Identification and Retrieval of WWW-based Courseware A Meta-data Approach

Dissertation submitted to the University of Dublin, Trinity College, in partial fulfillment of the requirements for the degree of Master of Science (Computer Science)

Sarah Anne Kenny BA Mod CSLL

September 1999

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university. I agree that Trinity College Library may lend or copy this dissertation upon request.

Sarah Kenny B.A. (mod) CSLL
September 17, 1999

Acknowledgements

There are a number of people I would like to thank for the special part they played in the sucessful completion of this project. Many thanks to my supervisor Mr. Vincent Wade for all his help, encouragement and guidence throughout the course of this project. I would like to thank my parents for all of their support and encouragement throughout my years in college. Thanks to Kevin, Tarlach and my fellow classmates, for friendship, help and support.

Abstract

Information retrieval is often a tedious and time consuming task on the Internet, as information is stored in an unstructured manner. There is no guarantee that what you find is relevant to your needs, on the correct topics, up to date or written by a reliable source. In the educational domain, the focus has shifted towards an ongoing training process, where a lifelong education is required. The Web provides an ideal medium for the delivery of distance education and online courses if the resources could be found. There is a need for an agency or broker to provide a search and retrieval service for educational information.

This dissertation concerns itself with educational resources and the problems of identification and retrieval of educational resources on the World Wide Web. It intends to investigate the approaches and technologies used for the description of educational resources and to develop a suitable architecture. It will examine how information is currently represented on the World Wide Web with reference to the emerging educational standards. It will investigate the technologies used to represent educational resources with an in-depth look at the use of XML. The technology of search engines and intelligent agents will be analysed to evaluate their information retrieval methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The World Wide Web is growing at a phenomenal rate, as is the number of people using it. It has increased from over 36 million hosts in July 1998 to 56 million hosts in July 1999 and is estimated to provide access to over 800 million documents [Con]. It is comprised of a conglomeration of millions of hosts that has a formal structural hierarchy. However, the information that these hosts make available to the web lacks a global co-ordination as each host acts autonomously.

The original motivation behind the Internet was to allow information sharing within geographically and internationally dispersed teams. It was pre-dominantly used by the research community for the dissemination of experimental data and information. There were few hosts and the task of locating a particular resource was not an issue [W3Cd]. The Internet has evolved immensely since those days and it is currently being used for e-commerce, user collaboration, global communication, financial data, up to date news, travel information and many other services. One of the main reasons for this growth is the amount of information that is freely available.

Information retrieval on the World Wide Web is an area that is rapidly growing and increasing in importance. More and more people are being driven to the Internet in the hope of finding that essential piece of data that they can not find anywhere else. Yet, hours are wasted on fruitless searches, as it is often difficult to find exactly what is desired. Many think that the problem is that there is too much information available on the Internet. However, the real issue is that the lack of structure inherent in the information, renders information retrieval at best haphazard. The task of searching for information in this vibrant developing Web has become more and more tedious and time consuming.

How can a search be conducted when people are unsure of how to represent the inquiry? A common form of representation would aid queries significantly. Most people have experienced the phenomena of issuing a search query through one of the many search portals and receiving back an exponential number of results. The majority of these results are generally irrelevant, yet the issuer of the query has to sift through all possibilities until they find an accurate one. Another common problem often encountered is that the results are out of date and the information is stale. Also, there is no general form of classification available to inform people whether the data retrieved is from a source that has the authority to provide reliable, accurate facts. These are some of the issues that arise when attempting to identify and retrieve resources from the Internet.

1.2 Motivation

The educational domain on the Internet suffers from the problems of information retrieval. The number of educational resources available on the Internet has increased enormously over the last decade. This is due to significant improvements in technology and a general change in people's outlook towards education. Computers have become much faster and more powerful at prices that are now affordable to the general public. They are no longer an elitist tool. Practically anyone can connect to the Internet and take advantage of its services. Therefore, the potential audience for educational courses has increased enormously, as has the number of educators with access to the necessary technology. The quality of these resources has improved with the availability of multimedia tools for video, audio and graphics.

A metamorphosis is occurring in the way people view education. It is no longer feasible for people to train once and hope to keep the same job for the rest of their working years. We are moving towards a lifelong education where there is an ongoing training process throughout an individual's career. The current employment market requires continuous training to survive. People also demand it due to their own expectations and goals. It is not always possible for people to go back to college to re-train due to financial, personal or employment commitments. Hence, they require some mechanism of accessing the information or courses they need in the least disruptive manner possible to their everyday life.

This has led to a growth in distance education and online educational courses. There has been a paradigm shift in professional education, the classroom is no longer seen as the ultimate learning environment. [Dem97]. The benefits of other forms of teaching such as distance education has been recognised. An individual can take such a course in their own time and in the environment they find most comfortable. It also provides a way for employers to offer training to their employees without losing them for a number of days while they go off site. However, the problem of finding and accessing educational resources to suit an individual is still there.

There is no database or list containing all of the educational resources or services available on the Internet. Therefore the task of finding the course that best suits an individual can often be very difficult and time-consuming. There is a need for a broker or an agency service to provide a way for people to find and retrieve the educational resources that fit their particular requirements. The Broker would undergo the searching process of all the educational resources available online and return accurate and relevant results from reliable authorities.

1.3 Objectives

The following are the objectives of this thesis.

1. Investigate approaches and technologies for the description of educational resources and services.
2. Develop an architecture suitable for the identification and retrieval of educational resources.
3. Prototype proof of concept demonstrator which supports automatic and semi-automatic retrieval of educational resources and services.

1.4 Approach

The intention of this dissertation is to investigate the approaches and technologies used for the description of educational resources. Firstly, the educational resources will be examined to determine how they are described. This will incorporate an in-depth look at both the existing and the evolving educational representation standards and the technologies used to define the format of educational resources.

This will be followed by an examination of the various search technologies that are currently available for information retrieval on the World Wide Web. The tools currently used for the creation of metadata are discussed. Finally, the semantics and technologies will all be considered with a view to developing an architecture for accessing heterogeneous information sites, both automatically and semi-automatically on the World Wide Web.

1.5 Structure of the Report

The report has the following structure:

- Chapter 1 Introduction

  The motivation and aims of the project are outlined, followed by the approach adopted and the structure of the report.

- Chapter 2 Educational Metadata and WWW based Retrieval

  The semantics, standards and technology used to describe educational resources are examined in chapter two, along with an analysis of information retrieval methods for these resources. It also details a number of tools, which are currently being used for the creation of XML metadata.

- Chapter 3 GESTALT

  Chapter three outlines the GESTALT European research project. It provides an alternative perspective on how the tool created throughout this research could be used.

- Chapter 4 Design

  This chapter outlines the requirements and the design of the system.

- Chapter 5 Implementation

  Chapter five details the tools used in the design of the system. It explores in greater depth the various classes used within the system and provides the implementation of them.

- Chapter 6 Evaluation and Conclusion

  Chapter six closes the report and project. It provides an evaluation of the objectives of the research and the tool developed. It describes what has been achieved, and draws conclusions on the success of the tool. It also provides suggestions for future work in this field to turn the tool into a commercial product.

Chapter 2

Educational Metadata and WWW based Retrieval

2.1 Introduction

This chapter looks into the background of this project. Firstly, the semantics of how information is represented in the educational domain is presented. This shows the motivation for the creation of metadata. The material was looked at in terms of semantic representation, standards in the area and projects being implemented using the current standards and contributing towards them. This is followed by an analysis of the technology used to represent educational resources. The technologies used to retrieve the information from the Internet are examined and compared. Finally, the existing metadata tools are presented.

2.2 Motivation

As the web continues to grow, more and more educational resources and courses are being developed and made available on the Internet. These courses are generally being developed independently by various institutions and companies in different parts of the world, resulting in the development of many different structures and course types. The semantics of how the educational resources are represented is a very important issue in the development of identification and retrieval systems for educational resources on the Internet. A common structure or standard format would improve information retrieval and could also enable automated searching. It is difficult for machines to deal with information on our behalf. This is because representing information about things, in a way that is easy for machines to deal with is not well defined. This has lead to the development of Metadata for educational resources.

Metadata is "data about the data". It is a description of objects, documents or services which may contain data about their form and content. It may be part of the resources themselves or kept separately from them [JHK96]. Metadata allows learning resources to be described digitally by attaching a label to the object. The label provides information about the contents of its container without having to actually open the container. The descriptive information metadata supplies, allows the user to locate, evaluate, access and manage online available learning resources. In essence, metadata serves as a complement to its contents and reflects the content's attributes to interested users. It facilitates document cataloging and searching as it helps authors to describe their documents in ways that search engines, browsers and Web crawlers can understand. Work on web based metadata is being managed by the W3C [W3Cc].

2.3 Models

A number of standard bodies were established to try and identify a set of guiding principles which can be adopted by educational institutions for the development of educational resources. They aim to represent the contents of the information in an independent manner to the way in which the information is stored. This section examines a number of the emerging standards bodies and describes the metadata models they have produced.

2.3.1 DC

The Dublin Core (DC) group is an international group that aims to create a common core of semantics for resource description on the Internet. The group is composed of policy leaders from various disciplines and geographical areas along with a number of working and technical groups.

The original motivation behind the Dublin Core in 1995 was the description of author-generated Web resources. However, it now caters for resources that are both digital and exist in traditional formats. It is used by a number of formal resource description communities such as museums, libraries, government agencies and commercial organisations. This information is described in more detail on the DC homepage [Corb].

The Dublin Core Standard is a simple Content Description Model for Electronic resources. It is based on a metadata element set and its intended use is to facilitate the discovery of electronic resources. The standard was composed of 15 elements in version 1, but this has just been reducded to 10 elements in version 1.1. The elements are outlined in greater detail on the Dublin Core Web Site [Cora]. The element's names are descriptive enough to convey a certain meaning, e.g. Title, Author, Subject and Publisher. Each element has a limited set of qualifiers and attributes that may be used to further refine the meaning of the element. Each element is optional and may be repeated. It takes the form of:

Label: value

It can be used in the following way:

Publisher: Sarah Kenny

The Dublin Core has recently announced their intention to set up an Education working group. The working group plans to discuss and develop a proposal for the use of Dublin Core metadata in the description of educational resources.

The Dublin Core element set has a number of distinct advantages over other metadata resource description models. It has a very simplistic structure as its intended use was by non-cataloguers as well as resource description specialists. This facilitates both machine processing and automatic retrieval of resources encoded using the Dublin Core element set. Its simple structure provides a good base for the description of general resources, which can be refined and extended further for specialised domains such as the Educational domain. However, sometimes the structure is viewed as too simplistic and any real application requires a further extension of the element set.

Another key advantage of the Dublin Core is it's semantic interoperability. It promotes a commonly understood set of descriptors that helps to unify other data content standards across different disciplines. This has also lead to widespread acceptance and international recognition in the development of an effective discovery infrastructure. Dublin Core includes sufficient flexibility and extensibility to encode the structure and more elaborate semantics inherent in richer description standards.

The Dublin Core also provides a number of tools related to Metadata. It provides a Metadata tool for the creation and modification of metadata templates, automatic extraction and gathering of metadata tools, conversion tools between various metadata form formats and integrated tool environment tools.

2.3.2 IMS

The Instructional Management Systems (IMS) [oST] is a US based consortium group composed of educational institutions in the USA. IMS is sponsored by EDUCAUSE [EDU], an American professional association that addresses the application of technology in higher education. The educational institutions can contribute to the consortium by becoming active members at an investment level or a development level. An investment level member means that the institution is represented on both the Advisory Board and Technical Board so they have more control over the specifications. A Development level member gets the specifications as soon as they have been approved as a draft.

IMS is developing and promoting open specifications for facilitating online activities such as locating and using educational content, tracking learner progress, reporting learner performance and exchanging student records between administrative systems. The goal of IMS is to promote the widespread adoption of specifications. This will facilitate distributed learning environments where resources from different authors can be used together seamlessly. The motivation behind IMS was the development of instructional software and its integration into the learning environment. This was being impeded by the lack of standards to permit sharing across institutions and across a wide range of technical environments [Proa].

IMS is involved in writing technical specifications. It has written the IMS Metadata Specification [oST] where it focuses on the structure of the data. It aims to provide a common metadata structure for educational resources. This is to facilitate both the discovery and management of these resources on the Internet. The IMS specification provides the labels for learning resources on the Internet and it is Dublin Core compatible, although it does have a number of extensions. In order to describe a resource, particular characteristics of the resource must be identified. Multiple categories provide several different paths to the same object. These descriptive categories form the object metadata.

The Specification is comprised of two parts, a dictionary of fields and values and the organisation of these fields and values into groups [prob]. These are represented in XML/RDF format as specified by the W3C [W3Cb]. Essentially, IMS defines the terms to be used while W3C gives the format of these terms so that applications such as Web browsers can process the metadata automatically. The general structure is shown below.

```
Course
    Category
        Field name value pairs (repeated to any depth or level)
```

The IMS specification provides a common standard for educational resources based on the general Dublin Core set. It has quite a simple structure although it can become quite deep with repeated field name value pairs. The structure is compliant with the W3C XML specifications ensuring its compatibility with Web Browsers.

2.3.3 IEEE

The Institute of Electrical and Electronic Engineers (IEEE) is an international organisation that aims to promote engineering and foster technical development in the world. It is composed of a number of working groups each striving to develop standards in different areas. The Learning Technology Standards Committee (LTSC) is an IEEE committee that is trying to develop the ways and means to deploy education and training systems, or as they put it "The mission of IEEE LTSC working groups is to develop technical Standards, Recommended Practices, and Guides for software components, tools, technologies and design methods that facilitate the development, deployment, maintenance and interoperation of computer implementations of education and training components and systems" [IEEa]. The LTSC has a number of different working and study groups for different standards including The Learning Objects Metadata Working Group.

The Learning Objects Metadata (LOM) Working Group is writing a standard that will specify the syntax and semantics of Learning Object Metadata. The standard represents all of the information-collected to-date, input from the working group and existing work being done in this area. The IMS Project (section 2.3.2) and ARIADNE project (section 2.4.1) which were both based on the Dublin Core standard (section 2.3.1), have both contributed a lot of work to the LOM.

"Learning Objects are defined as any entity, digital or non-digital which can be re-used or referenced during technology supported learning. The LOM standards will focus on the minimal set of properties needed to allow these learning Objects to be managed, located and evaluated" [IEEb].

LOM has a hierarchical structured metadata model with three levels. It consists of categories, data elements and abstract data types. The general form of the structure is a top level of categories, each category contains a middle level of data elements which in turn are defined by the final level of abstract data types.

There are 8 current categories that provide the top-level organisation of the data elements and create the context in which the data elements are evaluated.

The general structure is:

Category
    Data Element: Abstract Data Type

It can be specified as shown below.

Characteristics
    Language:Dlocale

It may be used in the following way

Characteristics.Language: Us_en

The IEEE LOM has a very complex structure that can take advantage of the power of XML. It originates from the IMS (section 2.3.2) and ARIADNE (section 2.4.1) projects, so it has taken the best characteristics without being burdened down by their respective problems. The LOM is also compatible with the Dublin Core element set and can be mapped directly to the base structure. LOM has a tree structure, where the leaves are field name value pairs and can be searched easily by a machine. IMS are likely to migrate to the LOM structure [DWN^{+}99]. The LOM has quite a heavyweight structure, which will increase the complexity of tools written to deal with the LOM.

2.4 Projects using the Models

There are two European projects of significance actively contributing and furthering the development of the IEEE LOM standards. These are ARIADNE and GESTALT, which are described in the following sections.

2.4.1 ARIADNE

The Alliance of Remote Instructional Authoring and Distribution Networks for Europe (ARIADNE) has been involved in standardisation activities performed under the IEEE LTSC Committee since December 1997. ARIADNE consists of corporate users, made up of representatives from all sponsoring organisations and academic users, which is open to members of all European public service universities and higher education institutions. It is a European implementation project that uses the emerging standards and is actively contributing to the development of educational standards especially the IEEE standards. Its goal is to foster the sharing and reuse of electronic pedagogical material, by building up an international system of interconnected knowledge pools. It focuses on the semantics of the data. [ARI]

ARIADNE has been evolving in a number of different stages. Firstly, prototype tools and basic methodologies where developed for maintaining and exploiting the knowledge pool system in a number of educational and training areas. Then the intention was to perfect these tools further and validate them through large-scale demonstration. Overall, ARIADNE hopes that the evolved schemes and set of tools developed will contribute to a European standard for collaborative and trans-cultural Information Technology based training approaches. It provides tools for authoring, pedagogic hypertext generator and validation tool, query tool, curriculum editor, knowledge pool system and learner interface [ARI].

ARIADNE uses a simple two level structure that is based on the Dublin Core metadata specification (section 2.3.1) to categorise its metadata. It is composed of a Course, Category and Field name pair. The general structure is presented below.

```
Course
    Category
        Field name Pair.
```

The Two levelled structure of ARIADNE is easy for a user to comprehend and search. It can be dealt with automatically by machines and retrieval agents. It does not have a wide enough acceptance to achieve interoperability but it is actively contributing to the IEEE LOM standards.

ARIADNE and the IMS project have recently agreed to collaborate and work towards a common structure and core of educational metadata descriptors which they will then propose for international standardisation [ARIMS].

2.4.2 GESTALT

Getting Educational Systems Talking Across Leading-edge Technologies (GESTALT) is a European research project funded under the ACTs programme. Chapter three describes the GESTALT project in detail, as it provides a broader context for the development of this thesis. The motivation behind GESTALT along with the project's aims and metadata model will be discussed.

GESTALT has evolved from a combination of the IEEE LOM standard and the IMS standard. It has its own metadata model described in section 3.2 and it is active in the IEEE standardisation process.

2.5 Projects implementing Educational Retrieval Systems

An investigation was carried out to discover whether any of the metadata model groups had created any search and retrieval tools for educational resources.

IMS was found to be more concerned with defining technical specifications that products should follow in order to work together. It did not have any educational resource retrieval tools although it stated in the projects scope that it intended to develop specifications for searching for resources who's content could run on any learning server. However, there were no collaborative projects to be found in the process of implementing such a search system. Instead it focused on educational authoring tools and management systems for looking after the course content.

The ARIADNE project has not addressed the issue of search and retrieval tools for educational courseware. It is more focused on the Learning Environment and the creation of authoring and management tools for the sharing of knowledge within this type of environment. It has built tools to author courses and curriculums, to validate pedagogic material, generate multimedia enriched courseware and evaluation student performance.

The GESTALT project is in the process of developing an educational resource retrieval and discovery service as part of its framework. This is described further in chapter 3 which is devoted to the GESTALT project.

GESTALT was found to be alone in its quest to create a broker system for educational resources. The other educational tools focused more on the creation of educational courseware and pedagogic material.

2.6 Technology

The primary technology used for the representation of educational resources metadata is XML. The following section describes the origins of the langage and why it is particularly useful for resource description on the Internet. The DOM is another technology often used with XML, it is also discussed in this section.

2.6.1 XML

The eXtended Mark-up Language (XML) is published by W3C [W3Cb]. This mark-up language takes it's origins from both SGML the Standard Generalised Mark-up Language (SGML) and Hyper Text Mark-up Language (HTML). SGML is a language used to describe any type of document. It is a very powerful language because of its ability to store the content separately from the data. However, its syntax and structure are very complicated and proved very difficult for machines to deal with automatically. HTML has quite a simple structure and is basically free format text. It is quite limited in its way of representing data as it was aimed at creating pages for browsers to interpret on the Web. Unfortunately, it does not deal very well with data content. XML was thought to be the solution to these problems.

XML allows documents with a flexible data structure to be created, where the content can be stored separately from the data. However, its structure is not as complicated as the structure of SGML. Since XML allows data to be stored in a structured format, data can be processed by many applications, which is unlike HTML as it can only be browsed. XML also permits fields to be tagged based on author defined tags.

XML was originally intended to enable metadata to be embedded in web pages. This allows much more accurate search and retrieval of resources using the Internet. However, XML is in fact a self-declarative meta-language, thus lending itself to a wider range of uses than just metadata for web pages. It can define metadata for a wide range of document types. There are two significant parts to XML called the Document Type Data (DTD) and the Resource Description Format (RDF) [St.97].

The Resource Description Format is a language for representing metadata. It was developed as a result of the W3C's interest in metadata. RDF is destined to support a wide range of applications over the web and the wider Internet, including search engine data collection (web crawling) and access to digital library collections. Its key focus on interoperability in the exchange of metadata and it uses XML as its transfer mechanism. RDF is an application of the Extensible Mark-up Language (XML). Applications written in the RDF/XML language can adopt quite different headings and categories when it comes to organising material. The vocabulary is likely to be application-specific. RDF leaves metadata authors free to choose the vocabulary of their choice, spelling out in detail the allowed use of the vocabulary - essentially the 'grammar' of the application. The Resource Description Format Model and Syntax Specification is complete and is a W3C Recommendation since February 24th, 1999. The Resource Description Framework Schema Specification is nearing completion and was made a Proposed Recommendation on March 3, 1999.

Document Type Definition (DTD) is a data structure specification that can be applied to documents. It provides a formal definition for the mark-up rules. XML requires documents to conform to a complex set of specifications outlined in the DTD. The XML processor has two jobs, it ensures that the documents are valid in that they conform to a DTD and then it builds a document tree structure that it passes to applications. The DTD provides a critical link between the data files given to the XML processor and the data that is transmitted from the XML processor to the application. Document Type Definitions help computers to understand structures that may seem obvious to humans. They can appear in the document they describe or in a separate file.

2.6.2 DOM

The Document Object Model (DOM) is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents [W3Ca].

The model was created to support both HTML and XML by the W3C. An XML document is essentially, a plain text file. An interpreter must be written to interpret the file in such a manner that it can be used. This process is time consuming and repeated throughout the world. This is what motivated the W3C to create the DOM. It provides a standard way of parsing and accessing the document's contents.

Since XML provides more processing functionality than HTML and it is extensible, users were seen to require the ability to program XML. A standard Application Programming Interface (API) was designed to increase the interoperability of documents sent over the Internet. The development of modules for different applications is also facilitated by the existence of a standard API.

2.7 Search Technologies

Various search technologies are used to retrieve information on the Internet. Search Engines and Intelligent Agents are two contrasting retrieval methods. These will be discussed in the following section along with a brief comparison of the two technologies at a conceptual level.

2.7.1 Search Engines

A search engine provides users with a way of searching for information on the Internet. A user inputs a query into a search engine and it searches its database for references to the input, then it returns the number of references to the user. Behind every search engine, there is a database which stores all of the (Uniform Resource Locators or specially formatted Internet addresses) of web pages and other Net resources.

