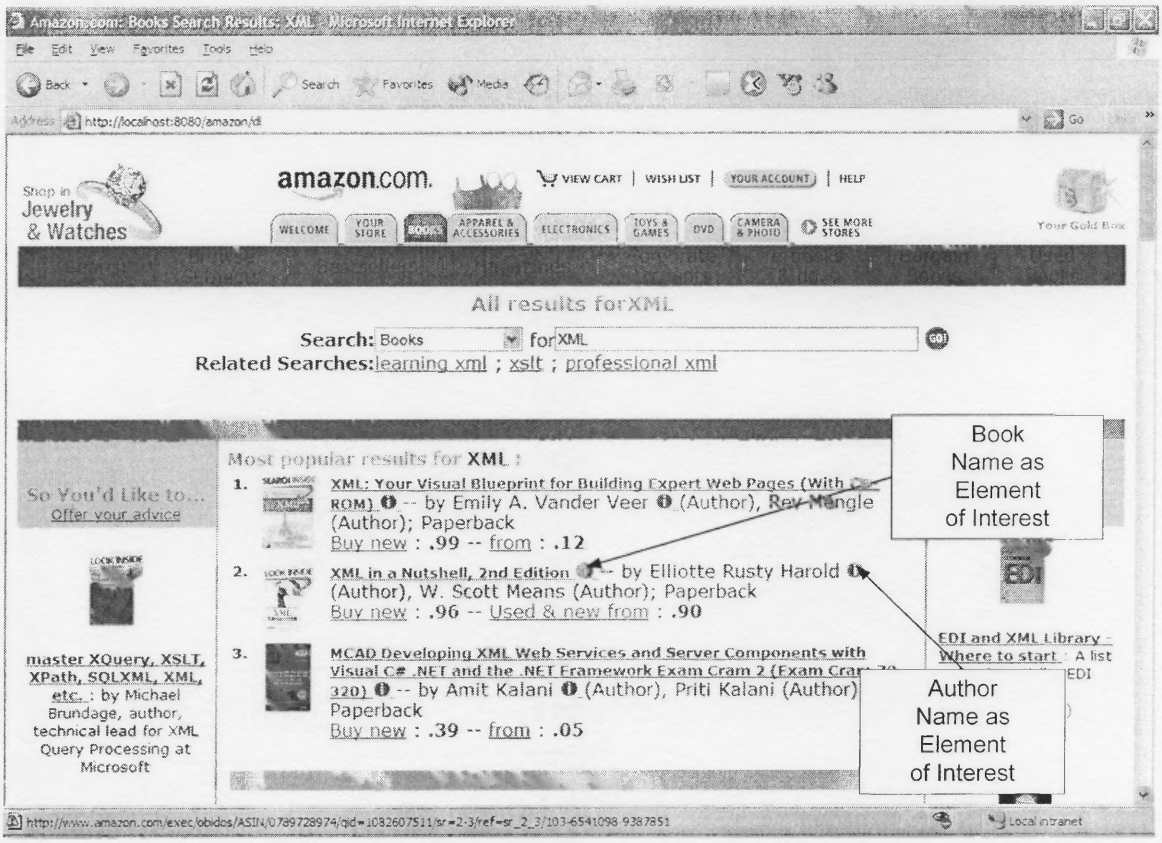

Figure 5.1 Amazon's search results page augmented with link anchors seen as an "i" tag and the elements of interest are marked.

5.3.5 Mapping Rules

Mapping rules provide a mechanism to add hyperlinks to the elements of interest. The enginelet's class `MyLinkFixer` creates a relationship object for each element. The Mapping Rules Engine (MRE) goes through the Metainformation object and generates the relationships with parameters such as the book name and the author name.

For each element of interest (book name and author name) the system will generate two relationships:

- Search for the book with this title or by this author (depending upon the element of interest) in NJIT library Database
- Borrow the book via Interlibrary Loan at NJIT

The following figure shows the pop-up which shows the relationships for each element of interest.

