JISC Information Environment Service Level Descriptions

Study for UKOLN

Matthew J. Dovey, Oxford University
(matthew.dovey@ccc.ox.ac.uk)

Contents

Figures

Tables

JISC Information Environment Overview

JISC Information Environment Broker Architecture

Current developments in the JISC's strategy for an integrated information landscape are based upon concepts developed during the MODELS workshops hosted by UKOLN. The MODELS Information Architecture (MIA) developed in these workshops aims to produce an environment where services are integrated, in that the end user need not be aware of the different systems and interfaces for the services with which they wish to interact. This separation of content from presentation is currently referred to as Content Syndication. MIA also includes mechanisms for the discovery of services.

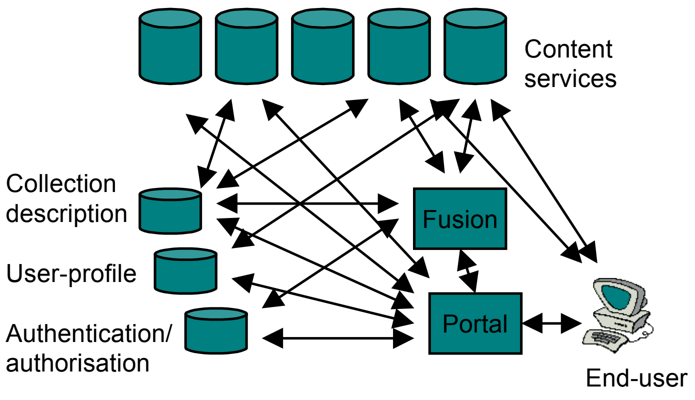

Figure 1 – JISC Information Environment Overview

The MIA forms the basis for the proposed architectures of both the Resource Discovery Network (RDN) and the JISC Information Environment (JISC IE). Figure 1 gives the outline of the MIA architecture as proposed for the JISC IE whilst Figure 2 gives a more detailed overview of the its conceptual framework.

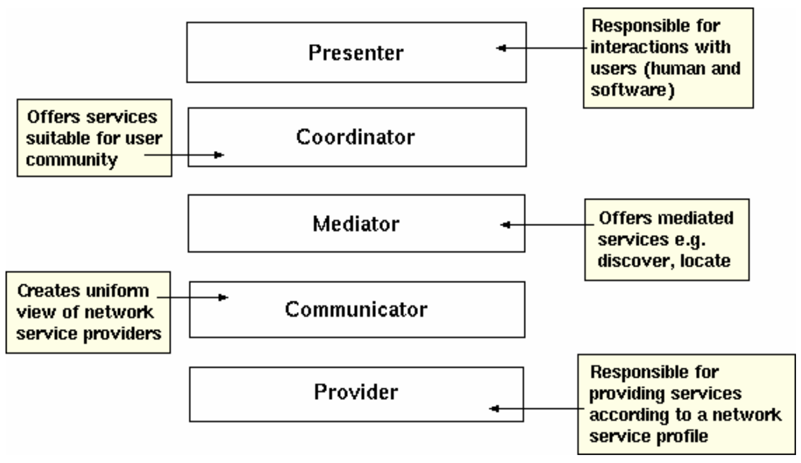
Presenter
Responsible for interactions with users (human and software)
Offers services suitable for user community
Coordinator
Mediator
Offers mediated services e.g. discover, locate
Creates uniform view of network service providers
Communicator
Provider
Responsible for providing services according to a network service profile

Figure 2 – JISC Information Environment Conceptual Framework

Clearly for this to work, both the Mediator and the Communicator need to access a registry of services. However, they require slightly different information about the services concerned. Figure 3 shows the overview of the Mediator.

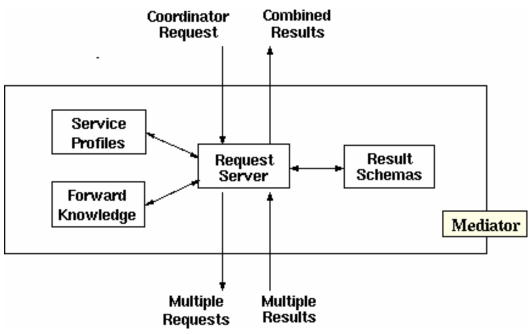

Figure 3 - Mediator Overview

The mediator needs to be able to locate services on the basis of the incoming request. It needs to be able to determine the best services for this request , namely it needs Service Profiles and Forward Knowledge. In this report, this type of metadata shall be referred to as the *service location metadata* (although particular schemas will define their own terms for this type of metadata). This location metadata helps locate the service and describes describes the nature of the service and/or data that can be accessed by the service. It may also include details of where the service is located, who runs the service, how much the service costs and other policy and management information.

The communicator requires a different type of service metadata. Figure 4 shows the conceptual framework for the Communicator.