Most of the databases are created by crawlers, which are also known as robots or spiders. Crawlers are software programs that roam the Web looking for new sites by following links from page to page. They are based on the concept that every page on the web must be linked to another page and by searching through a large number of pages and following all of the links, a user will discover new pages with a new collection of links. When a spider finds a new page, it adds information about the page and its URL to the database, which can store data about a few million pages.

The size of the engine's database has a big impact on the success of the user's search. At the moment, there is no search engine powerful enough to index the whole Internet. The search engines database is simply an index of words and phrases associated with URLs. When a search is performed, what actually happens is that the databases index is searched. There are many different ways for engines to index their database including indexing web pages URL and title, web headers, the most frequently mentioned words in a web page, the first few lines of text, every word on the whole page and some engines even employ concept-based indexes.

There are two main types of search systems that are in popular use on the Internet. The first is a 'keyword' search system where the user types one or more descriptive words from which the system displays a collection of web pages that contain them, e.g. Altavista [Alt]. The second type extends this principle with the addition of a subject tree. This allows the user to browse and explore the subject tree in depth, and perform keyword searches within different areas, e.g. Yahoo! [Yah]

Both systems are centralised in the way they retrieve web pages from the Internet, and in the way they store all this information in one web location. Meta-engines forward search queries to a number of major web engines at once. They function by reformatting the search engine output from the various engines so that it indexes into one concise page. The problem with meta-engines is that they lose the capabilities of advanced search at the sites.

There are important differences between the search services, as far as size, selection of the indexed servers, update period, user interface and support is concerned. Each Search Engine uses different retrieval software and searching functions leading to varying results from simultaneous search engines. Also the update periods vary from weekly up to once a year. The larger centralised services normally have longer periods between re-indexing the same page or server.

In many cases, not all the indexed information about a document is displayed as part of the search result. Some services show the title of the hits only. Best-match methods are predominant as search software, but the respective ranking algorithms are rarely described in a satisfactory way. The services focus in general on high recall. Precision improving facilities such as relevance feedback or vocabulary control are seldom used. A variant of citation searching in hypertext, provided by AltaVista [Alt] and WWW Worm, is amongst the most advanced applications.

Search Engines like all other technologies have a number of weaknesses. They gather everything into a central location and run the search queries there. This gives rise to slow performance and demands considerable resources. With web pages being added to the Internet at an exponential rate, the problem of indexing new pages arises. The search engine must maintain the index with the most up-to-date information possible. There is also theoretical problem of indexing virtual hypertext. As the depth of hyperspace is infinite, an infinite index would be required to index it. [Tec].

Search Engines are free to use any information retrieval software they choose. They vary greatly in terms of the harvesting methods, the indexing methods and the retrieval software. The results returned to a user depend on the use of exact match or best match, natural language and linguistic methods, ranking algorithms and the use of relevance feedback. The combination of techniques does not always complement the search process.

2.7.2 Intelligent Agents

An intelligent agent is computer code that can act autonomously on behalf of a user or a process. It is called intelligent because it can perform actions with some level of pro-activity or reactivity. It also exhibits some level of the key attributes of learning, co-operation and mobility. Agents are able to communicate with the user, the system and each other. Agents can work collaboratively to help each other complete a task or they can work against each other. Multi-agent systems provide a network of problem solvers that work together to solve problems that are beyond their individual capabilities [GHC97].

John Vittal developed the first operational intelligent agent system in 1979 at the company Bolt, Beranek & Newman, although the term 'agent' was first mentioned by Thomas Standish in 1971 [Fis99]. However, agent technology has only recently entered the commercial arena and increased in both use and popularity. The increase in use of Intelligent Agent technology has lead to the need for standards. The Agent Society is an international organisation that was established to assist in the widespread development and emergence of intelligent agent technologies and markets [Soc]. It is working towards standardising the intelligent agent technology. The Foundation for Intelligent Physical Agents (FIPA) is an organisation that is developing interoperable specifications and standards for generic agent technologies to work across agent based applications [FIP].

Intelligent Agent technology can be used in the field of information retrieval. The Internet provides an ideal domain for multi-agent technology as the agents can address problems that may be too large for a centralised single agent. A Multi-agent system can be used to search for and retrieve data about a particular educational topic. Intelligent agents provide a distributed search system so the searching task is broken down into a number of smaller operations. This increases both the speed and performance as the operations can be performed in parallel. However, there must be some way of dividing up the work amongst the agents, collating the results and communicating between the agents. These tasks all incur a certain overhead in performance. Intelligent Agents can also deal with uncertain data and knowledge, a characteristic that is very desirable in a search system.

The OASIS service described in [IN] and [MB] presents a distributed system of Internet Intelligent search agents. Each agent built a collection relevant to a certain topic. The OASIS Crawler constructed an index of Internet documents relevant to its associated OASIS collection. (The Collections index is essentially the same as a search engine's index and composed of the same data.) Crawlers worked collaboratively and gave each other URLs that were thought to be of relevance to their collection. A harvesting strategy was required by the Collection to filter out documents that were not relevant. An OASIS Server was used to process the users query and to search particular collections. It had to rank the query results in order of relevance.

The distributed search service in the OASIS service performed faster than a centralised search service. It also proved to be very effective in searches on very specific topics yet it was found to be imprecise on general searches. There was a need for inter-communication standards, as the agents needed to communicate in order to propagate URLs and increase the size of the Collections.

Searchbots developed in the University of Massachusetts is another example of a co-operative information gathering multi-agent system [Wag]. It differs from the OASIS system in many ways. Searchbot agents perform live search at heterogeneous remote sites via the Internet. They focus on goal driven information gathering. Each agent searches a different domain for relevant data and co-ordinates results in order to locate additional sources of information. The results are then assembled and returned to th e user within a certain time frame. These agents suffer from much the same communication and co-ordination overheads as OASIS. The OASIS system was more accurate than Searchbots due to their topic-specific nature. Searchbots were limited in the information they could retrieve due to their live search operation.

Intelligent Agent technology raise certain security issues due to their ability to move from one host to another. An agent operating on a machine can suddenly halt execution, move to a remote host and resume execution there. It takes its program code and state with it. This means that certain security measures must be enforced to ensure that a destructive agent does not execute any problem inducing code on a remote machine. Intelligent Agents are very effective when performing information retrieval on a group of Web Servers where their access to materials can be controlled. The system Mobile Assistant Programming (MAP) described in [SP] searches for relevant HTML documents on a set of WWW servers. However, people generally regard agents as 'untrusted' code and are wary of them. There is a need for standards to restrict the area an intelligent agent has access to, perhaps similar to the restrictions imposed on CGI programs in a web server. A number of companies have begun implementing agents with security restrictions such as IBM's Aglets [IBMa].

The number of intelligent agents currently available and able to perform search and retrieval operations on the World Wide Web is quite limited. They are largely research projects and prototypical systems. The architecture necessary for the widespread use of Intelligent Agents with the necessary security restrictions has not yet been implemented on the Internet. This is curtailing the use of Intelligent Agents in the Information retrieval domain.

2.7.3 A comparison of Search Technologies

In general search engines employ a centralised system while intelligent agents use a distributed system. This has many repercussions for both systems. Search Engines require a huge amount of resources and computing power hence making them costly in terms of money and performance[MB][IN][KABL98]. The search engine downloads all pages to its centralised site where it performs its indexing operations. The index is stored at this centralised site and depending on the number of users accessing the search engine at a give n time, it can become a bottleneck and cause long delays in returning theresults to the user. The distributed system suffers from problems of a different sort. Each agent goes to a site where it examines the pages and sends information back to its index. Each agent has it's own index. It has to incur the overhead of communication between the different agents in the system and co-ordinate search activities between them. However it usually has faster results than a centralised site because it can search each of its indices in parallel [Tec].

The Search Engine is very good for a general search, it usually has some information about just about every subject in its database. However, if you are looking for very specific information about a particular topic, it does not always return relevant resources of documents. An Intelligent Agent is often built to retrieve information on a specific topic, therefore if you are looking for something very specific it can often return good results. There are a number of topics that the intelligent agents leave out making them quite poor at retrieving general information[IN].

The Search Engine system does not raise many security issues. It goes to a web site and downloads what it needs without running any programs on the site. An Intelligent Agent on the other hand often executes code at the remote site. This has security implications. The remote site must trust the agent not to run any virus programs or restrict the agents access to certain files. These are issues that need to be addressed in the global intelligent agent standards.

Search Engines are a well-established technology. They have been tried and tested over the last few years. They have achieved wide spread acceptance and use. They have been written for many domains and spoken languages. There is an enormous number of search engines available of differing qualities which can be bought or downloaded free. Intelligent Agent technology has been around for a number of years but its only recently that they have been used as information retrieval agents on the Internet. The information retrieval agents are generally at the prototype stage with a number of them being implemented. They are in limited supply and only few examples of them exist leading to a poor choice.

A large number of the Search Engines have implemented metadata support at their sites and use them for their information retrieval activities. Since Intelligent Agent search technology is still emerging, they offer little support for metadata. They are only beginning to incorporate metadata into their information retrieval processes.

Both Search Technologies have many differing abilities and each offers advantages over the other. However, the architecture for implementing Intelligent Agents has not yet been fully applied to the Internet.

2.8 Editing Tools for XML metadata

There are only a very small number of XML editing tools on the market at the moment due to the emerging state of the technology. A number of SGML editing tools have been adapted to support XML development but apart from that, there are only a few XML tools available, and they are expensive.

- XMLPro

  XMLPro is a "Pro eXtensible Mark-up Language" editor for XML. It was developed by the company Vervet Logic [Log] and was one of the first XML editors produced specifically for XML. It allows XML documents to be created and editing using menus and wizards. It does not support DTD creation but it does allow them to be imported into the tool, for the validation of the XML.

  XMLPro is not aimed at novice users and a working knowledge of SGML is assumed. The nomenclature is left up to the user to interpret and there is no glossary or help available to explain it. It imposes a learning curve on the average user and very little help. The interface consists of a number of pop-up boxes and menu bars. It will only run on window type systems.

- Visual XML

  Visual XML is a tool that enables you to create and modify DTD and XML documents. It is an open source project currently in development by Pierre Moral and his team [Mor]. However it provides little more than a Java-based text editor does.

- Xeena

  Xeena [IBMd] is an XML editor that enables the user to create and edit an XML document as long as it is given a valid DTD. It has a simple interface based on a tree directed view. It is easy to use and only valid elements or constructs are presented to the user so that the structure in the DTD is enforced.

All of these tools are general XML editing tools. They cater for any general XML document being created from a general DTD. There is nothing to guide the user or help them with any specific DTD. They are generic tools aimed to work in all areas. If a user wants to use these editors in a specific domain such as the educational domain, the tools do not provide them with any assistance. The tools do not have any knowledge about the LOM model to direct them in their functionality.

The tools range from XMLPro and Visual XML being difficult to use and not very user-friendly, to Xeena proving quite a simple usable interface. They were all written in Java and depend heavily on existing parser frameworks by SUN and IBM.

The tools do not always provide validation. The process of validation involves comparing the XML created against the documents DTD to ensure that there is no violation of the DTD's rules. Xeena forces the user to use a valid DTD so that it can validate documents. XMLPro requires the user to import the DTD into the document in order to carry out validation. The position of Visual XML with regard to validation is not known.

After completing an investigation of the currently available tools, it has become apparent that there is a need for a directed tool for use in the educational area. It would be more beneficial to an educational user if the tool had knowledge of the LOM model. This would greatly aid a user in their task of creating educational metadata. A tool of this kind must be user friendly, easy to use and extensible. The tree-based interface in Xeena was found to be simple and usable for the user and it enforced the structure defined in the DTD.

2.9 Summary

This chapter examined the semantics of how information is represented and presented the educational metadata models and educational implementation projects contributing towards the emerging standards. It described XML, the metadata representation technology and discussed the two main search technologies. The chapter concluded with a brief look at what is currently available in the area of metadata creation tools.

Chapter 3

GESTALT Architecture

3.1 Introduction

This chapter introduces the European project GESTALT in which the research carried out through the course of this thesis plays a role. The motivations and intended use of GESTALT are outlined. This is followed by a brief description of the architecture and the discrete components, which are envisaged as comprising the complete GESTALT system. There is a component of special interest to this thesis called the Resource Discovery Service. It is examined in further detail in order to show how the system developed in this thesis could be used as a working part in the Resource Discovery Service.

3.2 What is GESTALT?

Getting Educational Systems Talking Across Leading-edge Technologies (GESTALT) is an Advanced Communications Technologies and Services (ACTS) [1] it is the focus of the EU's research effort to accelerate sponsored European research project. It has been examining the area of remote learning environments in which services such as Resource Discovery, Learning Environment, Student Profiles and Asset Management are required for the student and lecturer to work remotely on-line. GESTALT consists of a number of prominent European businesses along with academic staff and researchers from universities throughout Europe.

The GESTALT vision is concerned with defining ways that will enable learning technology products to inter-operate in remote learning environments. They hope to achieve this vision by the use of newly available and emerging technologies. GESTALT intends to perform integrated trials and construct an online training demonstrator for their resource discovery service, learning environment, student profiler and asset management system [Con98].

In GESTALT metadata has been associated with educational material for use in the educational domain. Metadata enables inter application communication and more efficient searching and discovery. The GESTALT metadata structures have evolved from a combination of the IEEE LOM standard as described in

[1]The Advanced Communications Technologies and Services (ACTS) is a European Union collaborative research and development programme. It aims to accelerate the deployment of advanced communications infrastructures and services in the fields of information technology and telematics [Tc].

section 2.3.3 and the IMS standard described in section 2.3.2. However, due to the requirements of the GESTALT project some additions and modifications were necessary. These are contained in the Gestalt Extensions to Metadata Standards for ON-line Education Systems (GEMSTONES). The metadata model used by GESTALT is essentially GEMSTONES. A DTD based on this model was written.

3.3 GESTALT Architecture

The initial GESTALT Architecture is depicted in figure 3.1. The system is only complete when composed of the set of components shown above. These components are outlined below as described in [PF99].

Web Client

It is assumed that all User Services will be delivered to the desktop, using World Wide Web technology.

Resource Discovery Services

This is comprised of a CORBA-compliant Broker accessed via a web gateway, these services will allow users to explore what courses and modules are available from which institutions.

Learning Environment

The LE database will be used to manage access and on-programme progress

User Profiles

This will store user preferences for LE interaction.

External Internet Services

These would include services to allow users to search (using metadata) descriptions and access resources beyond those available on their immediate programme of study.

Asset Management System

This controls access to valued resources such as learning objects which should only be accessed within a paid for programme of study.

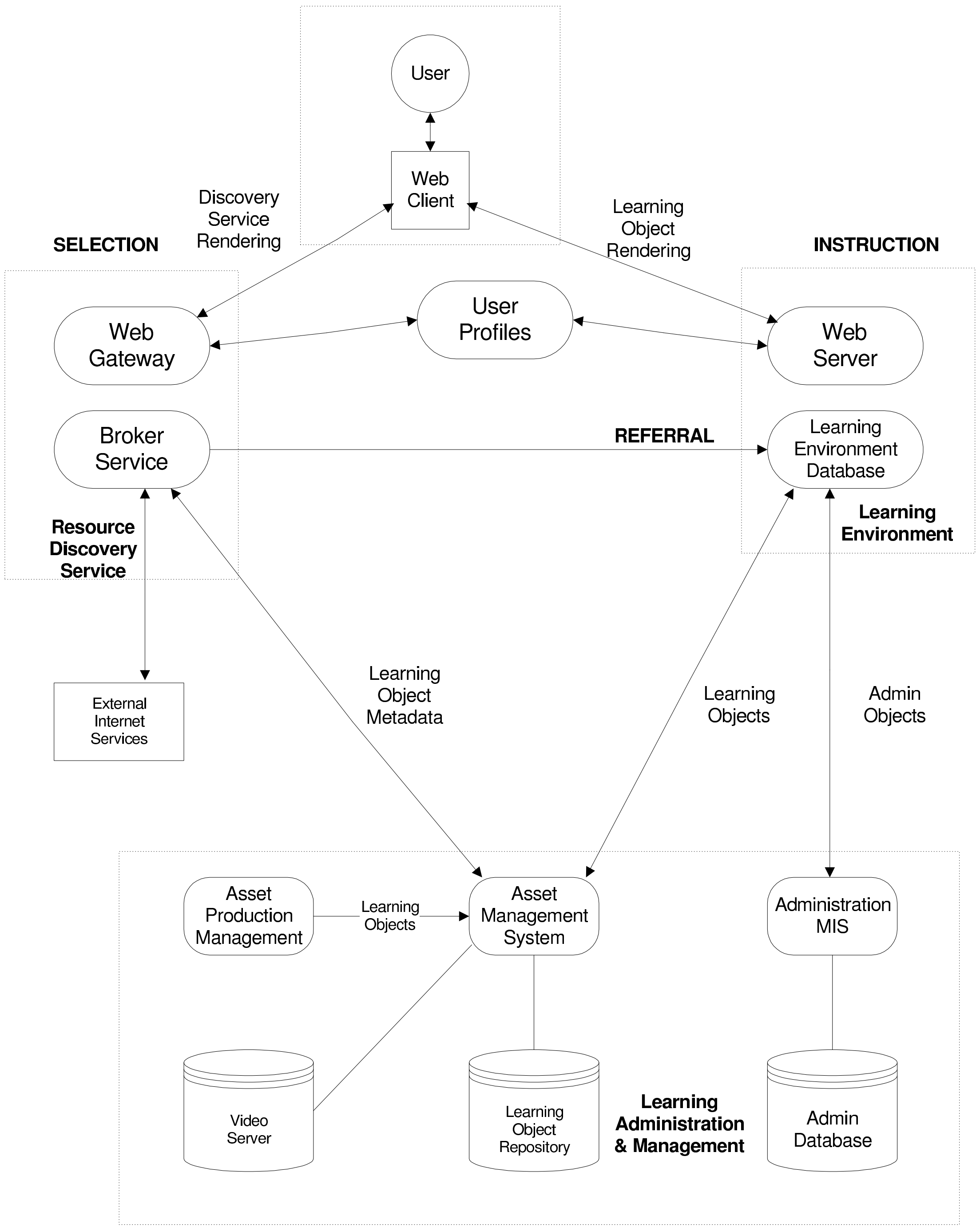


Figure 3.1: The GESTALT Architecture

Administration MIS

> It supports back of house management of the training institution. It relates learning outcomes and student progress as assessed by the Learning Environment to learning objectives held in the MIS.

Asset Production Management

> This system is for co-ordinating production of multimedia resources and its related metadata.

Learning Object Repository

> It contains one or more file stores, firewall protected to prevent access other than via the Asset Management System.

Video Server

> It is a specialist technology for supporting multiple access to streamed multimedia services.

Admin DBMS

> This contains one or more file stores, firewall protected to prevent access other than via the Administration MIS.

All of the components in the system interact together to provide a number of functions. A user can request a course from the selection component. This checks their requirements in their user profile and creates one for them if necessary. The requirements contain the user preferences, pre-requisite knowledge and any other information that could be relevant in finding a course to suit the users needs. The broker service takes the requirements and searches for matching courses. The broker service interacts with the asset management system and makes a It rning environment database. Once the user has received the names of the courses and resources that were found to match his requirements from the broker, he can then request them from the relevant sources. The sources have been pre-warned by the broker's referral and expect the user.

Although GESTALT consists of an Asset Management System, a Learning Environment and a Resource Discovery Service, it is the Resource Discovery Service that is of special interest in this thesis, as it provides a means by which potential students and learners can locate and order educational courses and resources.

3.4 Resource Discovery Service

The purpose of the Resource Discovery Service (RDS) is to help customers locate educational courses and resources. It offers facilities for searching, previewing and ordering them. These services are based on the brokerage functional architecture developed by the GAIA ACTS [2] project with the addition of a domain specific delivery management system.

XML based metadata is fundamental to the design of the RDS. Metadata provides a mechanism for resources to be described digitally. In this way, it is possible to structure and tag information available on the Internet. This can potentially transform the Internet from a resource which was previously accessed by Web Servers and Search Engines into a resource which can be accessed by other more domain specific applications, in this case Education Systems.

The advantage of using a RDS is that users do not have to try and locate educational resources and courses manually. It prevents the user wasting time searching for courses that may not be relevant or aimed at a different user level. The RDS provides metadata information about educational resources that can be compared against the needs of a user as defined in the user profile.

Architecturally the RDS uses a three tiered approach as shown in figure 3.2 taken from [DWN+99]. The major components are:

1. An Internet based Client.

2. An Educational Search component which keeps a cache of Course and Resource metadata and provides a search facilities.

3. A Customer Profiles component, which keep track of user profiles by interacting with Trusted Education Systems and directly with users.

4. Agents that provide protocol conversion. The RDS can then access metadata from services which offer interfaces, which are neither CORBA, or Internet based.

[2]GAIA was one of the ACTS projects. It developed a generic architecture for Information availability [DWN+99].