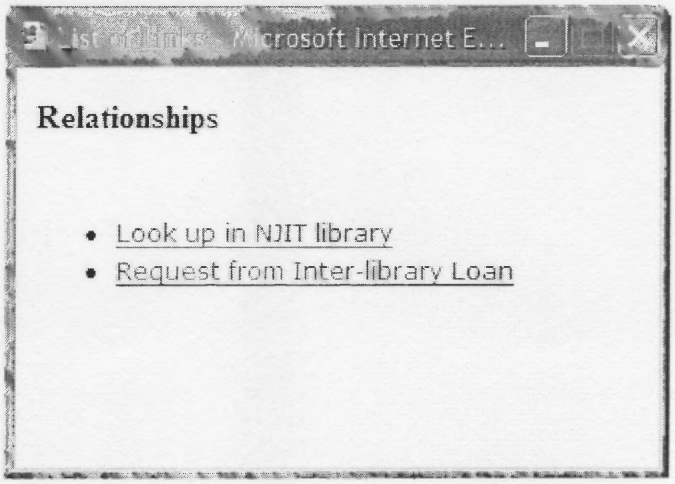

Figure 5.2 Relationships associated with each element of interest.

5.3.6 Problems Encountered

- The Tomcat server was starting but the login page of DLSI was not being displayed. The log file had an error message which looked liked like:

```
2004-02-08 14:26:50,424 [HttpProcessor[8080][0] ()] ERROR
com.dmi.core.desktop.xml.xsl.XSLTransform -
org.apache.xml.utils.WrappedRuntimeException: The output format
must have a '{http://xml.apache.org/xalan}content-handler'
property!
at
org.apache.xalan.serialize.SerializerFactory.getSerializer(Unknow
n Source)
```

  After some lookup on the web the resolution was found. The resolution was to copy a new version of the XALAN XML processor from JDK, as the one which came with the Tomcat was in conflict with that provided with the JDK.

- There were constant JDOM exceptions. The reason being the HTML document that was provided at amazon.com's website was not well formed, .i.e., they were not in compliance with HTML norms. Hence, when it was tried to convert the document into a JDOM object, the exception was generated. This was resolved by writing the code to make the document HTML compliant. The code edits the HTML page so that it gets processed by the JDOM engine.

5.4 Future Work

The code to locate the elements of interest will work fine as long as there are no structural changes in the HTML document. Each time Amazon's search results page loads the elements of interest occur at the same location. So the enginelet expects consistency in the document. But, if the format of the document changes such that the elements of interest (book name and author name) now occur at different location, the code will need some changes. The person who might make the changes for this should look into the `MyLinkFixer` class, as this class has code to locate the elements of interest.

There might be other things that can change with change at Amazon's website. The wrapper submits the user request to a particular URL at Amazon, so if that URL changes the wrapper will not be able to submit the request. The developer should make the required changes in the `AmazonLinkList` class as this class has all the URLs defined.

CHAPTER 6

PROPOSED DLSI ARCHITECTURE

6.1 Proposed Architecture

This thesis proposes a new architecture to the current DLSI. The basic processing of the DLSI remains the same, i.e., display a totally integrated system to a user. The proposal is to implement the DLSI as a Web Service.

6.2 DFD for the Proposed Architecture

Figure 6.1 shows the Data Flow Diagram (DFD) for the proposed architecture. The following is a description of the components involved in the DFD:

- **P1**: This is a query Web Service. It allows the requestor web service to query the DLSI and get information about the types of documents that DLSI can process.
- **P2**: This web service will process the documents sent by the requesting web service. It will process the type of documents that are already known to the DLSI.
- **P3**: This is a Generic Web Service. The requesting web service gives the parameters which include the document as well as the rules to find the elements of interests.

Each web service will be elaborated further on.