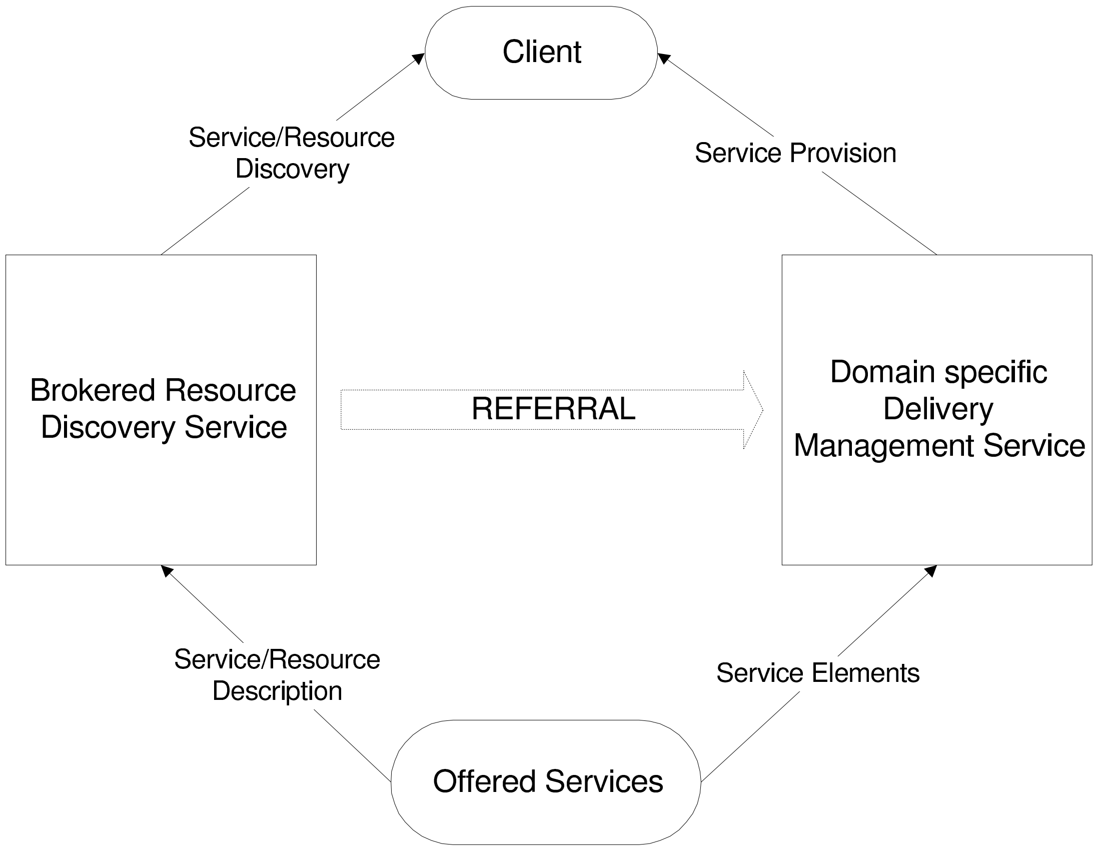

Client
Service/Resource Discovery
Service Provision
Brokered Resource Discovery Service
REFERRAL
Domain specific Delivery Management Service
Service/Resource Description
Service Elements
Offered Services

Figure 3.2: Resource Discovery Architecture

The Educational Search component outlined above provides the RDS search facility. It requires a cache of course and resource metadata in order to provide search functionality. This metadata must be created or found in an existing format. There is a need for a back end service to search the Internet for educational metadata and retrieve it for the Educational Service.

3.5 Role of the LOMAssistant

The LOMAssistant is the name of the tool created in this thesis. It is concerned with the metadata approach to educational resource identification. It provides a mechanism for the location and retrieval of the course and resource metadata that is required by the educational search cache in the GESTALT project. It performs this operation in two steps. Firstly it locates the necessary resources and retrieves them. Then it examines the resource to see if it has any existing metadata. If there is no meta data the LOMAssistant provides a way for the user to interactively create metadata for the resource. In this way, it creates the data necessary to populate the cache used by the RDS.

There were a number of factors that had to be taken into account by the LOMAssistant if it was to be used as part of the GESTALT project. An examination of the overall architecture of the RDS was necessary in order to analyse what restrictions or requirements GESTALT would impose upon the LOMAssistant.

Essentially, the role imposed on the LOMAssistant by GESTALT was to locate and retrieve courseware. This also involved an examination of the courseware once it was found, to ascertain if it had any associated metadata. If there was no metadata then it had to be created in accordance with the GESTALT metadata model. In the case where metadata was found, it may have needed to be modified so that it adhered to the model. This outlines the XML metadata support required by the LOMAssistant.

It was found that the LOMAssistant had to be web-based as the GESTALT project used current and emerging web-based technologies. This suggested that the LOMAssistant should run within a thin client browser and allow remote access. It also indicates that the LOMAssistant had to be distributed and run on many different platforms and environments.

Another important characteristic imposed on the LOMAssistant was the degree of usability. The LOMAssistant was to provide an interactive mechanism of creating metadata for educational courseware. In providing an interactive tool, the emphasis lies on usability and user-friendliness. It has to be easy to use and cater for a wide range of user experience ranging from proficient metadata creators to novices.

Although, all of the characteristics above had to be considered, one of the key requirements of any tool becoming part of a bigger project is that the tool can be integrated and interact proficiently with the entire system.

3.6 Summary

This chapter introduced the GESTALT project, and discussed how it provided a broader scope for the LOMAssistant. The background of GESTALT and the RDS were outlined in terms of motivation, aims and the architecture. Finally, the chapter ended with a discussion of how the LOMAssistant could become part of the greater system. The features required included XML metadata support, portability, usability and interactivity.

Chapter 4

Design

4.1 Introduction

This chapter discusses the design of the system. It outlines the requirements necessary to create a framework for the search and retrieval of educational courseware and the creation of metadata for these resources. The tool created throughout the course of this project was designed in two distinct phases. The first phase of the design addressed the issues of information retrieval on the Internet. It provided a solution to the problem of finding educational resources and courseware. The second half of the design deals with the creation of an editing environment and the associated services required in the generation of XML metadata.

There was a modularised approach adopted for the design of the tool. The system is composed of a number of components, which were created in distinct modules. This method was chosen so that each component could be developed and tested in separate units before starting the integration process.

4.2 Information Retrieval Requirements

This section examines the requirements of a system designed to perform information retrieval operations. There were three key areas that were considered in designing the retrieval system.

1. Search Technology
2. Result Presentation Format
3. Adaptability

It describes what technology was used to search for the information and why a particular one was chosen. This is followed by a discussion of how important the presentation format used for the results and adaptability is within the system.

4.2.1 Search Technology

Search Engine technology was chosen over Retrieval Agent technology for the reasons discussed in the Background chapter. In brief it is a well established technology, widely available, easy to adapt and it provides better metadata support. Various search engines could have been used, however due to time constraints only a modest investigation of the available search engines could be undertaken. There were a number of issues that had to be taken into consideration.

The cost of a fully functioning Internet search engine is very high. The budget of this project did not allow for a large-scale engine to be purchased. This led to an examination of freeware and the constraint that a free engine had to be used. The possibility of using an Internet wide search tool was eliminated as they all cost money. A small multi-site search engine was chosen for this reason. It operates by indexing a number of sites, which can be chosen by the user. If a prototype idea could be proved with a small engine then using a larger engine would just be a question of scalability . Other issues of less significance included the portability of the engine, whether the source code was freely available, the engine's support for XML and how easy it was to use.

4.2.2 Result Presentation Format

The presentation of the results retrieved from a search query was also found to be important, as the system needed to perform operations using these results. The way in which the results were presented could increase the user-friendliness and ease of use of the tool. The result format had to be adaptable so that it could be manipulated and changed into a desired format.

Upon observing a number of engines, including search engines, meta-engines and directories, it was noticed that they all returned the same general information in their results. This led to the definition of a search result type. A result describes a specific URL provided by some server, based on the information defined in the result type. A result type consists of:

Title

This is meant to capture the essence of the result contents.

URL

This represents the location of where the result can be found on the WWW.

Excerpt

This is textual data describing the contents of the URL.

Relevance Rating

It shows how relevant a particular result is to the original search query.

The result type consists of five tags, the relevance rating was usually indicated by the order in which the results were returned from the engine with the most relevant at the top, so it was ignored.

```
<!-- START -->
<!-- TITLE -->
<!-- URL -->
<!-- EXCERPT -->
<!-- END -->
```

The start of a result is indicated by `<!-- START -->`, while `<!-- END -->` means that the end of that particular result has been reached. `<!-- TITLE -->` signifies that the data following the keyword contains the title. The end of the title is reached when the beginning of another keyword is reached. The URL and excerpt keywords work in the same way as the title. The provision of a result type means that if another search engine is to be substituted for the one currently used, it will be compatible as long as the results are returned according to the same result type format.

4.2.3 Adaptability

The information retrieval system had to be adaptable. This meant that any search engine could be used within the system as long as the results retrieved from the engine could be changed into a certain desired format. With this in mind a typeI was designed for the results. This flexible approach was adopted so that the system was not dependent on a particular search engine. The searches could be adapted for any situation, all that had to be done was change the engine. The issue of actually finding the most relevant results to a query was left up to the search engine. If there was a new and brilliant free engine that indexed the web to come onto the market, then the tool should be adaptable enough to accept the results from this engine.

4.2.4 Information Retrieval Architecture

A search engine operates by receiving a search query, carrying out a search operation on the database and returning the results in some order of relevance. These results must be broken down and filtered into the individual results before they can be of any use. Each result is changed into a result type, which describes the URL.

The client browser must communicate the search query with the server and receive back the results. Two ways of carrying out the communication and presenting the results to the user were examined.

- HTML presentation over HTTP

  The user is presented with a HTML web page from which they are asked to make their search query. The query is input into a HTML form. As soon as the user submits the query, it is sent using HTTP to a Common Gateway Interface (CGI) script on the server, as shown in figure 4.1.

  The script takes the query string and puts it into the search engine. The server filters the results from the search engine, generates a HTML page to display the results and returns them to the browser over HTTP.

  The user can then select one of the presented results to retrieve. Selecting a result sends the result URL back to the server where another CGI-script is invoked to retrieve the file onto the server.

- Applet presentation over Sockets

  The user is presented with an Applet from which they are asked to make their search query. The applet is connected via a socket to the server. The applet takes in the query and sends it to the server over the socket.

  The server takes the query and puts it into the search engine. The server sends back the results to the applet. The applet has the task of filtering the search results. It then displays them to the user in a text format that it can display.

  The user then must select a result. The applet sends the URL of the result via the socket to the server. The server then retrieves the file.

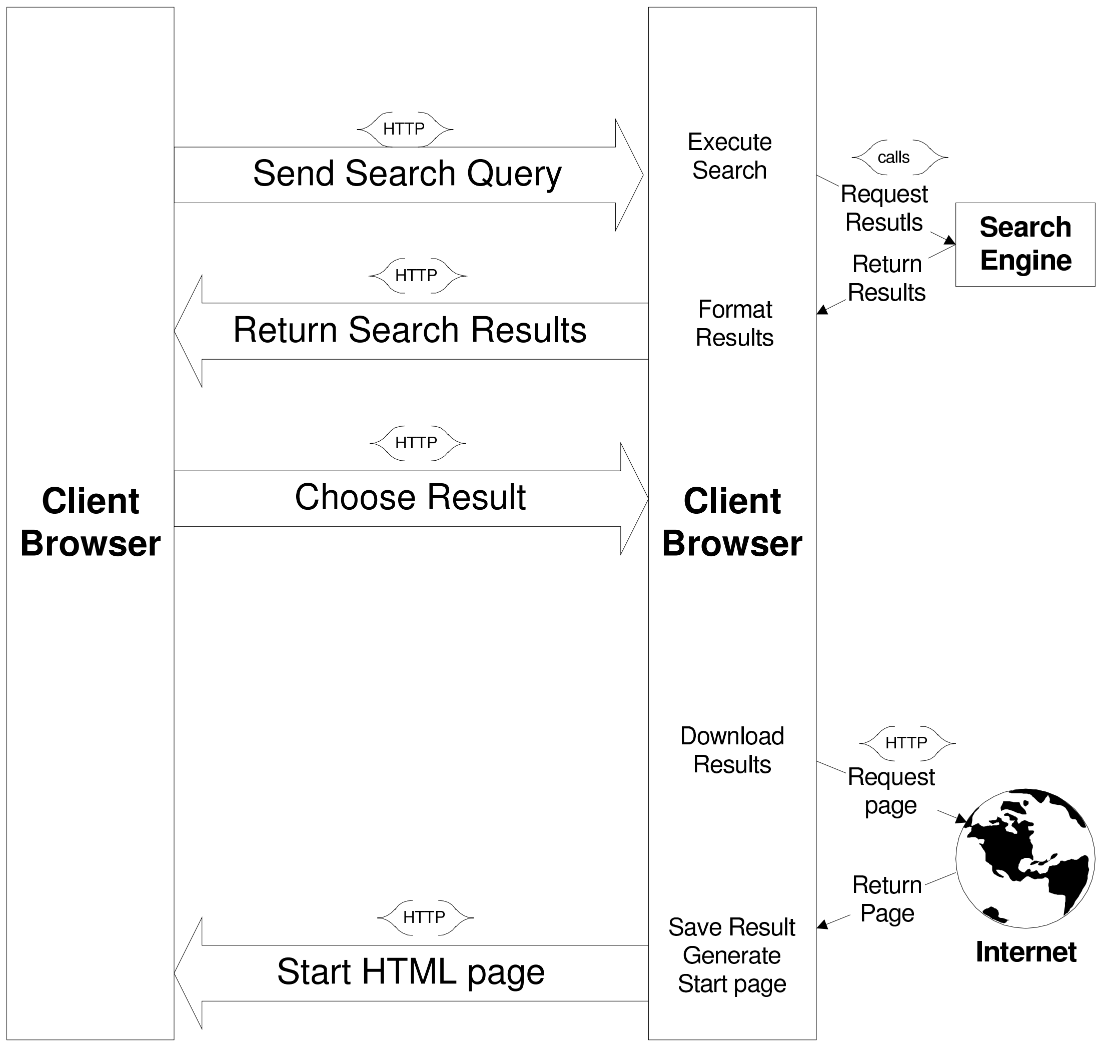


Figure 4.1: Information Retrieval Architecture

These two methods provide contrasting ways of retrieving the information. They both provide the same essential services at the server, execution of the search engine and retrieval of the file. Choosing between them is a matter of comparing their presentation formats and the communication mechanisms.

HTML displays text in a very clear and user friendly format. It is easy to navigate, easy to read and can be readily formatted. Applets on the other hand have to be specially configured to display text and it is difficult to format, may be unclear, may be difficult to navigate and it is not necessarily the most user friendly and intuitive manner of displaying text.

Both sockets and HTTP provide reliable communication mechanisms. The use of sockets would require the creation of the socket, maintenance and mechanisms of sending and receiving information to and from the socket. HTTP is a standard form of communication with well-developed forms that send information in a predefined way. It was decided to use HTTP for these reasons.

Once the educational courseware had been found and retrieved the work of this section was complete. The second half of this tool had to address the issues involved in the creation of metadata.

4.3 LOMAssistant

The LOMAssistant was designed to provide interactive editing tools in a user-friendly environment. It was envisaged to be different to an XML editor as it aimed to provide specific assistance for creating XML metadata based on the GEMSTONES model, which incorporates the LOM. It also provided editing facilities generally present in an XML editor.

It was decided to make the LOMAssistant an applet. This was advantageous for many reasons. It meant that the tool could be accessed through browser technology and run on a number of different platforms. It fitted in with the technologies used by the GESTALT project. It was compatible with the information retrieval half of the system and would complement it, along with easing the integration process. However, the use of an applet incurred a number of security restrictions, when attempting to read or write files to the local file system.

The design incorporates two parts, the back end operations and the graphical user interface (GUI). There were a number of requirements that had to be addressed, in the creation of this tool. These are listed below and outlined in more detail in the following sections.

1. File Access
2. Mode of Communication
3. DTD parser
4. DOM
5. Help Information
6. GUI usability

4.3.1 File Access

Firstly, there was the question of how to access information stored in files. One of the key requirements of this project was extensibility so as little information as possible was hard coded into the tool. This implied that the information used by the system had to be stored in a format that could be accessed easily. This lead to the development of a file intensive design.

The files could be accessed on the local system but since the tool was applet based, it could not read from or write directly to the file system. It had to access all files remotely so a mode of communication was needed to send the data to and from the server.

4.3.2 Mode of Communication

It was intended that the tool be accessed through browser technology, which is one of the most widely available modes of accessing remote servers. A mode of communication had to be chosen, both HTTP and CORBA were considered.

HTTP

The remote files are made available on a web server at a location that the client has permission to access. Figure 4.2 shows this process. The client makes a 'GET' request for a file, using HTTP as the mode of communication. The web server responds over HTTP by returning the file. It does this by opening

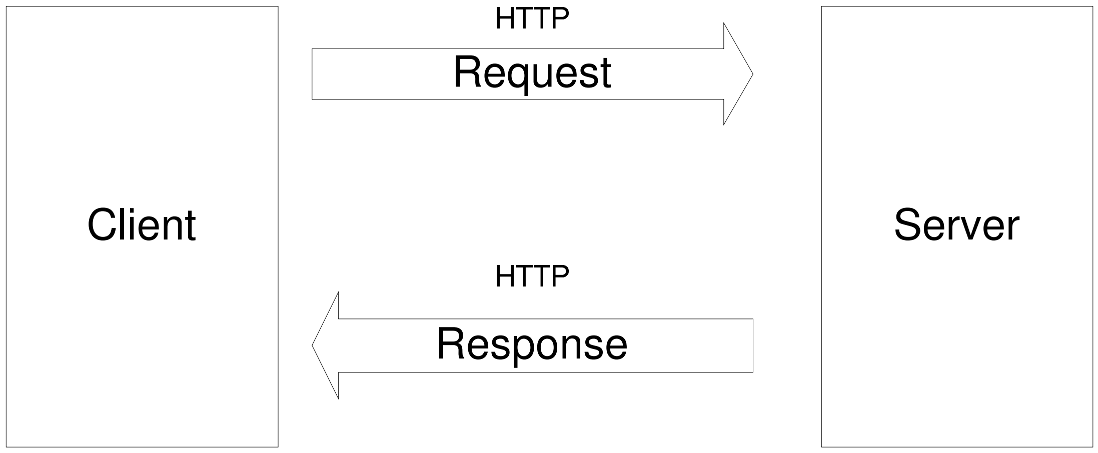


Figure 4.2: Communication over HTTP

a connection stream, making a get request of the file, receives the response and closes the connection.

CORBA

Another mode of file access considered was the use of a Common Object Request Broker Architecture (CORBA), as presented in figure 4.3. An Interface Definition Language (IDL) has to be written for the following reasons. It has to define what methods the client has permission to run, to define what input arguments the client must give and also to specify what format the response can be returned in. The client then makes a request for a file using one of the permissible methods and it receives the file as a response.

The sample IDL would look like this:

```
interface fileAccess {

/** A method requesting data stored on a remote system
                @param query - the query used to request the data
                @returns a string array containing the requested data
                (each entry in the array would represent a data source
                that matched the query)
        */
        string[] request_data( in string query );
```

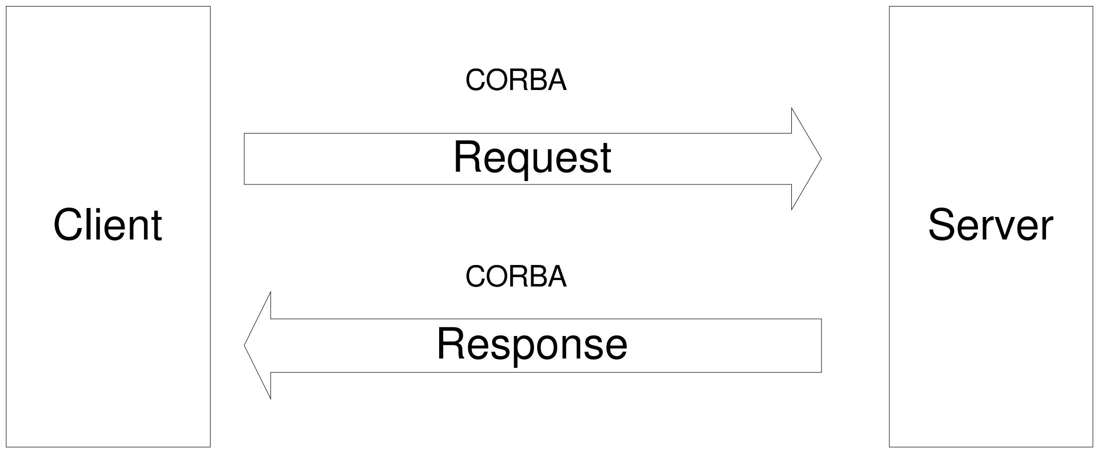


Figure 4.3: Communication using Corba

```
        /** A method requesting the data stored in a particular file
              @param filename - the name of the file
              @param data - the data to be written out to the file
              @returns a boolean indicating success or failure
        */
        boolean save_data( in string filename, in string data );
};
```

HTTP is the general mechanism used by web browsers to transfer information between the client and the server. It is simple, extensible and follows a standard format. It sends simple text-based messages. It does not support multi-purpose messages or complicated messages without encoding. It can give rise to certain security issues because any machine in the network can make a HTTP request to the web server once the address of the web server is known.

CORBA provides a structured way of accessing information. It is an international standard, widely accepted and used in place of HTTP. It is easily extensible but can be difficult to use. It often costs money to use and can be very expensive. There is a communication overhead and it is generally slower than HTTP.

It was arbitrarily decided to use HTTP as the mode of communication although either methods could have been used.

4.3.3 DTD Parser

Another key issue was how to parse the GEMSTONES DTD. The DTD provides the fundamental metadata tree upon which the LOMAssistant tool is built B.

It contains:

- The names of all entities, attributes and elements
- The default values for elements and attributes
- All choice lists - or available values from which the user must choose
- Indication of whether the element or attribute is required, fixed or implied
- Indication of whether the element or attribute is optional, can occur once or multiple times.

The DTD must be read into the tool, as due to the extensibility requirement, it could not be hard coded. This required the use of a DTD parser to read it and create a structure that could be accessed at runtime. Since the DTD was based on a tree structure a DTD parser based on a tree structure was also used.

There is an IBM DTD [IBMb] parser freely available. However, it is only a beta copy and it was found to be over-complicated and difficult to use. After fruitless efforts and failed attempts to get to grips with their parser, it was decided to write a simple yet extensible DTD parser. The parser needed to store information about entities, elements and attributes that form part of an attribute list (called an attlist in the DTD).
The entity data consisted of:

- entity name

  The unique name of the entity.
- entity arguments

  One or more elements that are contained in the entity.

The element data consisted of:

- name

  The name of the element, this is unique amongst the elements.

- value

  There are four types of accepted element content specifications.

  - A list of other elements that can be specified as content
  - A mixture of both data and elements
  - The keyword 'EMPTY' that specifies the element will contain no content
  - The keyword 'ANY' that specifies that the element will allow any type of content (data) or element mark-up information.

The attribute list data consists of:

- name

  The name used to identify the attlist in the DTD and reference it in a document. This is also the name of the element that it is associated with.

- one or more attributes

  A list of all the attributes that can be used with a given element. The attribute contains three distinct parts as described below.

The attribute data consists of:

- name

  The name used to identify the attribute in the DTD and reference it in a document.

- type

  The type that identifies the attribute as a string, tokenised or enumerated attribute.

- value

  A list in the specification of all the possible values an attribute can take, in a document, this is the specific value assigned to the attribute by the document developer.

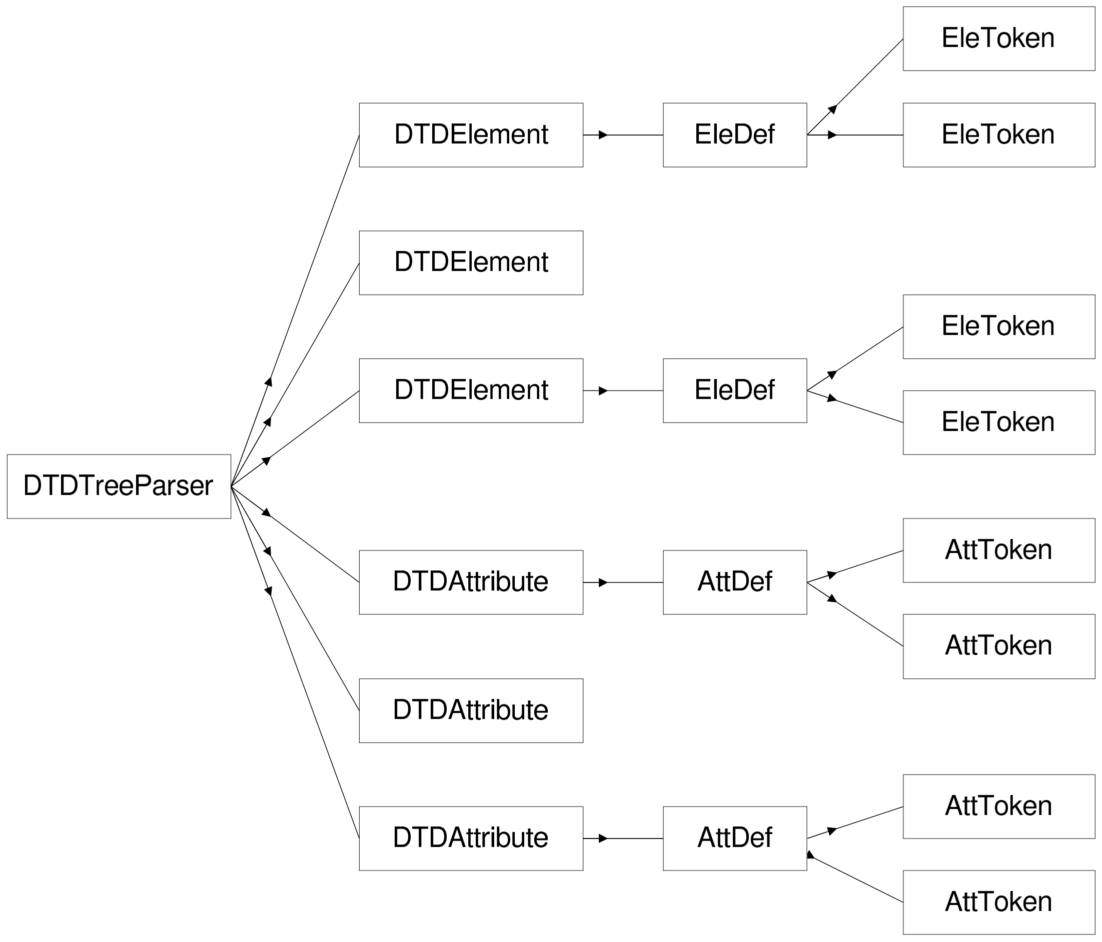


Figure 4.4: Parser Architecture

The element and entity were found to contain similar data types, with the entity seen as a simple case element. This led to the use of a general design for both of these types in the DTD parser. The attribute list was found to have a different format based around the attribute. The DTD parser had to treat the attributes and elements separately. It took each attribute or element/entity and broke it down into a number of tokens.

Parser Architecture

Figure 4.4 shows the architecture of the parser. The DTDTree contains an infinite number of DTDAttributes and DTDElements, the DTDElements also encompasses the entities. Each DTDElement is broken down into an Element definition, which contains information about the element and a number of element tokens. The DTDAttribute is similarly broken down into an Attribute definition and a number of Attribute tokens.

The parser takes in the DTD and sequentially traverses it until it has created an internal structure of DTDElements and DTDAttributes. It also provides ways of accessing these elements and retrieving information about them. It was designed to be a general parser so that it could deal with modifications or changes to the DTD.

4.3.4 DOM

The W3C Document Object Model (DOM) provides tree-based support for XML. This makes it an ideal candidate for storing the XML, once it is created from the DTD. There are a number of well-established vendors such as SUN and IBM [IBMb], who have written frameworks and freely available interfaces and classes for parsing XML and accessing the DOM model. These frameworks promote the notion of reuse and also allow data to be easily read into the DOM and taken from it.

SAX, the simple API for XML is another alternative model that could have been used as a way of storing XML. It is an event driven model and it allows users to access XML documents by a sequence of events.

DOM was chosen over SAX for use in this tool because DOM is based on a tree structure. It provides more powerful functionality, it that reads in an XML document, parses it and creates a model that can be manipulated. With SAX the parser does not do much, it reads in the XML document and fires a number of events which the user much catch and deal with. These events have to be interpreted and dealt with by the user. In SAX a model has to be written to hold all the information obtained in the XML document. This is another reason why SAX was not chosen, as the purpose of using the DOM was to hold the information read in from the XML document.

The DOM is based on a tree structure. There are three main interfaces of interest in this model that are required by an XML document. The Element interface represents an element in the document, the AttributeList interface represents an element's attribute specifications and the Document interface represents the entire XML document. It is the root of the document tree and provides the primary access to the documents data. A number of helper interfaces are also required.

The DTD parser designed in section 4.3.3 maps directly onto the DOM framework. A DOM Document can be created where the DTDElement is mapped onto a DOM Element and the DTDAttribute is mapped to the DOM AttributeList.

4.3.5 Help Information

Usability and user-friendliness are essential requirements of the LOMAssistant. This implies that the tool must incorporate a large amount of help information. The tool aims to provide more assistance to the user than just an ordinary off the shelf XML editor. The DTD that the tool is based on can be extended at any given time and so the help information must also be extensible.

The help information consists of:

- Tagname

  A unique name used to distinguish it from all other tagnames.

- Definition

  The official GEMSTONES definition from the project deliverable [DWN$^+$99].

- Description

  A brief description of the definition

- Necessary

  States whether that tag is required or optional.

- Example

  An example of the tag as it is used in an XML document.

- Notes

  Any other information of significance.

It reduces the amount of information hard coded into the tool and makes it more dynamic.

It was decided to use XML as the mechanism for storing this information. It reduces the amount of information hard coded into the tool and makes it more dynamic. This meant that an existing XML Parser could be used to extract the XML from the file. It also meant that the help file would be extensible as a new help instance can be added at any time. Each instance of help related to one particular tag was called a lomelement. An example of a lomelement is outlined below, as it is defined in the XML file. Appendix A presents a small version of the LOMAssistant help file.

```
<lomelement>
        <name> Identifier </name>
        <definition> A unique label for the resource.</definition>
        <description> This is the label used to uniquely identify the
resource from all other resources.  It is usually a Uniform
Resource Locator (URL) on the Internet.
        </description>
        <necessary> Mandatory </necessary>
        <example> http://143.225.230.104/IPER_CPP/index.html </example>
        <notes> This element can be transparent to the metadata creator.
</notes>
 </lomelement>
```

A DTD was defined to specify the format of the help file. The sample DTD is presented below.

```
<!ELEMENT lom (lomelement)+>
<!ELEMENT lomelement
(name,definition,description?,necessary?,example?,notes?)>
<!ELEMENT name (\#PCDATA)>
<!ELEMENT definition (\#PCDATA)>
<!ELEMENT description (\#PCDATA)>
<!ELEMENT necessary (\#PCDATA)>
<!ELEMENT example (\#PCDATA)>
<!ELEMENT notes (\#PCDATA)>
```

The name of the document root is lom. It contains one or more lomelements, indicated by '+'. Each lomelement can contain a name, definition, description, necessary, example and notes. The name and definition are required fields, as they have no associated special characters. All other information inside a lomelement is specified as optional, this is indicated by the use of '?'. The keyword '#PCDATA' means that there is a string expected as the value of this tag.

4.3.6 GUI

The Graphical User Interface (GUI) was designed to be user friendly. It should provide the user with an intuitive method of carrying out actions. It incorporated a high degree of user help as it aimed to provide more than just editing capabilities, it attempted to provide assistance in the form of helpful descriptions and examples. The interface captures actions taken by the user and packages them into the appropriate events that can be dealt with by the tool.

There were certain restrictions imposed on the GUI because it was an applet. The applet had to be of such a size that it would fit inside a browser window. The attention of the user would be lost if he had to continuously scroll up and down out of the browser window. Another feature of applets is that they are action and event driven. This meant that the GUI had also to be event driven. Applets are generally slow to download but once this is done, they are quick to process actions.

The Java Swing classes were used in the creation of the GUI since they provide a graphical user-interface toolkit that simplifies the development of windowing components. Once a GUI is written using swing, it can be run anywhere as long as they have a sufficiently up to data browser that supports them. The Java Abstract Windowing Toolkit (AWT) could have been used but the Swing toolkit was more up to date and provided better features, such as the tree component.

It was decided to create a tree-based GUI. This was due to the use of the tree based DOM and DTD. The traversal of the tree is an interactive process and it generates events that could be harnessed to show help information whenever possible. This would increase the assistance provided by the tool. The focus would be on the tree, as it would provide the main source of events.

The GUI was designed to be highly Object Oriented. It was composed of a number of display panels. Each panel was a separate object, so panels could be swapped in and out depending on the tool's current requirements. The main panels required by the display pane were the text panels, tree panel and button panel.

- Text Panel

  The Text panel contained a Swing component for displaying text on screen. Two text panels were used on the display. One displayed the HTML/XML source code of the courseware file that the user had searched and retrieved. The other displayed the XML metadata that was generated for the file. These were put at the top of the display as it was based on the source code that the user was creating the metadata.

- Tree Panel

  The Tree panel was placed in the centre of the browser window, as it was the main focus of the tool. It was composed of two parts. On the left-hand side, the GUI tree was displayed and on the right hand side was a panel that changed depending on where in the tree the user selected.

  The GUI tree was created according to the structure of the DTD. It consisted of branch nodes and leaf nodes. The branch nodes held the name of the element. The leaf nodes contained either an attribute indicator or the keyword '#PCDATA' showing that the user must input element data. A branch also signified that there were more tags underneath the branch, while a leaf indicated that some data was required at this level. Whenever the user clicked on a leaf, the panel beside the tree changed to prompt the user for input. At a later stage, if the user re-visited the leaf, it would display the information that the user had previously entered. This meant that as long as the user traversed the tree, and entered information whenever prompted, there would be no need to try and edit the XML generated in the textbox. This reduced the XML pre-requisite knowledge required by a user to use the tool, even a novice could use it. All data could be edited directly from the tree. The use of a tree also ensured that any XML generated adhered to the structure defined in the DTD and the user could only choose tags that were applicable at that level.

  The panel beside the tree presented different information depending on whether a branch or leaf was traversed. It displayed help information about the tag name if a branch was selected. It showed a panel prompting the user for information if a leaf was clicked.

- Button Panel

The button panel contained control buttons for the applet. These included buttons to save, help, quit and view a page. The Save button saved the newly generated XML to the server. This was done by sending the data over HTTP to a CGI script where it was written out to a file on the server. The Help button displayed help information about the current node selected in the tree. The Quit button allowed the user to quit the application and saved their work in doing so. The View Page button enabled the user to view the html page that they were marking up in a pop up browser window. The purpose of the pop up window was so that the user could view the web page without having to leave the editing environment. The button panel was placed at the bottom of the window in general one works from top to bottom of the page.

4.4 Resource Discovery Service

The information Retrieval System and the LOMAssistant can easily be integrated into the GESTALT Resource Discovery Service as they are all separate components. Figure 4.5 shows the different components that make up the system and the flow of information between them. The information retrieval system finds educational resources on the Internet and returns them to the LOMAssistant which creates metadata for the resources in accordance with the GEMSTONES metadata model. The LOMAssistant transfers the metadata to the resource discovery broker which uses the data to populate its database.

Although the broker system and the tool designed throughout this thesis are separate entities, the integration process of the two systems should be straightforward. Metadata is the link between the two systems. Ones creates it, while the other requires it for its operation.

4.5 Summary

In this chapter the analysis, requirements and design of the system were discussed. From the analysis carried out in the previous chapters, certain requirements presented themselves. Using these requirements, the design decisions and the objects needed were introduced, giving general details on the flow of control and how they

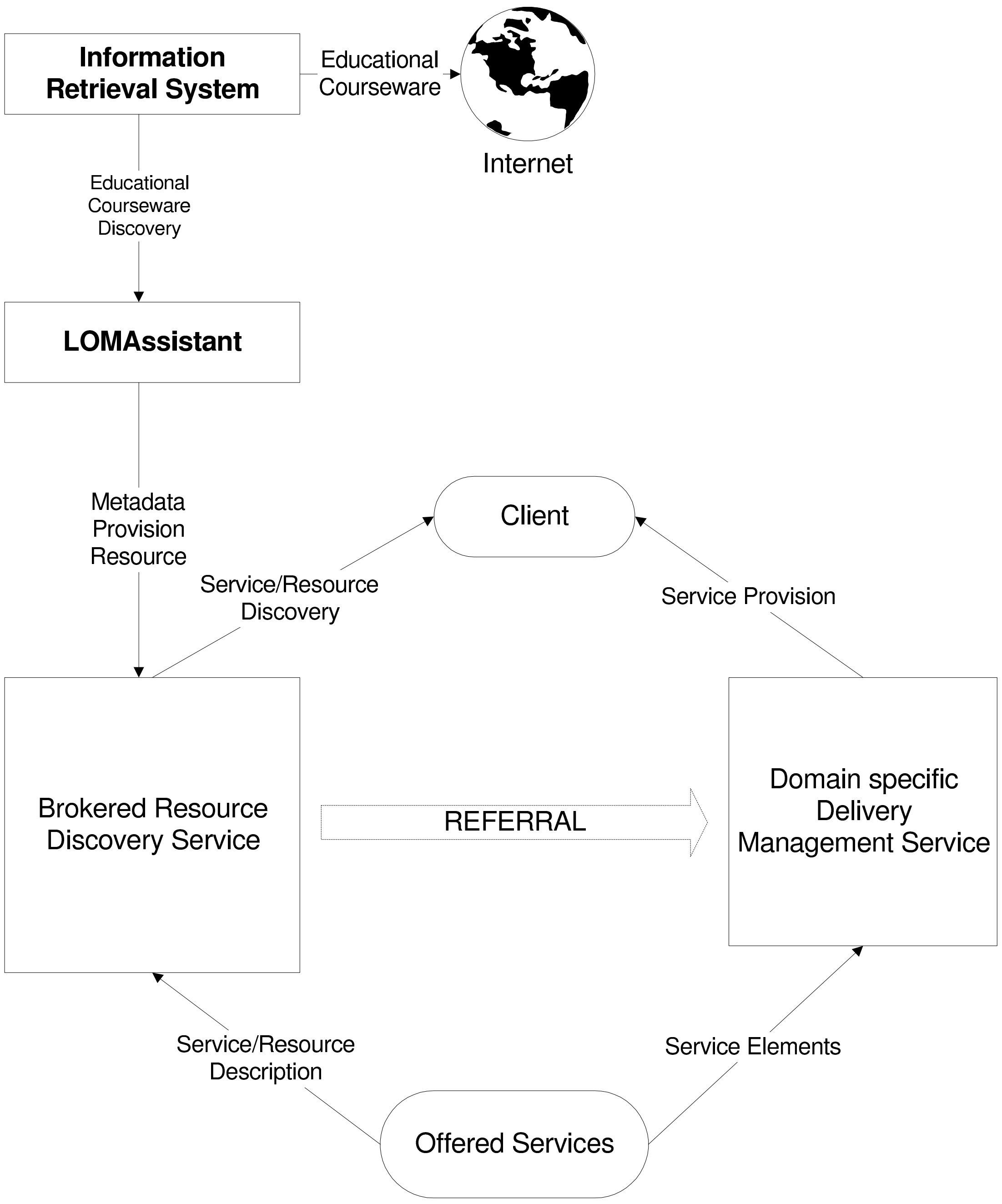

Information Retrieval System
Educational Courseware
Internet
Educational Courseware Discovery
LOMAssistant
Metadata Provision Resource
Client
Service/Resource Discovery
Service Provision
Brokered Resource Discovery Service
REFERRAL
Domain specific Delivery Management Service
Service/Resource Description
Service Elements
Offered Services

Figure 4.5: The tool as a part of the Resource Discovery Service

should interact.

Chapter 5

Implementation

5.1 Introduction

Chapter 4 presented the design of the system. This Chapter outlines the tools used in the implementation of this design including the Apache Web server, ht Dig search engine, XML Parser, DTD Parser, the Java Swing package and the DOM. The different classes used in the system are described and these were implemented in two separate parts.

5.2 Implementation of the Information Retrieval System

There are a number of components of interest in this section. Firstly, the Web Server and the Search Engine used will be described, followed by a brief outline of the communication mechanisms and an overview of the classes operating in this system.

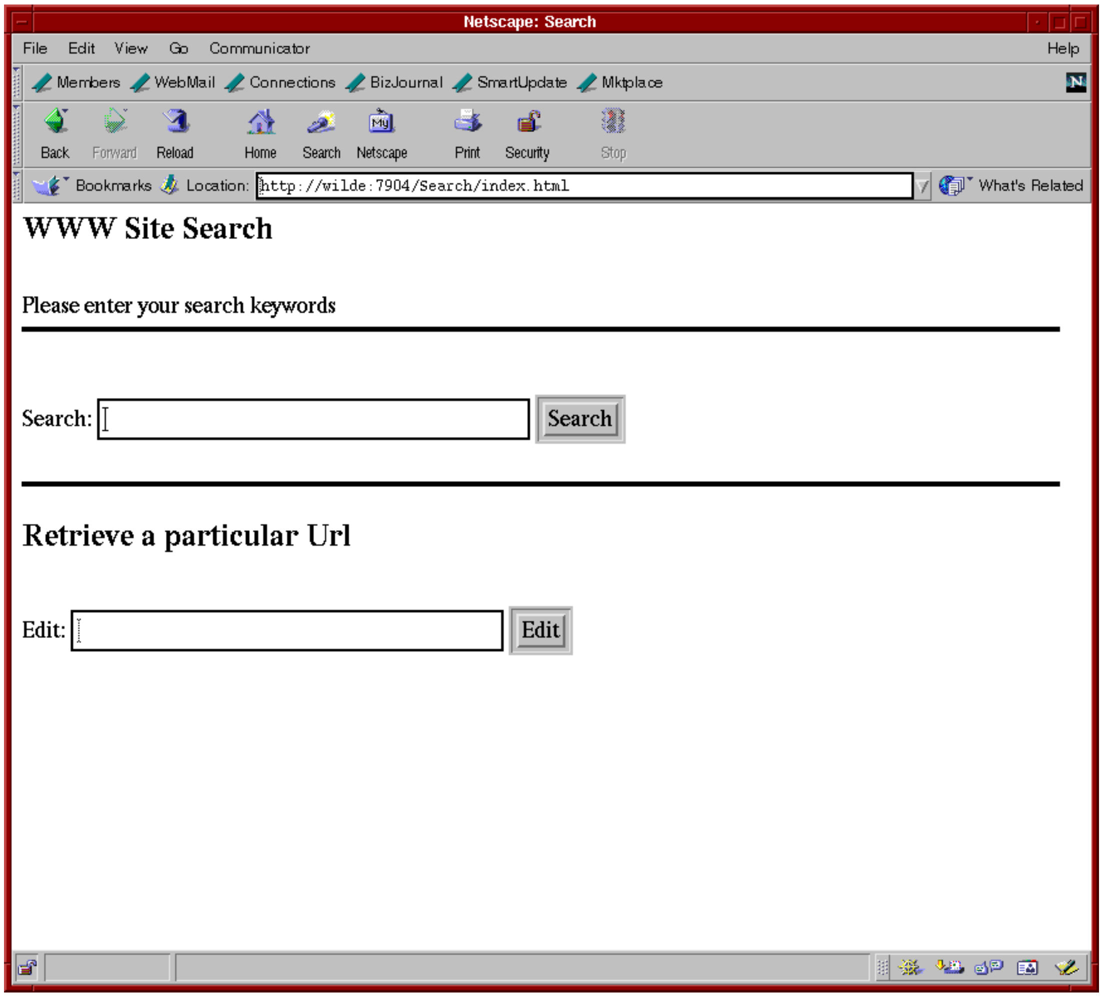
5.2.1 Web Server

The Web Server used for the implementation of this project was Apache Web Server. The Apache Software Foundation is a collaborative software development group aimed at creating robust, commercial-grade, feature rich and freely-available software. It provides access to a free HTTP Web Server implementation, which is available with source code. [APH]

Apache web server is used in this project to manage the site from where the search engine operates and the LOMAssistant applet tool is downloaded. It also manages access to the file system where the newly created metadata courseware files, the GEMSTONES DTD and the XML help file are stored. These files are all required for the implementation of the system.

5.2.2 Search Engine

ht://Dig [HTD] was the name of the Search Engine chosen for use in this project. The main reason it was picked was because it was free, available, easy to setup and accompanied by downloadable source code. ht://Dig is a small site search engine that indexes a given a list of URLs.


Netscape: Search
File Edit View Go Communicator Help
Members WebMail Connections BizJournal SmartUpdate Mktplace
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://wilde:7904/Search/index.html
What's Related

WWW Site Search

Please enter your search keywords

Search: [Search]

Retrieve a particular Url

Edit: [Edit]

Figure 5.1: The initial search query page

The system provides three major tasks, digging, merging and searching. Digging is the first step towards creating a search database. A spider is sent out to the pre-defined sites where it follows all hyperlinks (within these sites) that it comes across. It examines each page and extracts all the unique words. It then stores these words in a database along with information about the related URLs in a separate database. The merging process takes the databases already created and merges them into several other databases that the search engine can use. Searching is when the users actually use the information gathered by the digging and merging processes.