Data Flow Diagram for Proposed DLSI Architecture

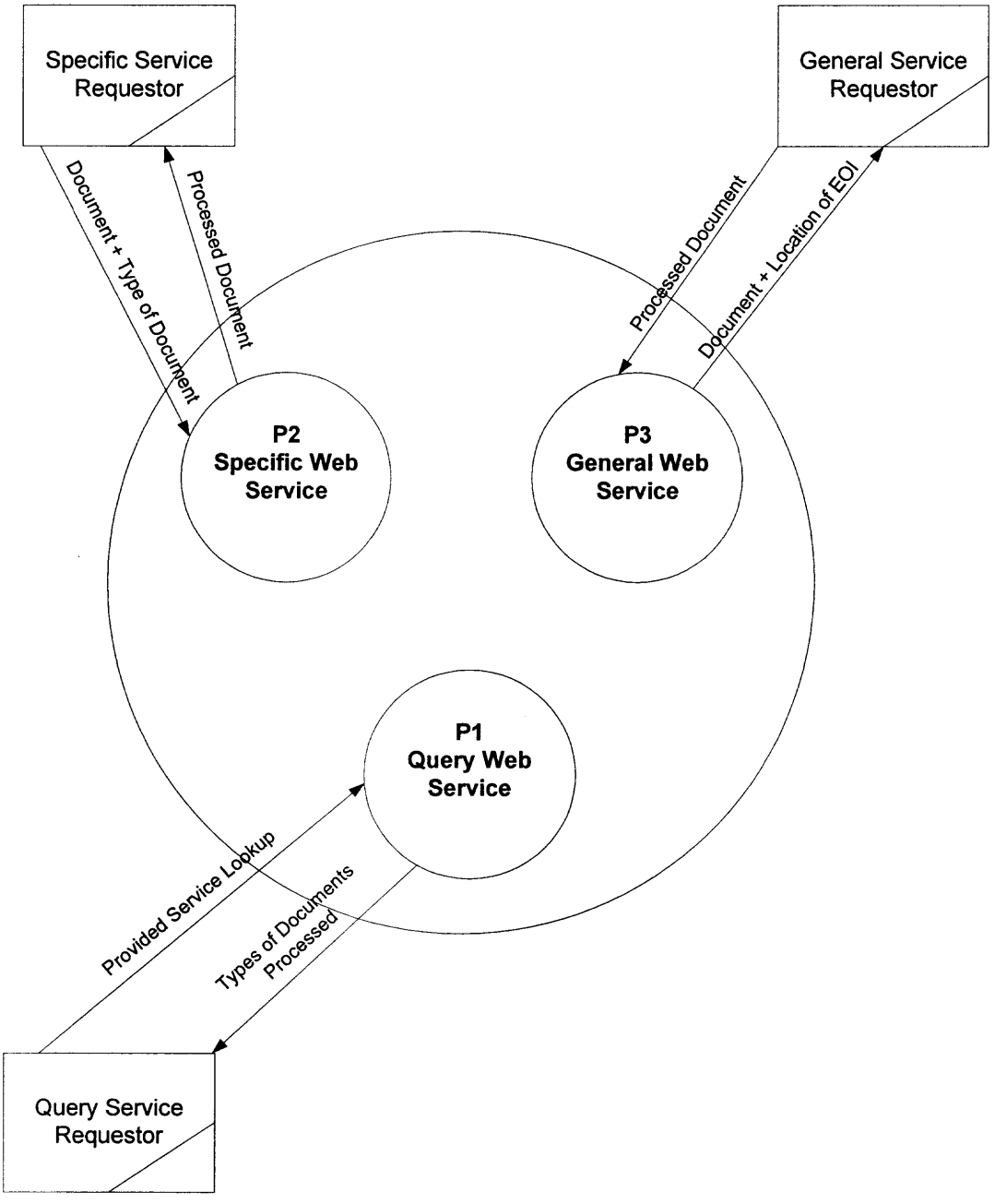

Figure 6.1 Message flow between web services in the proposed DLSI architecture.

6.2.1 Query Web Service

As the name suggests, this will be a web service which will respond to queries. This web service will send a response (once invoked) about the documents that can be processed by the DLSI. It will respond with a set of arguments containing the type of document that can be processed (like HTML, PDF, etc.) and the accompanying repository (like Amazon, NSSDC, etc.)

6.2.2 Specific Web Service

The concept of this web service is such that the requesting web service will provide the document, repository name and the type of document as the input parameters. The repository name will be required by the DLSI to route the request to the class which is written to process this repository's documents This web service will process the document using the DLSI engine in the background and then send the augmented document (with supplemental links) back to the requesting web service. The author insists upon the fact that the requesting web service should send a document instead of merely a link because this will give the DLSI the ability to process any kind of document (provided DLSI starts supporting any document).

6.2.3 General Web Service

The General Web Service can be thought of as a web service which can generate hyperlinks in a document provided the requesting service provides it with a way of locating the elements of interest along with the document. This will give the DLSI unlimited extensibility because this way it can process any document. All a service has to do is to invoke this web service with the document, format type of document and a

document data type which contains (or can be used to look up) the know-how of locating the elements of interest. The DLSI will identify the elements of interest, augment the document with link anchors and send it back to the requesting service.

6.3 Implementation Details

This section will give specifications and details about the communications between the requesting web service and DLSI's web service implementation.

As discussed before that web services use SOAP messages to communicate with each other and the SOAP messages are composed of XML, which is pure text. So, a mechanism is needed to send more than text messages between the services in communication, i.e., the web services should be able to send complex data types such as image files, word documents, etc. These kinds of messages can be handled by using the specifications given by the W3C for SOAP messages with attachments.