A CGI web form is used to submit a query to the search engine. This web page is illustrated in figure 5.1

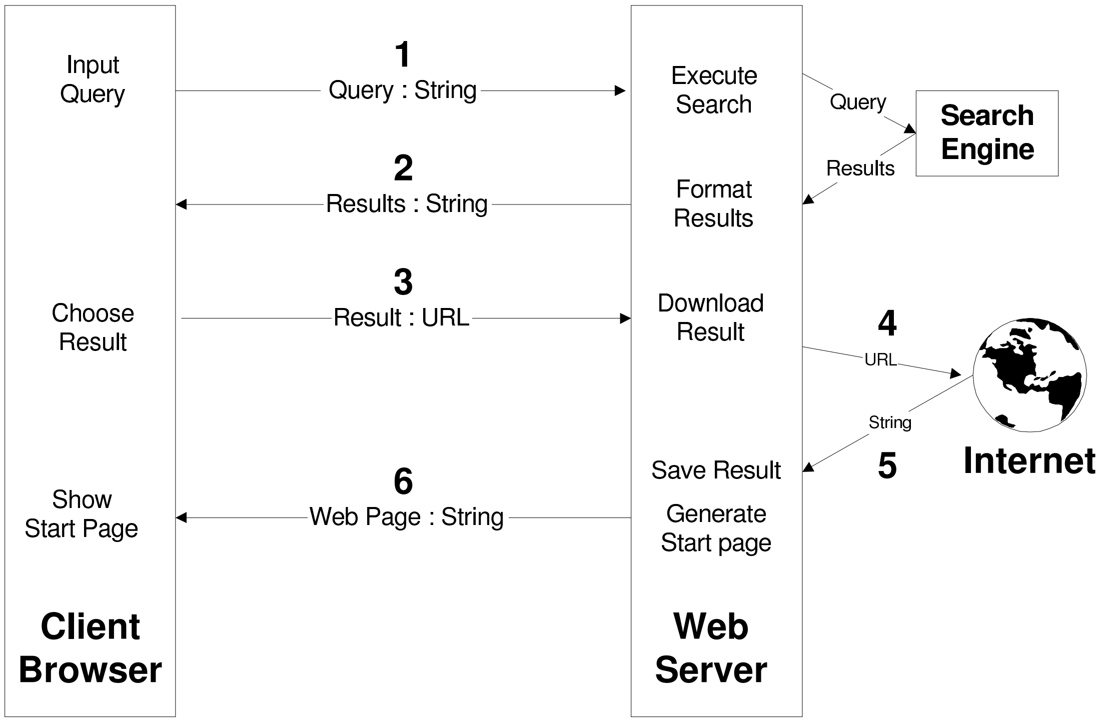


Figure 5.2: Communication messages of a search query

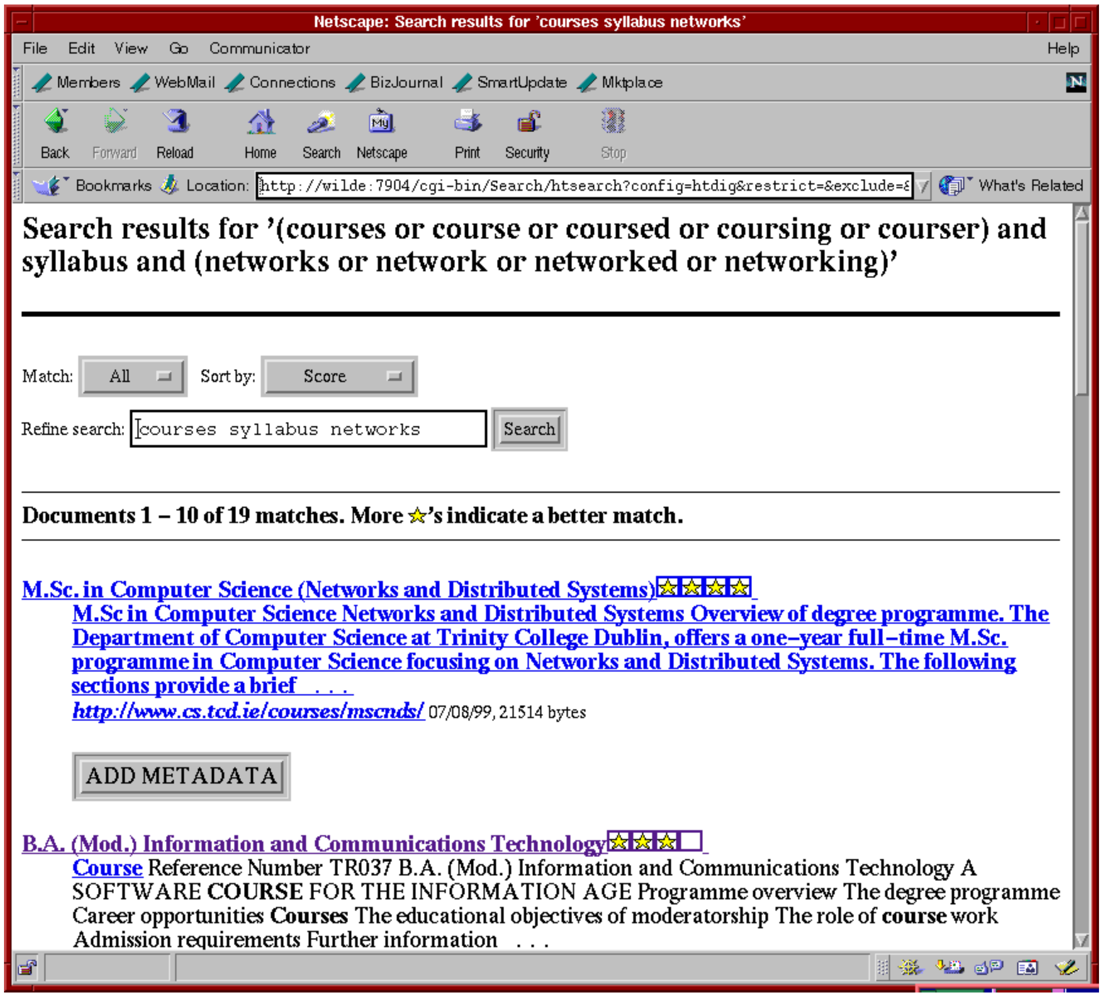
5.2.3 Communication

The client browser must communicate with the web server so that data can be passed over and back within the system. In section 4.3.2 the possible modes of communication were outlined. It was arbitrarily decided to use HTTP although CORBA could equally have been used. There are two ways this communication can occur. The client can invoke a search query and choose a result which then must be downloaded, or the client can enter a url directly to be downloaded.

Search Query

The user is presented with a web page asking them to enter a search query. This web page was presented in figure 5.1. Figure 5.2 shows the messages that have to be sent between the client and the server when a search query is issued.

1. The client sends the query string that contains the information for which they are searching. The query is sent to the program search.CGI over HTTP.


Netscape: Search results for 'courses syllabus networks'
File Edit View Go Communicator Help
Members WebMail Connections BizJournal SmartUpdate Mktplace
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: What's Related
Match: All
Sort by: Score
Refine search:
Search
ADD METADATA

Location: http://wilde:7904/cgi-bin/Search/htsearch?config=htdig&restrict=&exclude=&

Search results for '(courses or course or coursed or coursing or courser) and syllabus and (networks or network or networked or networking)'

Match: All Sort by: Score

Refine search: courses syllabus networks

Documents 1 – 10 of 19 matches. More ☆'s indicate a better match.

M.Sc. in Computer Science (Networks and Distributed Systems)☆☆☆☆
M.Sc in Computer Science Networks and Distributed Systems Overview of degree programme. The Department of Computer Science at Trinity College Dublin, offers a one–year full–time M.Sc. programme in Computer Science focusing on Networks and Distributed Systems. The following sections provide a brief . . .
http://www.cs.tcd.ie/courses/mscnds/ 07/08/99, 21514 bytes

ADD METADATA

B.A. (Mod.) Information and Communications Technology☆☆☆
Course Reference Number TR037 B.A. (Mod.) Information and Communications Technology A SOFTWARE COURSE FOR THE INFORMATION AGE Programme overview The degree programme Career opportunities Courses The educational objectives of moderatorship The role of course work Admission requirements Further information . . .

Figure 5.3: A screenshot of the results from a search query

After receiving the first client request (message 1) the web server must execute a search of ht://Dig's database. Then the results have to be filtered so that they appear with the button that allows a user to create metadata for a particular result. The results are presented in a web page as shown in the screenshot in figure 5.3.

2. The server responds with the search results of this query.

3. The client chooses one of the possible results and sends it to the server. What actually happens here is that the URL of the clients chosen result is sent to the server. It is sent to the program markup.CGI which calls the class MarkUp.

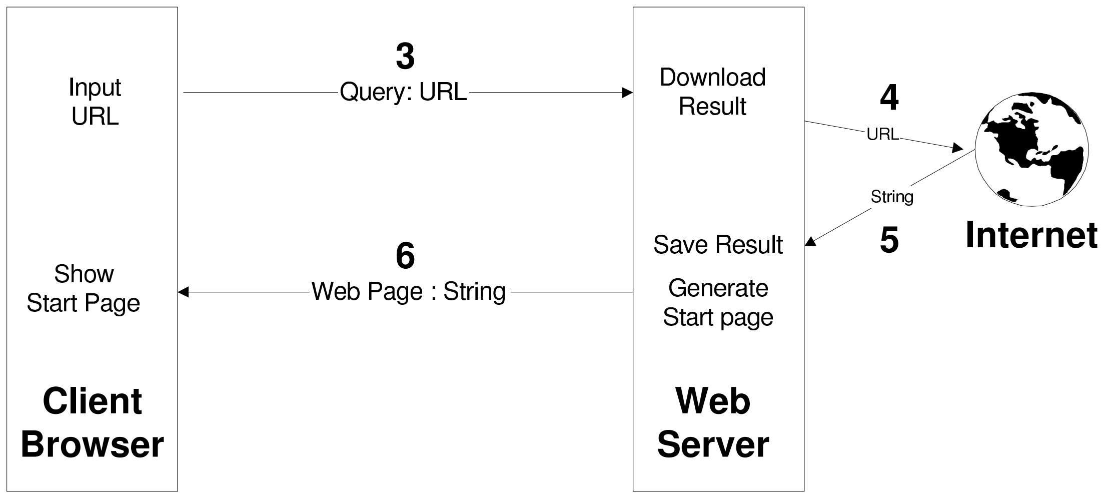


Figure 5.4: Communication messages without a search query

When the server receives the second client request (message 3) it passes it to MarkUp which has to request the URL of that result from the host of that URL on the Internet. The class Load performs this operation. These classes will be detailed further in section 5.2.4.

4. The server requests the URL from a remote machine.

5. The remote machines responds with the contents of the URL.

   The web server receives the contents of the URL from the remote host as a response and this must be saved to a file on the Web Server. The class SimpleFileIO performs this operation, and is detailed further in section 5.2.4.

6. The server responds to the client, with a HTML page from which the client can open up the metadata creation tool. The signifigance of this page is that it contains the name of the file the user has retrieved as a parameter.

Direct URL

In this case, the user does not perform a search but enters a URL directly. The messages sent between the client and the server are essentially the same except for the stages 1 and 2 are cut out. This is illustrated in figure 5.4. The client inputs a URL directly, this is sent to the CGI script Markup.CGI which calls the

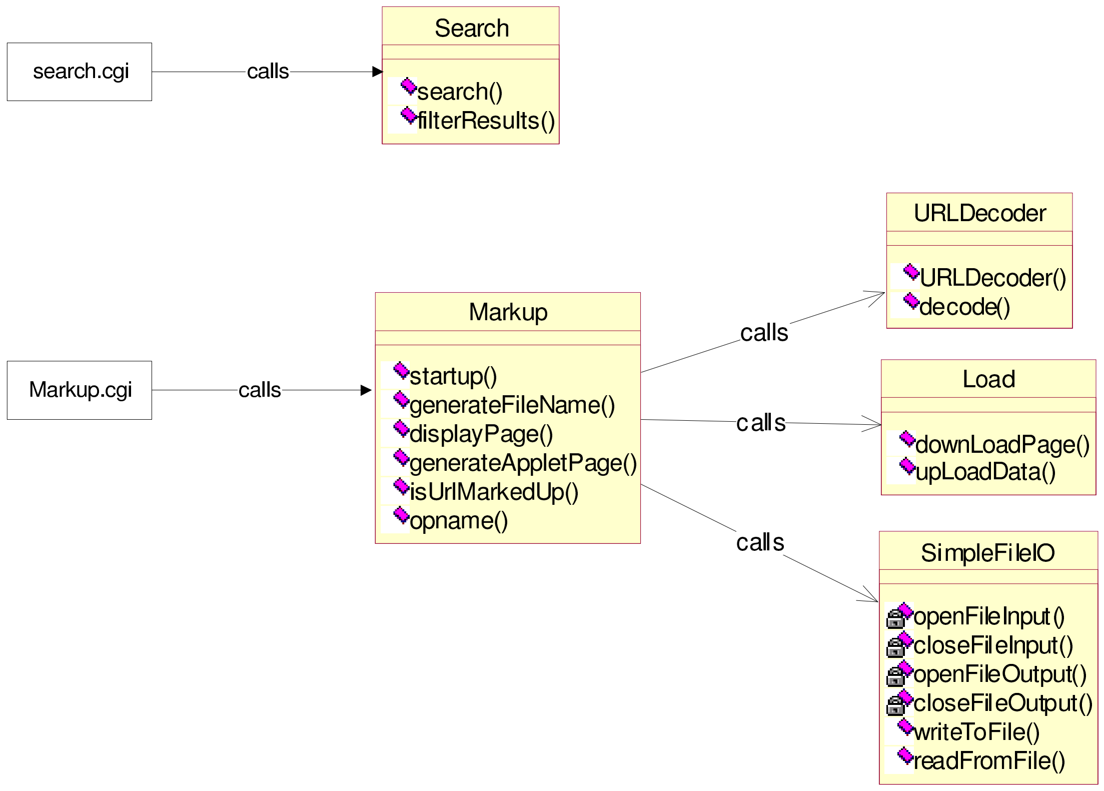


Figure 5.5: Information Retrieval Class Diagram

class MarkUp on the server. It performs the same operations here as it does in the previous section.

5.2.4 Class Overview

This section gives a quick overview of all the classes used in the information retrieval system. The previous section outlined when each of them was invoked. Figure 5.5 shows the interaction of these classes.

MarkUp

This class takes in the result URL and decodes it by calling the URLDecoder class. It checks to see if this file is already on the Web Server and if it is not it invokes the Load class to download the contents of the URL onto the Web Server. It generates two web pages. The first page contains a button to allow the user to call a CGI script to open a new window in which to open the metadata tool.

The second page contains the applet tags for actually opening the tool, with the users filename as a parameter.

URLDecoder

The purpose of this class is to decode any string that has been distorted by passing it over HTTP. When a query string is passed over HTTP, all spaces are removed and certain characters are converted into their hexadecimal values. This class converts the hexadecimal values back into their correct ASCII values and changes all '+'s back into spaces.

Load

This class provides methods for downloading a file from a remote site and for uploading data to a file on a remote site. The download method requires the URL of the required file and it uses the HTTP 'GET' method to request the file contents. It calls a method in the SimpleFileIO class to save the data to a file. The upload method uses a HTTP 'POST' request to send the date to the web server. It requires the data to be saved, the name of the CGI file which calls a java program to save the data and the URL of where this CGI file is to be found.

SaveData

This class takes in data and writes it out to a filename using the SimpleFileIO class. It was written as a separate class because it was invoked by a CGI script for the sole purpose of saving data.

SimpleFileIO

This class is used to read from files and write to them. It takes care of the file opening and closing and it manages the data streams required for these operations. The write method is synchronised so that it can only be called by one process at a time. This is to ensure that the correct information is written to the correct file, and it is not overwritten accidently.

5.3 Implementation of the LOMAssistant

The implementation of the LOMAssistant was divided into a number of parts. These are outlined in the following sections.

5.3.1 XML Parser

An XML Parser is required to process the metadata so that it can be used by an application. A parser reads in XML metadata, creates an internal structure and provides API's so that the data can be accessed. There are a number of publicly available parsers including Lark, Larval, MSXML, XP and XML4J [MTU99] [IBMc] [SN99]. The parser XML4J created by IBM was chosen for the implementation of this project as it was found to provide the most robust, efficient and easy to use parser. It is used in the application by two classes. The class LomHelp which will be described further in section 5.3.3 uses the parser to read in the help data which is stored in XML format. The class CreateDom uses the parser to read in XML metadata which has already been created by the LOMAssistant tool and needs to be edited.

5.3.2 DTD Parser

The DTD parser was composed of the classes shown in figure 5.6.

XMLDTDTree

This is the main class of the DTD Parser. It breaks down the file into instances of elements, attributes and entities and creates the appropriate classes to deal with them. It performs this operation by looking for an open tag '<' and the matching closing tag '>', and extracting the data in between. It contains two vectors, one which stores the attributes and another which stores the elements and entities. Vectors were used as the mechanism for storing the data as they can grow dynamically and they are not bounded by a pre-defined upper limit. This means that if the DTD is extended, this parser can still accomadate it.

Within a DTD there is a special syntax used to define an element, entity and attribute. Here is a example of how an entity and an element is defined in the DTD. Note that there are many other forms, this is just to illustrate the data that a DTD parser must parse.

Element

```
<!ELEMENT GEMSTONE  (General,LifeCycle,Metametadata,Technical,
Educational*, RightsManagement+,Relation*,Annotation*,
Assessment?,QoS?) >
```

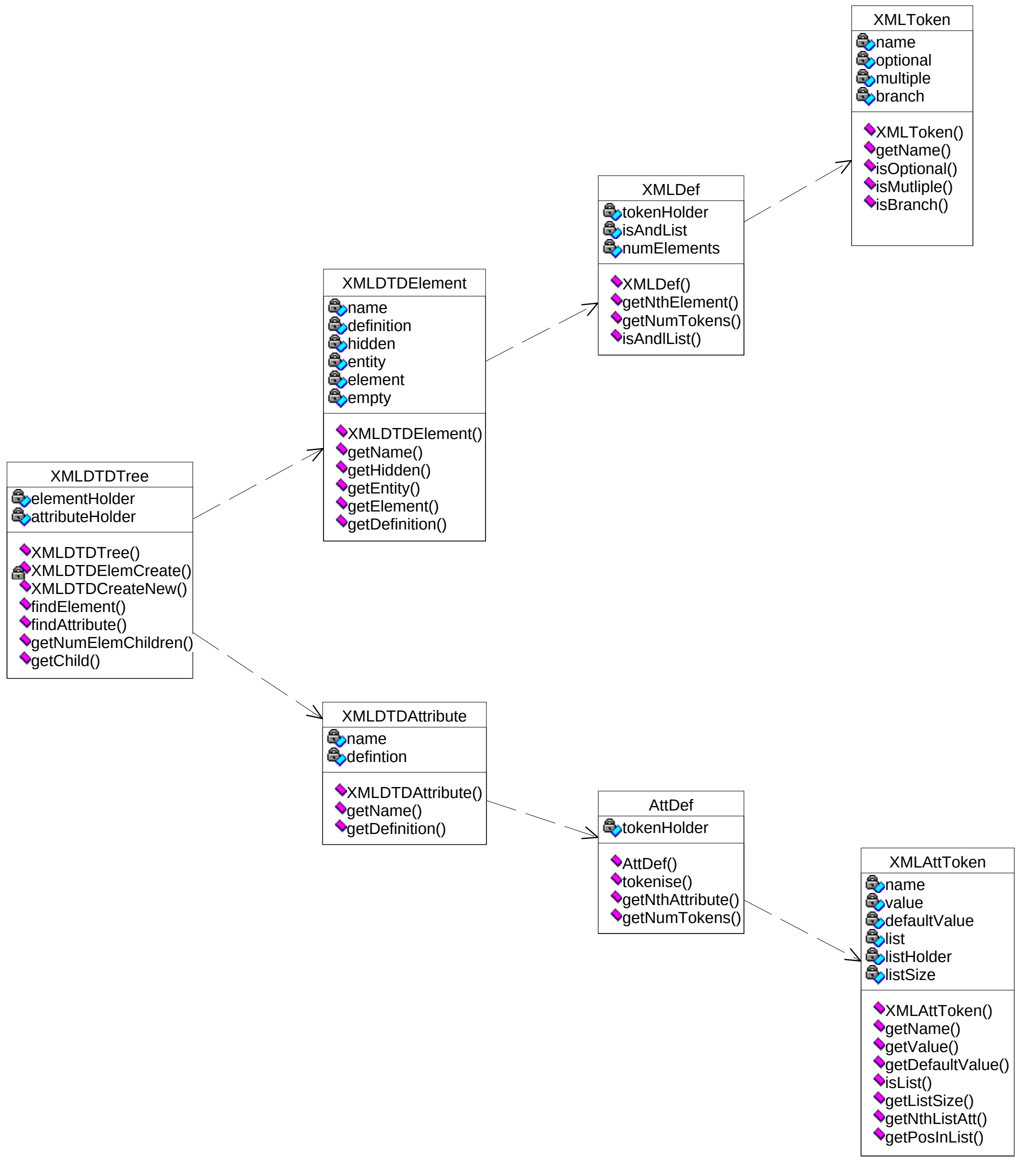


Figure 5.6: DTD Parser Class Diagram

symbol	meaning
*	element appears one or many times
+	element only occurs once
?	element is optional

Table 5.1: Element Symbols

The !ELEMENT indicates that it is an element. The next word is the name of the element and everything within the brackets after the name are arguments of the element. Each of the arguments has it's own element structure. The symbols '*', '+' and '?' mean specific things and this information must be also be recorded. Table 5.1 gives an explanation for each of the symbols.

Entity

```
<!ENTITY % SemanticScheme  "(TaxonPath,Description*,Keywords*)" >
```

The !ENTITY shows that it is an entity, this is followed by the name and the arguments are stored inside quotation marks. An entity is essentially an alias, it is used as a shortcut instead of typing the list of arguments repeatedly. As can be seen it has a very similar structure to the element so the two are dealt with in the same way.

XMLDTDElement

The XMLDTDElement class stores information about an element and provides methods for accessing this information. It is created by the XMLDTDTree. It contains the element or entity name, indication that it is an element not an entity, status information and a reference to another class XMLDef which breaks down the element or entity further.

XMLDef

This class takes the list of arguments to an element and breaks each argument down into a token. The token is contained in a XMLToken class. The class stores an array of these tokens and ways to access them.

XMLToken

This class is the bottom level of the parser as it contains the base part of the Element. It stores the token name and whether the token is optional, multiple or it can occur once only. It also provides methods for accessing this data.

The Attribute syntax within the DTD is shown below. It is processed by the DTD parser in much the same way the Element and Entities are processed.

Attribute

```
<!ATTLIST Language
               xml:lang  NMTOKEN   "en" >
<!ATTLIST OnLine
               value (Yes | No)  "Yes" >
<!ATTLIST Date
               day  CDATA    #REQUIRED
               month  CDATA  #REQUIRED
               year  CDATA   #REQUIRED  >
```

The Attribute can have many different structures. Each attribute is stored in an attribute list. The !ATTLIST is used to identify this type. Each attribute list has a name and then a number of attribute tokens, this can be one, none or many attributes. Each attribute has three specific parts. The first part contains the name of the attribute, the second part contains the type while the third contains the default value.

XMLDTDAttribute

This class stores data about an attribute list and provides methods for controlling access to the data. It is created by the XMLDTDTree. It stores the name of the Attribute list and a reference to the AttDef which breaks the attribute list into attributes.

AttDef

This class takes in the list of attributes in the attribute list and it separates them into separate attributes. It provides a vector for storing the attributes and methods for accessing them.

XMLAttToken

This class contains the information about an attribute. It contains it's name, type and default value along with methods for accessing this data. It is the base level of an attribute.

The classes within the parser are very generic. They were designed to parse an extensible DTD so no specific information was hard coded into the parser.

5.3.3 Backend Operations

There are a number of classes that perform essential tasks for the LOMAssistant that are behind the GUI. These will be outlined briefly in the following section. Figure 5.7 presents a class diagram.

CreateDom

This class takes the tree structure generated by the DTD parser and reads it into the W3C's DOM model. Methods for accessing elements and attributes from this model are provided by this class along with methods for finding particular elements and saving values to any given element.

The task of finding the correct element within the DOM proved to be quite tricky. The DOM model provides a specific API for accessing the internal structure. If a particular element is required, the user must use a method *getTagByName* , with the name of the element as a parameter and they are returned a list of all the occurrences of that element name. Although this sounds straightforward, there could be any number of elements in the list, and the problem remained of how to find the correct one. Table 5.2 shows some of the possible paths for one particular tag called Identifier. It occurs a large number of times and in some cases, the pathways are very similiar, differing in only one tag.

This issue was resolved by finding the pathway of the tag in question from the root of the tree and comparing this path against that of each possible element in the list. Other API methods were required to get the name of the element in questions parent. The element was only considered correct, when the two pathways were found to be identical, as a number of elements had similar parents in the tree up to a certain level.

LomHelp

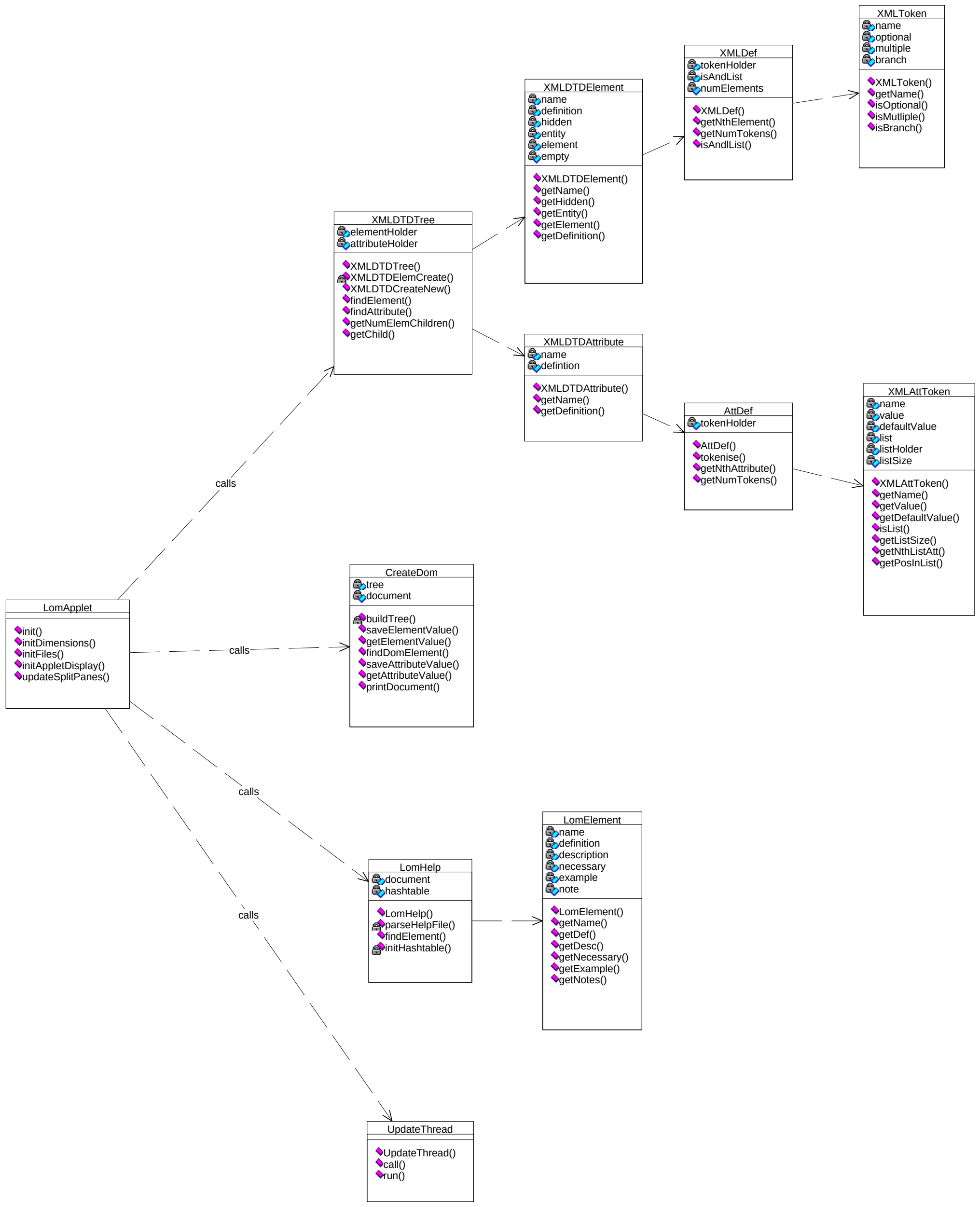


Figure 5.7: Class Diagram of the LOMAssistants backend operations

Tagname	Pathway from the tag Identifier to the root
Identifier	Gemstone, General, Identifier
Identifier	Gemstone, LifeCycle, Create, Contribute, Person, Identifier
Identifier	Gemstone, LifeCycle, InitiatedBy, Person, Identifier
Identifier	Gemstone, Metametadata, Create, Contribute, Person, Identifier
Identifier	Gemstone, Metametadata, Validate, Contribute, Person, Identifier
Identifier	Gemstone, Technical, LocSpec, Identifier
Identifier	Gemstone, Educational, EducationalUse, Prerequisite, Identifier
Identifier	Gemstone, Educational, EducationalUse, EducationalObjective, Identi fier
Identifier	Gemstone, Relation, Resource, Identifier
Identifier	Gemstone, Annotation, Person, Identifier

Table 5.2: An example of element paths from the root

key	Data
General	LomElement
LifeCycle	LomElement
Identifier	LomElement

Table 5.3: Hashtable showing the storage of Help Data

This class manages access to the help file of information. Essentially, an XML parser is used to read in the help file directly into the DOM model from which the information can be accessed. The interface to the DOM was found to be relatively slow, so it was decided to put the help information into a hash table. The use of a hash table would enable the information to be accessed quicker. Each segment of help information relative to one tag was put into an instance of the LomElement class. The hash table is made up from these LomElements with the name of the tag as the key to the table, as illustrated in table 5.3.

LomElement

A LomElement contains all the data relative to one particular help instance. It contains the elements name, official GEMSTONEs definition as defined in [DWN+99], a description of the definition, whether the tag is necessary, an example use and a section for miscellaneous information. The class also incorporates methods for accessing this information.

UpdateThread

The purpose of this class was to update the visual display containing the XML without interfering with the processing taking place on the display. It was designed as a thread so that it could perform this task in parallel to any other task. It was implemented using the methods *wait()* and *notify()* so that the class 'waited' until the DOM model which stores the XML was updated, then it was 'notified' that it had work to do.

5.3.4 GUI Implementation

The GUI was created using the Java Swing classes. There was one issue that was found to be particularly difficult in using these classes. It was found to be very difficult to place a Swing component at an exact location. There are a number of predefined methods in the Swing API, yet much effort and time was spent trying to co-ordinate the placement of components across multiple classes. There was a lack of material available on the use of the Swing classes, only one real source was found to be very useful [SUN]. This problem was encountered in the greater part of the following classes.

The graphical user interface is highly object oriented. Figure 5.8 illustrates the classes contained in the GUI. It can be visualised as one large panel with a number of smaller panels added on top. A panel is a lightweight component which acts as a container for anything that is added to it. The classes TabTextPane and SplitPane are used as container classes. They each display the data added to them in a specific way.

TabTextPane

This class creates the topmost panel of the GUI illustrated in figure 5.9 It uses the Swing component *JTabbedPane* which allows a number of components to be displayed in the same space. The user chooses which component to view by selecting the tab corresponding to the desired component. In this case, the user can choose between viewing two different TextPanes. One shows the HTML source code of a page while the other shows the XML metadata generated that has been generated.

TextPane

This class is used to display text on the screen. It uses a *JTextArea* which has methods for displaying and editing text. There are two instances of this class

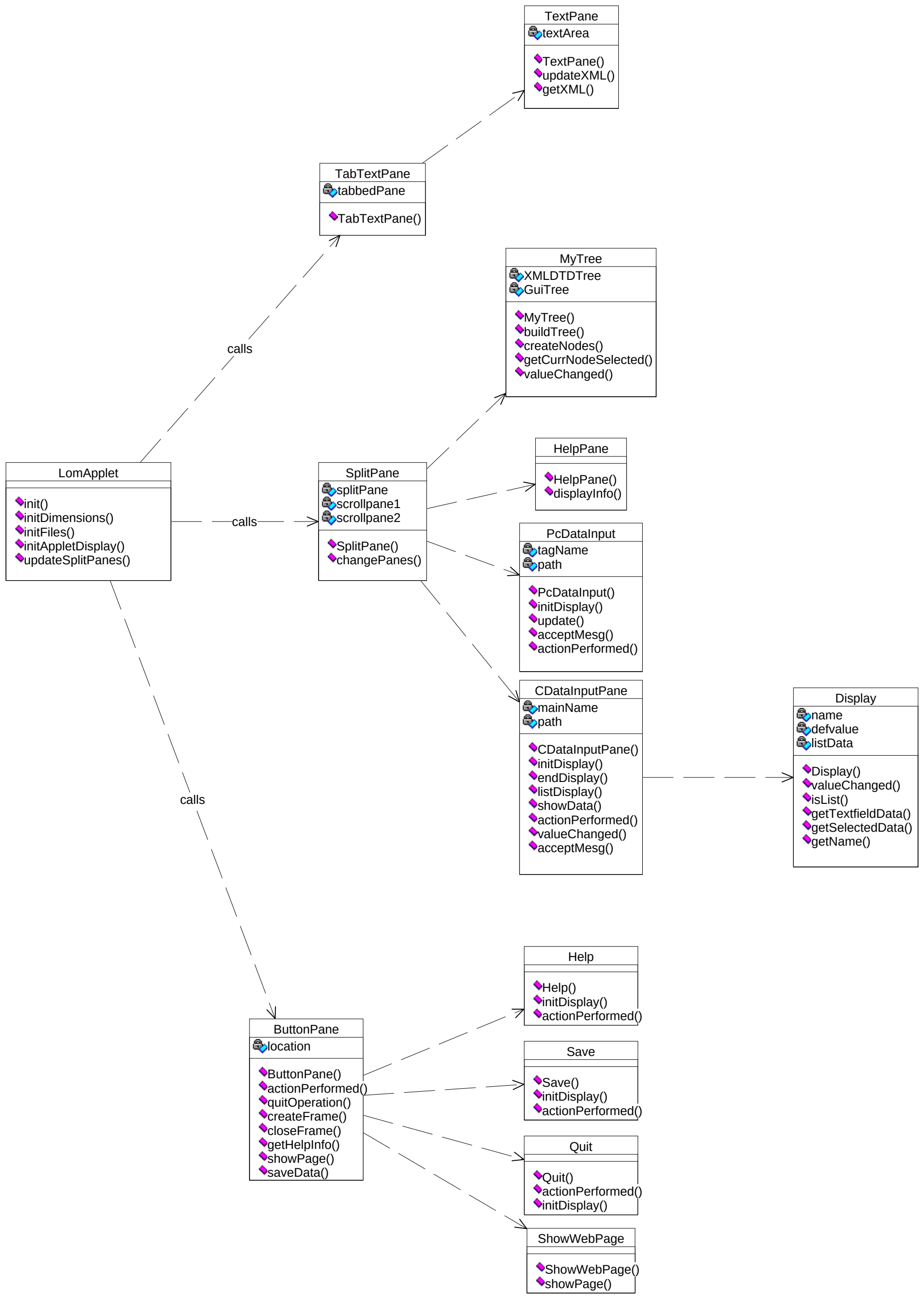
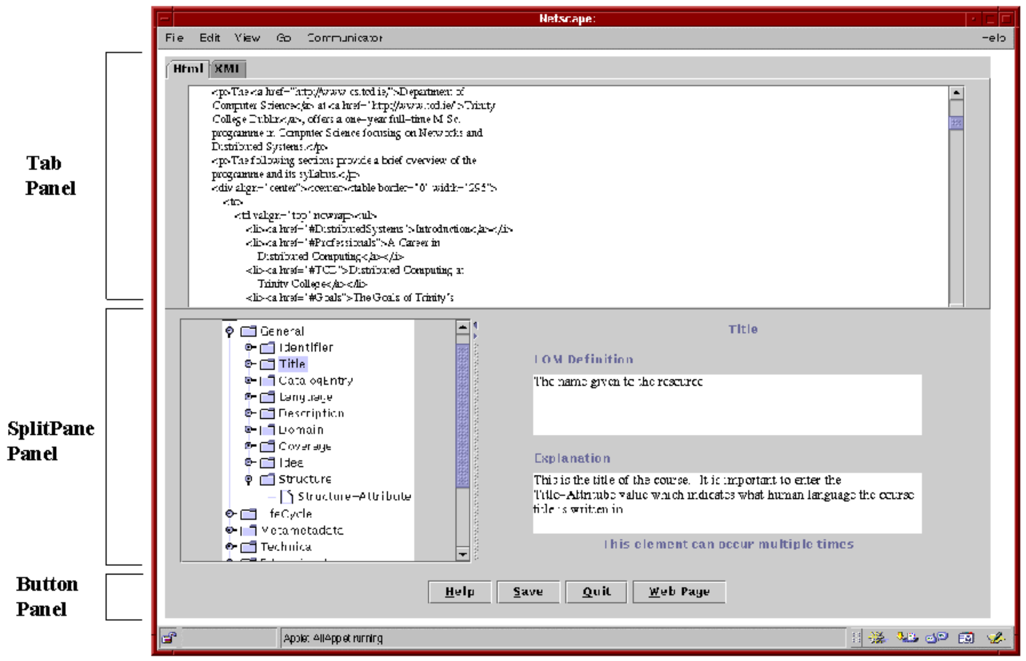


Figure 5.8: Class Diagram of the LOMAssistants GUI



```
<p>The <a href="http://www.cs.tcd.ie/">Department of
Computer Science</a> at <a href="http://www.tcd.ie/">Trinity
College Dublin</a>, offers a one-year full-time M.Sc.
programme in Computer Science focusing on Networks and
Distributed Systems.</p>
<p>The following sections provide a brief overview of the
programme and its syllabus.</p>
<div align="center"><center><table border="0" width="255">
  <tr>
    <td valign="top" nowrap><ul>
      <li><a href="#DistributedSystems">Introduction</a></li>
      <li><a href="#Professionals">A Career in
        Distributed Computing</a></li>
      <li><a href="#TCD">Distributed Computing at
        Trinity College</a></li>
      <li><a href="#Goals">The Goals of Trinity's
```

The name given to the resource

This is the title of the course. It is important to enter the Title-Attribute value which indicates what human language the course title is written in

This element can occur multiple times

Figure 5.9: A screenshot of the LOMAssistant initial Gui

used. One displays the HTML source code of the courseware page for which the user is creating metadata, while the other displays the XML metadata that the user is in the process of creating. The XML textarea was designed to be uneditable. This was so that the user would not try and edit it and change the data contained within the area manually. This is because within the model, there are a number of tags that occur more than once. If the user tried to change the data for one of these tags manually, they might change the wrong instance of the tag without realising it.

SplitPane

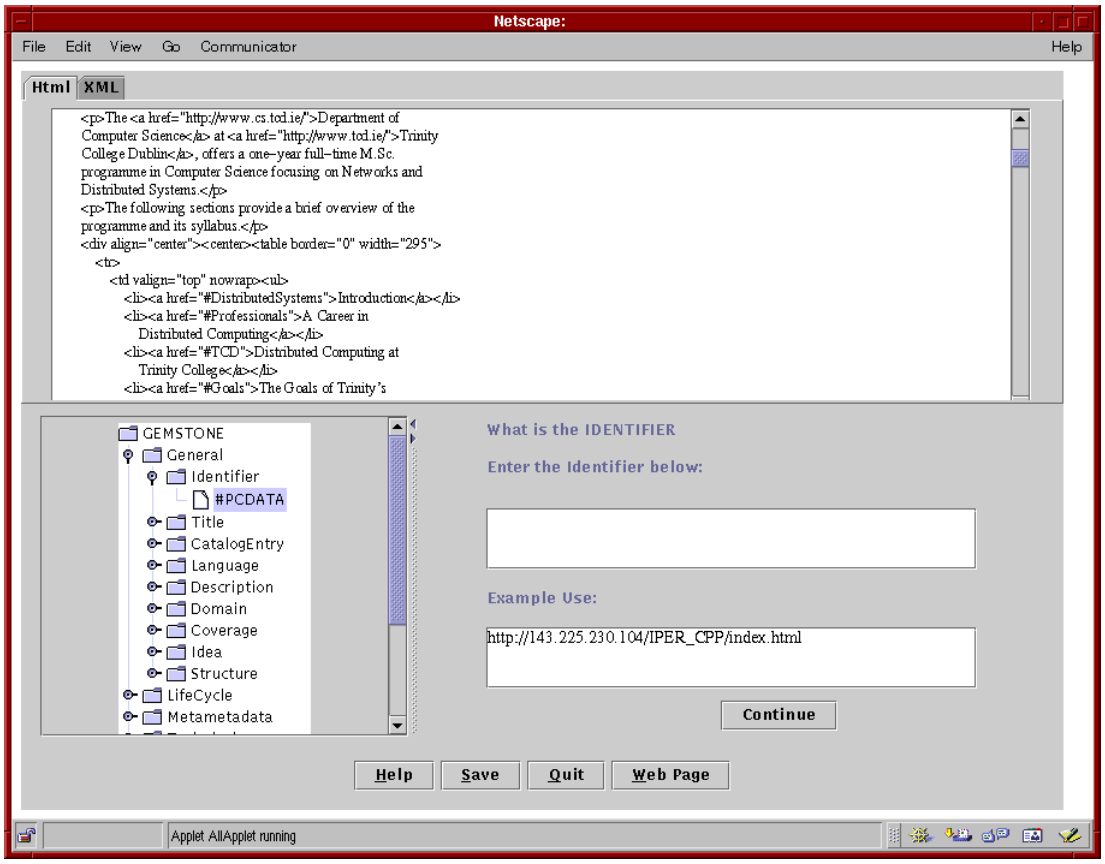
The SplitPane is the container class that makes up the middle section of the GUI, shown about in figure 5.9. It uses a *JSplitPane* to display two components side by side. The user can drag the divider that appears between the two components over and back and specify how much space is given to each component. This is desirable when one side of the component contains a huge amount of information and you want a more spacious view of it. The two components contained in this panel are a tree and a panel used to display different information to the user, depending on what the user selects in the tree.

MyTree

This is the most important class on the GUI. It displays all of the information contained in the DTD in a tree format. It uses the *JTree* Swing componet to draw the tree. The DTD is read into the applet, parsed by the DTD parser and then a method was written to recurisivly traverse the XMLDTDTree created by the parser to create the visual tree. The visual tree is very important for a number of reasons.

It provides a visual way for the user to interact and understand the model. The tree is interactive so that if the user clicks on a node or a branch, an event is generated and different data appears on the screen. The tree consists of a number of folders with names beside them as shown in figure 5.9. If a folder is present, it signifies that there are further levels below the folder. The user must traverse these folders until they encounter a page icon. This page icon signifies that the user must enter information.