The guideline given by W3C describe the binding of SOAP message to be carried within a MIME (Multipurpose Internet Mail Extensions) multipart/related message such that the SOAP message can be processed as it would be processed without any attachments. In case of attachments the SOAP message package contains the SOAP message as well as entities in formats other than XML which can be called as the attachments. The MIME `Multipart/Related` encapsulation of a SOAP message is semantically equivalent to a SOAP protocol binding in that the SOAP message itself is not aware that it is being encapsulated. That is, there is nothing in the primary SOAP message proper that indicates that the SOAP message is encapsulated (W3C). The MIME attachment mechanism is a

separate topic of discussion. The details provided here are enough to understand the scenario of attachments in SOAP messages.

Query Web Service

The web service can be invoked without any parameters. The web service will return a text file containing the document format and the associated repository.

Incoming Request:

Only a request no parameters.

Outgoing Response:

<response> = <SOAP message>

<SOAP message> = <text file as MIME attachment>

Specific Web Service

Incoming Request:

The incoming request can be described in pseudo BNF as:

<request> = <SOAP message>

<SOAP message> = <document> <repository> <format>

<document> = <document as MIME attachment>

<repository> = <AMAZON> | <NSSDC> | <ASKNSDL>...

<format> = <HTML> | <PDF> | <DOC>

Outgoing Response:

<response> = <SOAP message>

<SOAP message> = <enhanced document as MIME attachment>

General Web Service

Incoming Request:

<request> = <SOAP message>

<SOAP message> = <document> <mapping rules> <format>

<document> = <document as MIME attachment>

<format> = <HTML> | <PDF> | <DOC>

Outgoing Response:

<response> = <SOAP message>

<SOAP message> = <enhanced document as MIME attachment>

REFERENCES

Aoyama, M., Weerawarna, S., Mruyama, H., Szypreski, C., Sullivan, K., & Lea, D. (2002)Web Services Engineering: Promises and Challenges. Proceedings of the 24th International ACM Conference on Software Engineering (pp. 647-648). Orlando, Florida.

Barton, J. J., Thatte, S., & Nielsen, H. H. (2000, December 11). SOAP Messages with Attachments. Retrieved April 17, 2004 from World Wide Web: http://www.w3.org/TR/SOAP-attachments.

Bernstein, P. (February 1996). Middleware: A Model for Distributed System Services. Communications of the ACM, Volume 39, Issue 2 (pp. 86-98).

Bhaumik, A., & Bieber, M. (2003, January 13). Metainformation Engine v1.0.

Cerami, E. (February 2002). Web Services Essentials (1st Ed.). O'Reilly Publications.

Christensen E., Curbera, F., Meredith, G., & Weerawarna, S. (2001, March 15). Web Services Description Language (WSDL). Retrieved April 15, 2004 from World Wide Web: http://www.w3.org/TR/wsdl.

Curbera F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerwarna, S. (2002, March/April). Unraveling the Web Services Web: An Introduction to SOAP, WSDL and UDDI. Internet Computing, IEEE, Volume: 6, Issue: 2 (pp. 86-93).

Emmerich, W. (2000). Software Engineering and Middleware: A Roadmap. Proceedings of the ACM Conference on the Future of Software Engineering (pp. 117-129). Limerick, Ireland.

Gergic, J., Kleindienst, J., Despotopoulos, Y., Soldatos, J., & Polymenakos, L. (July 2002). An Approach to Lightweight Deployment of Web Services. Proceedings of the 14th ACM International Conference on Software Engineering and Knowledge Engineering (pp. 635-640). Ischia, Italy.

Heiler, S., Siegel, M., & Zdonik, S. (1991, April 9). Heterogeneous Information Systems: Understanding Integration. Proceedings of the IEEE First International Workshop on Interoperability in Multidatabase Systems (pp. 14-21).

Mitra, N. (2003, June 24). SOAP Version 1.2 Part 0: Primer, W3C Recommendation. Retrieved April 14, 2004 from World Wide Web: http://www.w3.org/TR/soap12-part0/.

Nilsson, E. G., Nordhagen, E. K., & Oftedal, G. (1990, April 23). Aspects of Systems Integration. Proceedings of the IEEE First International Conference on Systems Integration (pp. 434-443).

Nnadi, N., & Bhaumik, A. (2003, January 25). DLSI Developer Guide.

Roy, J., & Ramanujan, A. (2001, November/December). Understanding Web services. IEEE IT Professional, Volume: 3, Issue: 6 (pp. 69-73).

Vinoski, S. (2002, March/April). Where is Middleware? IEEE Internet Computing, Volume: 6, Issue: 2 (pp. 83-85).