There are two types of data that the user can enter. They can enter element data indicated by a '#PCDATA' page in the tree shown in figure 5.10. As soon



```
<p>The <a href="http://www.cs.tcd.ie/">Department of
Computer Science</a> at <a href="http://www.tcd.ie/">Trinity
College Dublin</a>, offers a one-year full-time M.Sc.
programme in Computer Science focusing on Networks and
Distributed Systems.</p>
<p>The following sections provide a brief overview of the
programme and its syllabus.</p>
<div align="center"><center><table border="0" width="295">
  <tr>
    <td valign="top" nowrap><ul>
      <li><a href="#DistributedSystems">Introduction</a></li>
      <li><a href="#Professionals">A Career in
        Distributed Computing</a></li>
      <li><a href="#TCD">Distributed Computing at
        Trinity College</a></li>
      <li><a href="#Goals">The Goals of Trinity's
```

```
GEMSTONE
  General
    Identifier
      #PCDATA
    Title
    CatalogEntry
    Language
    Description
    Domain
    Coverage
    Idea
    Structure
  LifeCycle
  Metametadata
```

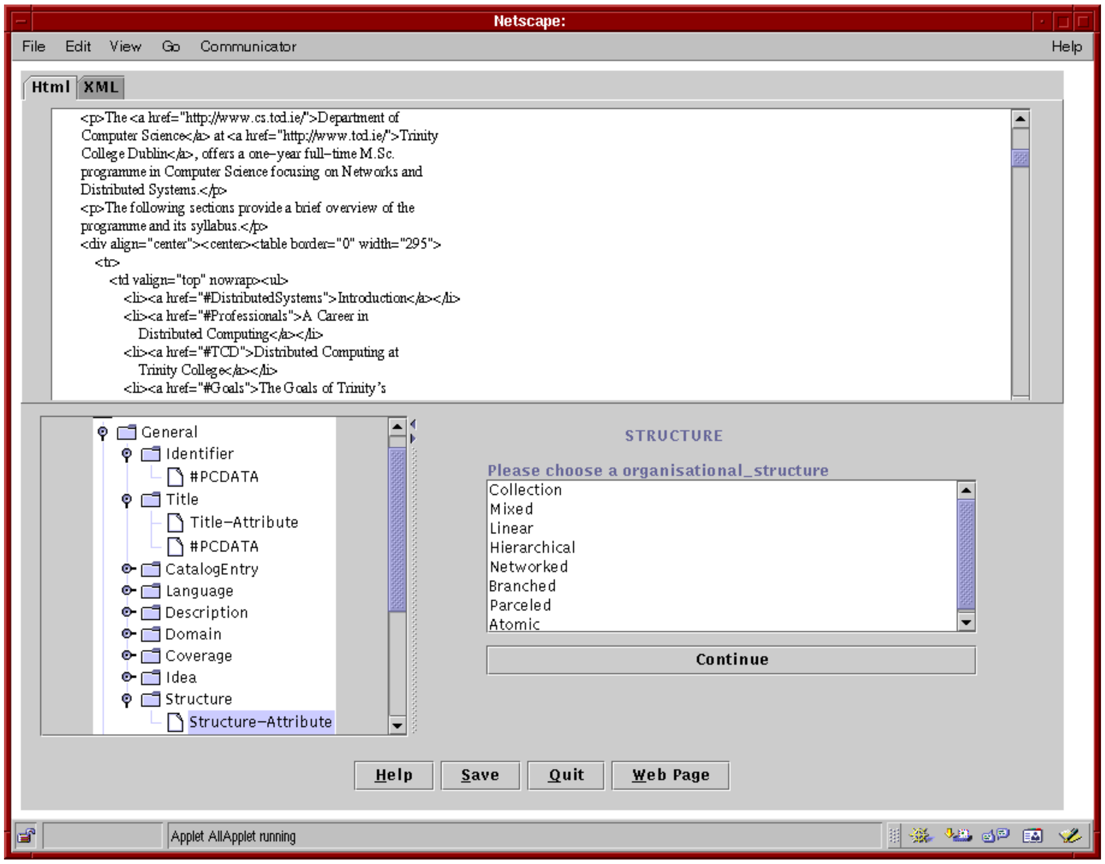
What is the IDENTIFIER

Enter the Identifier below:

Example Use:

http://143.225.230.104/IPER_CPP/index.html

Figure 5.10: An element data entry screen of the LomAssistant



```
<p>The <a href="http://www.cs.tcd.ie/">Department of
Computer Science</a> at <a href="http://www.tcd.ie/">Trinity
College Dublin</a>, offers a one-year full-time M.Sc.
programme in Computer Science focusing on Networks and
Distributed Systems.</p>
<p>The following sections provide a brief overview of the
programme and its syllabus.</p>
<div align="center"><center><table border="0" width="295">
  <tr>
    <td valign="top" nowrap><ul>
      <li><a href="#DistributedSystems">Introduction</a></li>
      <li><a href="#Professionals">A Career in
        Distributed Computing</a></li>
      <li><a href="#TCD">Distributed Computing at
        Trinity College</a></li>
      <li><a href="#Goals">The Goals of Trinity's
```

```
General
  Identifier
    #PCDATA
  Title
    Title-Attribute
    #PCDATA
  CatalogEntry
  Language
  Description
  Domain
  Coverage
  Idea
  Structure
    Structure-Attribute
```

STRUCTURE

Please choose a organisational_structure

```
Collection
Mixed
Linear
Hierarchical
Networked
Branched
Parceled
Atomic
```

Figure 5.11: An attribute data entry screen of the LomAssistant

as the user clicks on this page the other panel adjoining the tree in the splitPane changes to ask the user for input data. This calls a PcDataInput class. The second type of data that can be entered is attribute data. This is indicated by a 'name-Attribute' as seen in figure 5.11 where the name changes depending on the element the attribute is linked to. This calls an instance of the CDataInput class.

Traversal of the tree also generates an event to display helpful information to the user. If the user clicks on a folder icon, information about the name of that icon is displayed on the panel beside the tree. This information is contained in the LomHelp class and it a method to find the correct tag and extract the required information must be called.

PcDataPane

This class provides a way of asking the user for element data. It displays the element name and a *JTextArea* which the user can enter a value into as presented in figure 5.10. It displays a previous value if there is one in this *JTextArea*. It also shows an example use for the particular element, in an effort to provide assistance to the user. The data is submitted by pressing the *JButton* at the bottom of the pane. No data is actually stored in this class, it just displays data. It must pass any value entered back to the parent class who deals with the storage of information.

CDataPane

This class is used to recieve input data for an attribute list. It is shown in figure 5.11. An attribute list can contain one or many attributes so this component has to provide a way of inputing data for a number of components. These attributes often have default values and value lists from which the user must choose. Another class was created to actually display each different attribute on the panel, this was called the Display class. A number of these are added to the display depending on the number of attributes in the attribute list. A *JButton* is placed at the end of the panel, the user must press this when they want to submit their data. As in the PcDataPane, no data is stored in this class, it must pass it back to the parent class who puts it into the DOM.

Display

This class is called by the CDataPane to display information related to one instance of an attribute. It provides mechanisms for choosing values from a list or entering textual data. It uses a *JList* to show value lists and a *JTextField* to recieve textual input.

HelpPane

This class displays information about a particular tag on the screen for the user. It is intended to provide extra knowledge for the user. It is just a display component and must get the information via the parent class from the LomHelp class where it is stored.

ButtonPane

This class displays a number of command buttons on the screen. It catches the events generated when a user clicks on any of these buttons and processes the appropriate operations.

Save, Help and Quit

These three classes are pop up frames that display information to the user, using a *JFrame*. They are called when a user clicks on a button in the buttonPane. They catch user generated events. The save operation sends the information back to the server using a HTTP post request. The class Load uses the class SaveData to save the information to a file. These classes were outlined in section 5.2.4

ShowWebPage

This class opens up a separate window for displaying the courseware page for which the user is creating metadata. It is called when the user clicks a button in the buttonPane and can be closed at any time by the user.

5.4 Summary

This chapter has discussed the tools used in the implementation of this system and the modularised classes were outlined. The implementation details of the search and retrieval system and the metadata creation tool were presented.

Chapter 6

Evaluation and Conclusion

6.1 Introduction

This chapter evaluates the dissertation and presents the conclusions that have been drawn throughout the course of this research. It discusses whether the objectives of the thesis were met and it examines the effectiveness of the tool, it's limitations, the problems encountered and what extensions would be necessary to improve the system. It analyses the metadata model upon which the tool was based and provides a review of the system.

6.2 Evaluation of the thesis objectives

The three objectives of the thesis were addressed throughout the course of the research. The first objective of this thesis was to investigate the metadata approach towards information retrieval of educational courseware on the WWW. Chapter 2 presents the detailed research into the field of information representation on the Internet along with an examination of the metadata models used in both the general and educational domain. These metadata models showed the semantic and syntactic structure of how information is currently being represented. They provided a basis for comparison with the models developed in the implementation projects such as ARIADNE and GESTALT.

An investigation of the representation and search technologies was also undertaken in this chapter. XML was examined due to the support it provides for metadata resources. It was also looked at in order to ascertain how documents stored in this format could be accessed both automatically and semi-automatically. The DOM model was found to provide a tree- based mechanism for interfacing with XML. Search technology was the second form of technology investigated. Search Engines and Intelligent Agents were compared and contrasted with a view to choosing the best information retrieval method that most suited this project.

An analysis of the current existing metadata tools was carried out to find out how they operated, what services they supported, what limitations they had and how they could be improved upon. This provided a valuable insight into the field of metadata creation.

The research in this area would have been incomplete without an investigation of the GESTALT project. Chapter 3 described the GESTALT project and outlined the broader use of a search and retrieval tool that could create metadata according to a metadata model.

All of the research outlined above aided greatly the design process of a tool for the searching and retrieval of educational courseware and the creation of metadata. It helped identify the requirements necessary for this type of tool and showed how the web should be the medium for providing access to the tool.

A prototype tool was built according to the architecture designed in chapter 4. It was found to meet the requirements developed throughout the course of the research. The tool supported automatic and semi-automatic retrieval of educational resources and services and created LOM compliant metadata. Chapter 5 describes how the tool was implemented.

6.3 Evaluation of the prototype tool

As the tool was both designed and implemented in two sections, it was also evaluated in this manner.

6.3.1 The Search and Retrieval System

Ht://Dig was the name of the search engine used to implement the search and retrieval system. It was found to perform well as an off the shelf engine. It took in a search query, searched its database and generated results according to a standard accepted format. The results were found to be reasonably accurate and precise. The search engine component of the tool was interoperable so that in fact any search engine could be plugged in to replace Ht://Dig. The system was generic enough to allow any engine to be used.

There was one major limitation in using the search engine Ht://Dig. The engine had to be pre-configured to point at particular sites before it could be used. These user could choose these sites but it they were to be changed, the engine had to be reconfigured.

6.3.2 The Metadata Creation Tool

The LOMAssistant tool generated LOM compliant XML metadata. It read in the GEMSTONES DTD, which was used as a basis for the metadata. Only tags defined in the DTD were made available to the user, ensuring the compliance of any metadata created.

There was a graphical interface to the creation tool. Although it provided much functionality, there was a simple visual interface, which was easy to navigate. It was found to be relatively user friendly and quite clear. The GUI was event driven and captured user actions, dealing with them appropriately.

The tool was based on a tree model, which seems to be the preferred format in XML metadata tools. The DTD was based on a tree structure, as was the DOM model so this was reflected in the use of an interactive tree on the GUI.

The LOMAssistant was customisable. An XML file of help information was created about the tags in the GEMSTONES DTD. This was created in XML format so that it could be extensible and modified whenever necessary.

The tool was found to be extensible as very little information was hard coded into the implementation of the tool. The DTD parser created was generic enough to process any DTD, not just the GEMSTONES DTD. The help information was put into an XML file so that it too could be modified.

As with all tools, there are usually some limitations, and the LOMAssistant is no exception. There were a number of aspects that could be changed and improved if there was more time available.

The tool was not found to be user-friendly enough. In the tree on the GUI, the names 'PCDATA' and 'name-Attribute' were used to indicate to the user that they needed to input data. The general colour used in the screen could also be improved.

The performance of the LOMAssistant was considered to be relatively slow. This was due to the fact that most operations involved either a search for information in the DOM model or a search for help data from the XML file. The size of the DTD upon which this tool is based is very large therefore the DOM model which stores this information is also quite sizeable. The DOM model turned out to be quite slow to access.

Although the tool informed the user that an element was required, fixed or optional, it did not force the user to enter a value when one was outlined as required.

There was one major issue identified as a problem for the tool. Due to the sheer size of the GEMSTONES model, it took users on average between 45 minutes and one hour to create metadata for a course. This was seen as too time consuming a process. There are a large number of required tags and there is no way that the process could be completed without filling in all of these, if the user's intention was to create valid XML. This problem is seen as beyond the scope of the project and directly related to the metadata model GEMSTONES adopted by the GESTALT consortium.

6.4 Evaluation of the metadata model used

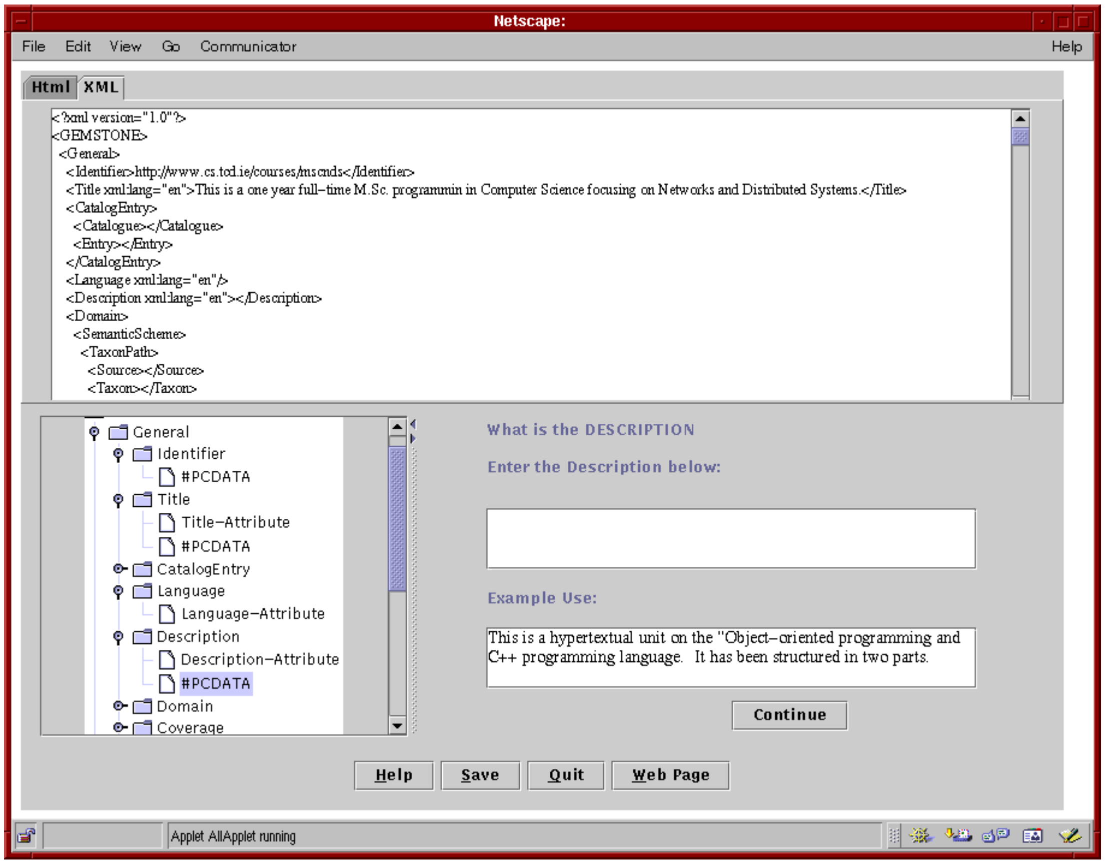
The aim of the LOMAssistant was to create metadata for educational courseware according to the GEMSTONES metadata model. Therefore the tool was dependent on the model. There were a number of issues that arose due to this factor.

The size of the model had a big impact on this tool and any other tool being created to deal with it. There were over 90 tags in the model and the user had to be presented with these tags in a certain order according to the structure predefined in the DTD. A visual way of presenting the tags had to be designed along with a method for storing them. A tree was used in this project as it provided a structural way of presenting a large number of tags in a relatively small space, according to a hierarchy.

The size of the model also affects the way in which they can be stored and accessed. The storage mechanism had to provide a relatively fast way of searching and accessing elements. The DOM was used in this system because it was based on a tree structure and it preserved the structure of the DOM, however it turned out to be very slow to access.

Mechanisms for interacting with the DTD data are required by the tool. If a user wishes to create an element from the model, they need to know the structure of the element in terms of what tags go before it and whether it has any children tags. They also need to know if it has a default type or default value. A search of the model must be undergone any time this information is required. This could lead to performance problems as the greater the number of tags, the longer it takes to search for one.

The most important question that was identified by the use of this model was whether all of the 90 tags were really necessary for the average user. The model



```
<?xml version="1.0"?>
<GEMSTONE>
 <General>
  <Identifier>http://www.cs.tcd.ie/courses/mscnds</Identifier>
  <Title xml:lang="en">This is a one year full–time M.Sc. programmin in Computer Science focusing on Networks and Distributed Systems.</Title>
  <CatalogEntry>
   <Catalogue></Catalogue>
   <Entry></Entry>
  </CatalogEntry>
  <Language xml:lang="en"/>
  <Description xml:lang="en"></Description>
  <Domain>
   <SemanticScheme>
    <TaxonPath>
     <Source></Source>
     <Taxon></Taxon>
```

What is the DESCRIPTION

Enter the Description below:

Example Use:

This is a hypertextual unit on the "Object–oriented programming and C++ programming language. It has been structured in two parts.

Figure 6.1: Screen shot of LomAssistant

takes into account practically every possible piece of data that could be related to educational courseware. This is a great feature for specialist users who require very precise detail on certain areas. However, most of these tags are not actually relevant to the greater part of the intended users. The increased use of optional tags would provide for these users, as it would eliminate the sections of the model that are irrelevant to them.

6.5 Review of the System

A system has been developed which searches for educational courseware, retrieves it and provides a mechanism for creating metadata for this courseware in accordance to a standard metadata model. The system is extensible, relatively user-friendly and easy to use. A screenshot taken from the metadata creation part of the system is presented in figure 6.1.

The users first view of the system presents them with a web page from which they have the option to make a search query or to create metadata for a particular URL. A user can initiate a search by submitting a query. This query is taken by the search engine and used to search the database. A list of possible results is returned, from which a result can be chosen and submitted for the creation of metadata. They can also initiate the creation of metadata for a particular page by typing in the URL directly into the index page. This URL must be downloaded onto the web server before the user is presented with a page from which they can initialise the LOMAssistant tool.

The LOMAssistant tool opens with the users chosen page on display at the top, an interactive tree containing the metadata model in the middle and a list of command buttons on the bottom. The user must traverse the tree and fill in data in the adjoining panel whenever they reach a leaf. This process continues until they have completely traversed the tree and filled in all required leaves. The GUI was designed to visually aid the user and provide additional help information whenever possible.

6.6 System Improvements

Although the system worked well, a number of possible improvements were identified during the course of evaluation. These were in relation to how user friendly the tool was, its performance and its lack of support for validation of the created metadata.

Meaningful names

In the LOMAssistant metadata creation tool, whenever the user reached a leaf node on the tree, it meant that they had the option to input information. The names used to differentiate between the types of information required were found to be meaningless to the user. The name 'PCDATA' was presented when element data was required and 'name-Attribute' was used when attribute data was needed. These names were taken straight from the DTD and more user-friendly names could perhaps be chosen. Although this is only a cosmetic change, the tool was designed to provide assistance and if this can be improved in any way then it should be done.

Validation

The tool did not provide any validation. It was envisaged that the system would provide validation for the XML metadata created by the LOMAssistant tool. However, although this functionality was designed, due to time constraints it was not implemented. This feature would definitely enhance the use of the tool.

Performance

The performance of the LOMAssistant was considered to be relatively slow. This was due to the fact that most operations involved a search for information in the DOM model. The size of the DTD upon which this tool is based is very large therefore the DOM model which stores this information is also quite sizeable. The DOM model turned out to be quite slow to access. This was unforeseen in the design stages of the tool and only realised when the implementation was complete. The tool would be more satisfactory to use if it performed faster.

Type enforcing

Although the tool informed the user that an element was required, fixed or optional, it did not force the user to enter a value when one was outlined as required. This functionality would greatly aid the user in creating valid XML metadata.

Generic Metadata

At the moment, the LOMAssistant only expects metadata in the format of the GEMSTONES metadata model. Support for other types of metadata must be provided in order to use the tool for editing metadata that has been created in accordance with other models.

Integration of help XML and DTD

There was another way in which the tool could be improved. At the moment the DTD and the XML help file are stored separately. It may be beneficial for other users if the two files were to be integrated and stored together. Whenever the DTD is used, it is extremely likely that there will be users curious about both the meaning and use of certain tags. After all, XML is powerful because it provides a way of dealing with both meaning and structure. The DTD addresses the structure of the document while the XML is concerned with the meaning. The two files would complement each other if they were stored together.

6.7 Future Work

There are many ways that the system could be improved and expanded. The system could become a much more highly useful product with the addition of the suggestions outlined in the previous section.

Another possible development in the area of educational resource retrieval could be the development of an educational 'crawler'. It could search the Web for relevant courses and educational courseware and populate a database with only educational data. This database could be searched rather than a generic search engine configured to point at relevant sites. A resource of this caliber could prove very valuable for automatic retrieval of educational data for many different tools and applications.

The area of brokerage and resource discovery services should be expected to play a much greater role on the Internet. The educational metadata once created in this project would prove very useful in a brokerage system. A database populated with the educational metadata related to a large number of educational resources could be searched on very specific criteria. With the demand for online courses continually increasing and the number of people using the Internet constantly growing, the use of a broker will become more and more prevalent.

6.8 Conclusion

This final chapter has described what has been learned and achieved during the course of this project. It has examined the objectives of the research, the effectiveness of the system, it's limitations, the problems encountered and what extensions would be necessary to improve the system. Possible future additions were mentioned.

In the introductory chapter there were three aims presented for this project. They have each been described in this report and implemented throughout the course of this project. This tool was implemented using all of the information and knowledge gained from research into the information retrieval and metadata fields. There are aspects of the system which could to be improved, these are outlined in the suggestions for further work.

Many thanks are due to all the people who offered suggestions which helped to further the development of this project.

Glossary

API Application Programming Interface

ARIADNE Alliance of Remote Instructional Authoring and Distribution Networks for Europe

AWT Abstract Window Toolkit

CGI Common Gateway Interface

CORBA Common Object request broker

DC Dublin Core

DTD Document Type Definition

DOM Document Object Model

FIPA The Foundation for Intelligent Physical Agents

GESTALT Getting Educational Systems Talking Across Leading-edge Technologies

GEMSTONES Gestalt Extensions to Metadata Standards for ON-line Education Systems

GUI Graphical User Interface

HTML Hyper Text Mark-up Language

HTTP Hyper Text Transfer Protocol

IEEE Institute of Electrical and Electronic Engineers

IMS Instructional Management Systems

LE Learning Environment

LOM Learning Objects Metadata

LTSC Learning Technology Standards Committee

RDF Resource Description Format

SAX Simple API for XML

SGML Standard Generalised Mark-up Language

URL Uniform Resource Locator

W3C World Wide Web Consortium

WWW World Wide Web

XML eXtensile Markup Language

Bibliography

[Alt] AltaVista. Altavista search engine homepage. `http://www.altavista.com`.

[APH] The apache software foundation. `http://www.apache.org`.

[ARI] ARIADNE. The ariadne web site. `http://ariadne.unil.ch/`.

[Con] Internet Software Consortium. Internet domain survey, july 1999. `http://www.isc.org`.

[Con98] GESTALT Consortium. D0201 review joint requirements. Technical report, 1998.

[Cora] Dublin Core. The dublin core element set specification. `http://purl.org/DC/about/element_set.htm`.

[Corb] Dublin Core. The dublin core society homepage. `http://purl.org/dc`.

[Dem97] Yu. Demchenko. Paradigm change in education in conditions of emerging new information technologies and global information infrastructure building. *Proceedings of EdMedia 1997*, pages 257–261, 1997.

[DWN+99] P Doherty, V Wade, Y Nicol, R Crawford, W Donnellyand Paul Forester, C Lambrinoudakis, M Konstantopoulos, and J S Darzentas. D0301 design and specification of the resource discovery service. Technical report, 1999.

[EDU] EDUCAUSE. The educause homepage. `http://educause.edu`.

[FIP] FIPA. The foundation for intelligent agents. http://www.fipa.org.

[Fis99] Lawrence M. Fisher. Agents and principals: The looming battle for standards. *Strategy business magazine, technology section*, 1999.

[GHC97] Shaw Green, Leon Hurst, and Dr. Padraig Cunningham. Software agent review. http://www.cs.tcd.ie/research_groups/aig/iag/toplevel2.html, 1997.

[IBMa] IBM. The aglets development homepage. http://www.trl.ibm.co.jp/aglets/.

[IBMb] IBM. The ibm xml developer site homepage. http://www.ibm.com/developer/xml/.

[IBMc] IBM. Ibms developer zone, parsing tools. http://www2.software.ibm.com/developer/tools.nsf/xml-parsing-byname.

[IBMd] IBM. Xeena xml editing tool. http://www.alphaworks.ibm.com/aw.nsf/xmltechnology/xeena.

[IEEa] IEEE. The ieee ltsc standards specification. http://www.manta.ieee.org/p1484/.

[IEEb] IEEE. The learning object metadata specification. http://www.manta.ieee.org/p1484/wg-12.htm.

[IN] Tadhg O'Meara Igor Nekrestyanov, Ekaterina Romanova. Building topic-specific collections with intelligent agents.

[JHK96] Ole Husby Juha Hakala and Traugott Koch. Warwick framework and dublin core metadata information. http://www.ub2.lu.se/tk/warwick.html, April 1996. In the proceedings of the Metadata Workshop II, Warwick, UK.

[KABL98] T Koch, A. Ard, A. Bremmer, and S Lundberg. The building and maintenance of robot based internet search services: A review of current indexing and data collection methods. 1998.

[Log] Vervet Logic. Xmlpro metadata editing tool. `http://www.vervet.com/`.

[MB] Udo Heuser Mikhail Bessonov. Open architecture for distributed search systems.

[Mor] Pierre Morel. Visual xml metadata editing tool. `http://www.pierlou.com/visxml/`.

[MTU99] Hiroshi Maruyama, Kento Tamura, and Naohiko Uramoto. *XML and Java: developing Web applications*. Addison-Wesley, Reading, MA, USA, 1999.

[oST] The National Institute of Standards and Technology. The ims metadata information web site. `http://sdct-sunsrv1.ncsl.nist.gov/~boland/ims.html`.

[PF99] Almerindo Graziano Paul Foster, Matthias Kraner. D0401 courseware metadata design (gemstones). Technical report, 1999.

[Proa] IMS Project. The background of the ims project. `http://www.imsproject.org/background.html`.

[prob] IMS project. The ims metadata specification. `http://www.imsproject.org/metadata/MDusing.html`.

[SN99] Paul Hermans Simon North. *Teach Yourself XML in 21 days*. Sams Net, Macmillian computer publishing, Indianapolis, Indiana, USA, 1999.

[Soc] The Agent Society. `http://www.agent.org`.

[SP] Andrzej Duda Stéphane Perret. Mobile assistant programming for efficient information access on the www. `http://fidji.imag.fr/map.html`.

[St.97] Simon St. Laurent. *XML: a primer*. MIS Press, P. O. Box 5277, Portland, OR 97208-5277, USA, Tel: (503) 282-5215, 1997.

[SUN] SUN. The java tutorial, creating a gui with jfc/swing. `http://java.sun.com/docs/books/tutorial/uiswing/index.html`.

[Tc] Advanced Communications Technologies and Services consortium. Introduction to the eu acts programme. `http://www.de.infowin.org/ACTS/ANALYSYS/INTRO/chap1.htm`.

[Tec] GWE Technologies. The gwe distributed search engine technical information page. `http://www.gwe.co.uk/technical/sespec.htm`.

[W3Ca] W3C. The document object model specification. `http://www.w3.org/DOM`.

[W3Cb] W3C. The w3c specification for xml. `http://www.w3.org/XML`.

[W3Cc] W3C. The w3c web site. `http://www.w3.org/`.

[W3Cd] W3C. Worldwideweb - summary. `http://www.w3.org/`.

[Wag] Tom Wagner. Searchbots, a multi-agent system. `http://mas.cs.umass.edu/research/searchbots.html`.

[Yah] Yahoo! The yahoo directory search engine homepage. `http://www.yahoo.com`.

Appendix A

LOMAssistant Help Information

A.1 Introduction

This appendix presents the LOMAssistant DTD and a simplified version of the LOMAssistant XML file with only a subset of the 90 tags shown. These files were created to accompany the tool. They provide helpful information about the tags used in the GESTALT DTD.

A.2 LOMAssistant DTD file

```
<?xml encoding="US-ASCII"?>
<!-- Dtd to accompany lomAssistant  -->

<!ELEMENT lom (lomelement)+>
<!ELEMENT lomelement (
                name,definition,description?,necessary?,example?,notes?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT definition (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT necessary (#PCDATA)>
<!ELEMENT example (#PCDATA)>
<!ELEMENT notes (#PCDATA)>
```

A.3 LOMAssistant XML file

```
<?xml version="1.0"? standalone="no">
<!DOCTYPE lom SYSTEM "LomAssistant.dtd">
<!-- For use with the LOMAssistant  -->

<lom>

   <lomelement>
      <name>General</name>
      <definition>Context-independent features of the resource.  This
      contains the basic reference to the resource.
      </definition>
      <description>This tag groups all of the context independent features
      of the resource plus the semantic descriptors for the resource.
      </description>
      <necessary>Mandatory </necessary>
   </lomelement>

   <lomelement>
      <name>Identifier </name>
      <definition>A unique label for the resource
      </definition>
      <description>This tag is used to uniquely identify the resource.  It
      is usually in the form of a Uniform Resource Locator (URL).
      </description>
      <necessary>Mandatory </necessary>
      </example>
      <notes>This element can be transparent to the metadata creator.  It
      can be created by the metadata management system. </notes>
   </lomelement>

   <lomelement>
      <name>Title</name>
      <definition>The name given to the resource
      </definition>
      <description>This is the title of the course.  It is important to
      enter the Title-Attritube value which indicates what human
      language the course title is written in.
      </description>
      <necessary>Mandatory </necessary>
      <example>Object-oridented programming and C++ course
      </example>
   </lomelement>

   <lomelement>
        <name>Language</name>
        <definition>The human language used by the typical intended user
        </definition>
        <necessary>mandatory</necessary>
        <example>en  (this would indicate english, it  - for italian )
        </example>
   </lomelement>

   <lomelement>
        <name>Description</name>
        <definition>A textual description of the content of the resource
        </definition>
        <description>A piece of writing describing what the resource contains.
        </description>
        <necessary>optional </necessary>
        <example>This is a hypertextual unit on the "Object-oriented
        programming and C++ programming language.  It has been
        structured in two parts.
        </example>
   </lomelement>

</lom>
```

Appendix B

GEMSTONES DTD

B.1 Introduction

This appendix presents the GEMSTONES DTD created by the GESTALT consortium based on the LOM model.

B.2 GEMSTONES DTD

```
<!ENTITY % SemanticScheme  "(TaxonPath,Description*,Keywords*)" >
<!ENTITY % CreateScheme  "(Date,Contribute+)" >
<!ENTITY % RequirementsScheme  "(Type,Name,MinimumVersion?,
           MaximumVersion?)" >
<!ENTITY % Address  "(Street?,TownCity?,StateCounty?,Postcode?,
           Country?,Email,Telephone?,Fax?)" >
<!ENTITY % PersonScheme  "(FirstName?,MiddleName?,PrefixToLastName?,
           LastName,Affiliation?,Identifier?,%Address;)" >
<!ENTITY % OrganizationScheme  "(Name,(%Address;)?)" >
<!ENTITY % EducationalObjectiveScheme  "(Description,Source,
           Identifier)" >
<!ENTITY % QuestionGeneral  "(Title,Create,Statement,Level?,Version?)" >
<!ENTITY % Answer  "(AnswerText,FeedBack?)" >
<!ENTITY % LocSpecScheme  "Identifier" >

<!ELEMENT GEMSTONE  (General,LifeCycle,Metametadata,Technical,Educational*,
          RightsManagement+,Relation*,Annotation*,Assessment?,QoS?) >
<!ELEMENT General  (Identifier,Title+,CatalogEntry*,Language+,
          Description*,Domain+, Coverage*,Idea*,Structure?) >
<!ELEMENT LifeCycle  (Version?,Status?,Create+,Publish*,Terminate?,
          InitiatedBy?) >
<!ELEMENT Metametadata  (Create,MetadataScheme+,Validate*) >
<!ELEMENT Technical  (Format,Size?,LocSpec+,Requirements+,
          InstallationRemarks*,OtherPlatformRequirements*,Duration?,
          OnLine?,Variant*) >
<!ELEMENT Educational  (PedagogicalType,PedagogicalClassification*,
          CoursewareGenre,Approach*,Granularity?,InteractivityLevel?,
          SemanticDensity?,EducationalUse*) >
<!ELEMENT RightsManagement  (#PCDATA) >
<!ATTLIST RightsManagement
               rights  CDATA   "INDECS" >
<!ELEMENT Relation  (Kind,Resource) >
<!ELEMENT Annotation  (Person,Date,Description) >
<!ELEMENT Assessment  (Questions) >
<!ELEMENT QoS  (DataLoss?,TimeRelated?,FlowControl?,TrafficDescriptors?,
          BitRate?) >
<!ELEMENT Identifier  (#PCDATA) >
<!ELEMENT Title  (#PCDATA) >
<!ATTLIST Title
               xml:lang  NMTOKEN   "en" >
<!ELEMENT CatalogEntry  (Catalogue,Entry) >
<!ELEMENT Language   EMPTY  >
<!ATTLIST Language
               xml:lang  NMTOKEN   "en" >
<!ELEMENT Description  (#PCDATA) >
<!ATTLIST Description
               xml:lang  NMTOKEN   "en" >
<!ELEMENT Domain  %SemanticScheme; >
<!ELEMENT Coverage  (#PCDATA) >
<!ATTLIST Coverage
```

```
               xml:lang  NMTOKEN   "en" >
<!ELEMENT Idea  (%SemanticScheme;)? >
<!ELEMENT Structure   EMPTY  >
<!ATTLIST Structure
               organisational_structure (Collection | Mixed | Linear |
               Hierarchical | Networked | Branched | Parceled | Atomic)
               #REQUIRED  >
<!ELEMENT Version  (#PCDATA) >
<!ELEMENT Status  (#PCDATA) >
<!ATTLIST Status
               status_conditions (Draft | Final | Revised | Unavailable)
               "Final">
<!ELEMENT Create  %CreateScheme; >
<!ELEMENT Publish  (Organization | Date) >
<!ELEMENT Terminate  (Date) >
<!ELEMENT InitiatedBy  (Person | Organization) >
<!ELEMENT MetadataScheme  (#PCDATA) >
<!ELEMENT Validate  %CreateScheme; >
<!ELEMENT Format  (#PCDATA) >
<!ELEMENT Size  (#PCDATA) >
<!ELEMENT LocSpec  (Identifier) >
<!ELEMENT Requirements  %RequirementsScheme; >
<!ELEMENT InstallationRemarks  (#PCDATA) >
<!ATTLIST InstallationRemarks
               xml:lang  NMTOKEN   "en" >
<!ELEMENT OtherPlatformRequirements  (#PCDATA) >
<!ATTLIST OtherPlatformRequirements
               xml:lang  NMTOKEN   "en" >
<!ELEMENT Duration  (#PCDATA) >
<!ELEMENT OnLine   EMPTY  >
<!ATTLIST OnLine
               value (Yes | No)  "Yes" >
<!ELEMENT Variant  (#PCDATA) >
<!ELEMENT PedagogicalType   EMPTY  >
```

```
<!ATTLIST PedagogicalType
          pedagogical_type (Active | Expositive | Undefined)
          "Undefined" >
<!ELEMENT PedagogicalClassification  %SemanticScheme; >
<!ELEMENT CoursewareGenre  (Features*) >
<!ATTLIST CoursewareGenre
          format (Hypertext | VideoClip | Excerise | Simulation |
          Questionnaire | Diagram | Figure | Graph | Image | Index
          | Slide | Sound | Table | Text | Exam | Experiment |
          ProblemStatement)  #REQUIRED  >
<!ELEMENT Approach   EMPTY  >
<!ATTLIST Approach
          pedagogical_approach (Inductive | Deductive | Exploratory)
           #IMPLIED  >
<!ELEMENT Granularity   EMPTY  >
<!ATTLIST Granularity
          relative_size (Course | Unit | Lesson | Fragment)
          #REQUIRED  >
<!ELEMENT InteractivityLevel   EMPTY  >
<!ATTLIST InteractivityLevel
          user_resource_interactivity (VeryLow | Low | Medium | High |
          VeryHigh)   #REQUIRED  >
<!ELEMENT SemanticDensity   EMPTY  >
<!ATTLIST SemanticDensity
          ratio_over_size_or_time (VeryLow | Low | Medium | High |
          VeryHigh)  #REQUIRED  >
<!ELEMENT EducationalUse  (UserRole,Description*,Prerequisite*,
     EducationalObjective*,Level?,Difficulty?,Duration?) >
<!ELEMENT Kind   EMPTY  >
<!ATTLIST Kind
          relation_kind (IsPartOf | HasPart | IsVersionOf |
          HasVersion | IsFormatOf | HasFormat | References |
          IsReferencedBy | IsBasedOn | IsBasisFor | Requires |
          IsRequiredBy | IsPreview) #REQUIRED  >
```

```
<!ELEMENT Resource  (Identifier | Description) >
<!ELEMENT Person  %PersonScheme; >
<!ELEMENT Date   EMPTY  >
<!ATTLIST Date
          day  CDATA    #REQUIRED
          month  CDATA    #REQUIRED
          year  CDATA    #REQUIRED  >
<!ELEMENT Questions  (MCQuestion*,SAQuestion*,CQuestion*,MatchQuestion*) >
<!ATTLIST Questions
          category  CDATA    #IMPLIED  >
<!ELEMENT DataLoss  (LossRate?) >
<!ELEMENT TimeRelated  (TransferDelay?,ArrivalTime?,StartTime?,
         StartOffsetTime?) >
<!ELEMENT FlowControl  (BufferSize?,ProbableOrder?,ConnectionCertainty?,
         DataInterference?) >
<!ELEMENT TrafficDescriptors  (NetworkRestriction*) >
<!ELEMENT BitRate  (UnspecifiedBR?,AvailableBR?,VariableBitRate*,
         ConstantBR?) >
<!ELEMENT Catalogue  (#PCDATA) >
<!ELEMENT Entry  (#PCDATA) >
<!ELEMENT Organization  %OrganizationScheme; >
<!ELEMENT Features  (#PCDATA) >
<!ATTLIST Features
          xml:lang  NMTOKEN   "en" >
<!ELEMENT UserRole   EMPTY  >
<!ATTLIST UserRole
          user (Teacher | Author | Learner | Coordinator)
          #REQUIRED  >
<!ELEMENT Prerequisite  %EducationalObjectiveScheme; >
<!ELEMENT EducationalObjective  %EducationalObjectiveScheme; >
<!ELEMENT Level  (#PCDATA) >
<!ELEMENT Difficulty   EMPTY  >
<!ATTLIST Difficulty
          difficulty_level (VeryLow | Low | Medium | High | VeryHigh)
```

```
                #REQUIRED  >

<!--Multiple choice question-->
<!ELEMENT MCQuestion  (%QuestionGeneral;,MCAnswer+) >
<!ATTLIST MCQuestion
               multiple_selection (yes | no)  "no"
               answer_layout (horizontal | vertical)  "vertical"
               format (text | html)  "html" >

<!--Short Answer question-->
<!ELEMENT SAQuestion  (%QuestionGeneral;,SAAnswer+) >
<!ATTLIST SAQuestion
               number_of_answer_boxes  CDATA    #IMPLIED
               format (text | html)  "html" >

<!--Calculated question-->
<!ELEMENT CQuestion  (%QuestionGeneral;,Formula,Variable+,Canswer,Unit?) >
<!ATTLIST CQuestion
               format (text | html)  "html" >

<!--Match question-->
<!ELEMENT MatchQuestion  (%QuestionGeneral;,MatchElement+) >
<!ATTLIST MatchQuestion
               left_column_type (Short | Long)  "Long"
               right_column_type (short | long)  "long"
               marking_scheme (eq_weighted | all_nothing | right_less_wrong)
               "eq_weighted"
               column_preview (yes | no)  "yes"
               format (text | html)  "html" >

<!ELEMENT LossRate  (#PCDATA) >
<!ATTLIST LossRate
               variation  CDATA    #IMPLIED  >
<!ELEMENT TransferDelay  (#PCDATA) >
```

```
<!ATTLIST TransferDelay
               variation  CDATA    #IMPLIED  >
<!ELEMENT ArrivalTime  (#PCDATA) >
<!ATTLIST ArrivalTime
               variation  CDATA    #IMPLIED  >
<!ELEMENT StartTime  (#PCDATA) >
<!ATTLIST StartTime
               variation  CDATA    #IMPLIED  >
<!ELEMENT StartOffsetTime  (#PCDATA) >
<!ELEMENT BufferSize  (#PCDATA) >
<!ATTLIST BufferSize
               variation  CDATA    #IMPLIED  >
<!ELEMENT ProbableOrder  (#PCDATA) >
<!ATTLIST ProbableOrder
               variation  CDATA    #IMPLIED  >
<!ELEMENT ConnectionCertainty  (#PCDATA) >
<!ATTLIST ConnectionCertainty
               variation  CDATA    #IMPLIED  >
<!ELEMENT DataInterference  (Frequency?,ModulationTechnique?) >
<!ELEMENT NetworkRestriction  (#PCDATA) >
<!ELEMENT UnspecifiedBR  (#PCDATA) >
<!ELEMENT AvailableBR  (#PCDATA) >
<!ATTLIST AvailableBR
               variation  CDATA    #IMPLIED  >
<!ELEMENT VariableBitRate  (#PCDATA) >
<!ATTLIST VariableBitRate
               bit_rate_type (NonRealTime | RealTime)   #REQUIRED  >
<!ELEMENT ConstantBR  (#PCDATA) >
<!ATTLIST ConstantBR
               variation  CDATA    #IMPLIED  >
<!ELEMENT TaxonPath  (Source,Taxon+) >
<!ELEMENT Keywords  (#PCDATA) >
<!ATTLIST Keywords
               xml:lang  NMTOKEN   "en" >
```

```
<!ELEMENT Contribute  (CreationRole,(Person* | Organization?)) >
<!ELEMENT Type   EMPTY  >
<!ATTLIST Type
               type_requirement (OperatingSystem | Browser |
               WordProcessor | VideoPlayer | AudioPlayer | GraphicsPackage |
               InputDevice | OutputDevice | Processor | Protocol | Memory |
               StorageDevice | NetworkConnection)   #REQUIRED  >
<!ELEMENT Name  (#PCDATA) >
<!ELEMENT MinimumVersion  (#PCDATA) >
<!ELEMENT MaximumVersion  (#PCDATA) >
<!ELEMENT FirstName  (#PCDATA) >
<!ELEMENT MiddleName  (#PCDATA) >
<!ELEMENT PrefixToLastName  (#PCDATA) >
<!ELEMENT LastName  (#PCDATA) >
<!ELEMENT Affiliation  %OrganizationScheme; >

<!--Specific parts-->
<!ELEMENT MCAnswer  %Answer; >
<!ATTLIST MCAnswer
               format (text | html)  "html"
               correctness  CDATA    #IMPLIED  >
<!ELEMENT SAAnswer  %Answer; >
<!ATTLIST SAAnswer
               correctness  CDATA    #IMPLIED
               allow_to_answer_in_box (number | all)  "all" >
<!ELEMENT Formula  (#PCDATA) >
<!ELEMENT Variable   EMPTY  >
<!ATTLIST Variable
               name  CDATA    #REQUIRED
               min  CDATA    #REQUIRED
               max  CDATA    #REQUIRED
               dec  CDATA    #REQUIRED  >
<!ELEMENT Canswer   EMPTY  >
<!ATTLIST Canswer
```

```
               format (decimals | significant_figures)  "decimals"

               value  CDATA    #IMPLIED
               tolerance  CDATA    #IMPLIED
               tolerance_type (percent | units)  "percent" >
<!ELEMENT Unit   EMPTY  >
<!ATTLIST Unit
               importance  CDATA    #REQUIRED
               name  CDATA    #REQUIRED
               unit_space (yes | no)  "no"
               unit_case (Yes | No)  "No" >
<!ELEMENT MatchElement  (LeftColumnMatch,RightColumnMatch) >
<!ELEMENT Frequency  (#PCDATA) >
<!ATTLIST Frequency
               variation  CDATA    #IMPLIED  >
<!ELEMENT ModulationTechnique  (#PCDATA) >
<!ELEMENT Source  (#PCDATA) >
<!ELEMENT Taxon  (#PCDATA) >
<!ELEMENT CreationRole   EMPTY  >

<!--RoleInvolvement Values are a best practice vocabulary to select
from, other values can be selected-->
<!ATTLIST CreationRole
               role_involvement (Creator | Validator | Editor |
               GraphicalDesigner | TechnicalImplementor | ContentProvider |
               TechnicalValidator | EducationalValidator | ScriptWriter |
               InstructionalDesigner)  "Creator" >

<!ELEMENT Street  (#PCDATA) >
<!ELEMENT TownCity  (#PCDATA) >
<!ELEMENT StateCounty  (#PCDATA) >
<!ELEMENT Postcode  (#PCDATA) >
<!ELEMENT Country  (#PCDATA) >
<!ELEMENT Email  (#PCDATA) >
```

```
<!ELEMENT Telephone  (dialcode,number) >
<!ELEMENT Fax  (dialcode,number) >
<!ELEMENT Statement  (#PCDATA) >
<!ELEMENT LeftColumnMatch  (#PCDATA) >
<!ELEMENT RightColumnMatch  (#PCDATA) >
<!ELEMENT dialcode  (#PCDATA) >
<!ELEMENT number  (#PCDATA) >
<!ELEMENT AnswerText  (#PCDATA) >
<!ELEMENT FeedBack  (#PCDATA) >
<!ENTITY % GSTLT_QoS   PUBLIC "QoS" "gstltQoS.dtd"    >
```