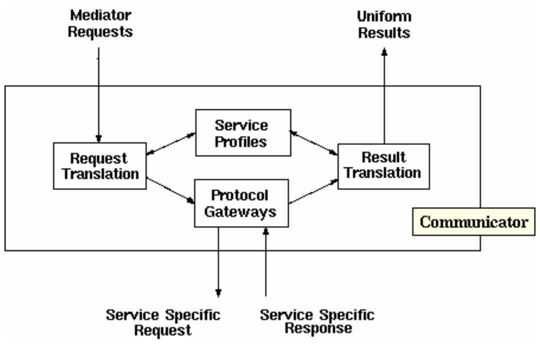

Figure 4 - Communicator Overview

The communicator needs to talk to a number of different services running numerous communication protocols (such as Z39.50, LDAP, Whois++ etc.) and translate these to the common format expected by the Mediator. This requires descriptive metadata which shall be referred to in this report as *service access metadata*. This metadata describes the technical of how to access the service, such as the protocol involved, the network address of the server and other configuration information the client would need to access the service.

JISC Information Environment Services

Z39.50 Based Services

It is anticipated that a large majority of the JISC IE services will be based on the ANSI Z39.50 Search and Retrieve protocol (ISO 23950). It is a generic search and retrieve protocol intended for use for searching any database or data source. It is a stateful protocol i.e. a connection is established and maintained between client and server throughout their interactions. It offers the following basic services:

- Init – establishes the connection, including authentication if required and other protocol negotiations
- Search – sends a query to the server and establishes a server side set of results
- Present – retrieves records from a server side set of results previously established by a Search
- Sort – sorts a server side set of results previously established by a Search
- ResultSetDelete – for informing a server that a server side set of results is no longer required
- Scan – retrieves keywords which can be used in searching from a controlled vocabulary
- Access Control – for fine grain authentication
- Resource Control – for fine grain control of server resources or costs
- Close – for terminating a session (and losing all server side sets of results)

It also offers an Explain service (detailed later) for determining the capabilities of a Z39.50 server, and Extended Services for adding additional capabilities (such as database updating). Typically a Z39.50 server will only offer a subset of these services.

Z39.50 abstracts the database to allow interoperability between clients and servers. A Z39.50 query is constructed by using search points which are then mapped by the server onto the underlying database. A set of agreed search points is called an attribute set. The most common attribute set is bib-1 used primarily for bibliographic searching and which defines search points such as author, title etc.

A Z39.50 client will also request records in a particular format called a Record Schema. If a Z39.50 server claims support for that format, it should construct records in that format from the underlying database. As a result of this, a client needs to know nothing about the underlying structure of the database in order to operate with a server, it merely needs to know the attribute sets and record schema supported by a server.

Many bibliographic Z39.50 server use MARC as the default record format. Some Z39.50 servers now also use XML. Z39.50 also has its own equivalent to XML called GRS.1. Like XML this is an extensible tree based format. In order to understand an GRS.1 record you need to know its Tag Set – this is essentially the list of element names that occur within the GRS.1 record and so is analogous to an XML DTD.

To reduce network traffic, a client can request the server to return partial records rather than full records. The Element Set defines what parts of a record are returned.

Z39.50 is a very versatile protocol in that it can act as a search retrieve protocol for almost any data source, but as a result it has a number of options. In order for a client to successfully interact with a Z39.50 server it needs to know (the following is not an exclusive list):

- The network address of the server
- Whether authentication is required
- The name of the database or databases available on that server
- What services are supported
- What attribute sets are supported
- What combinations of attributes are supported in a single query
- What record schemas are supported
- If GRS.1 is supported as a record schema, what Tag Sets are used
- What Element Sets are available

Overall, the service access metadata for a Z39.50 can be complex.

HTTP Based Services

Overview

A growing number of services are emerging based around the HTTP protocol. The simplest of these is, of course, the standard web site. Many services offer a web based user interface, and the metadata needed to describe access to such services is typically the URL of the front page. More sophisticated services may offer customised entry points based on different URLs. Amazon.co.uk for example allows access to their online bookshop through a URL which takes you to a particular book based on its ISBN. For this type of service, the access metadata required are the details of how to build up the required URL. In the case of Amazon.co.uk, a typical url would be of the form:

$$\underbrace{\text{http://www.amazon.co.uk/exec/obidos/asin/}}_{\text{Base URL}}\underbrace{\text{0563538481}}_{\text{Variable URL (ISBN)}}$$

Such approaches do not provide content syndication, as the service is responsible for both content and presentation. As such the service cannot be readily embedded into other applications, but these do offer benefits over those that only provide a single point of entry. OpenURL is another service of this type.

More sophisticated services based on HTTP do allow content syndication. These offer a URL (either a single URL or a URL format providing multiple points of entry as above) from which data can be downloaded via the HTTP protocol. Typically this data would be in the form of an XML file (either static or dynamically generated). OCS and RSS are typical services of this type.

OpenURL

OpenURL is a URL syntax designed for locating resources. The URL consists of a base component which identifies an OpenURL service running at the user's local institution and a dynamic component which contains enough metadata to identify a particular resource. The local OpenURL server then displays a web page indicating appropriate copies of the resource for that user (so for instance it may display location information for copies in that institutions library, or copies of online resources for which the institution pays a subscription). The intention is that online journal aggregators can provide links to resources tailored to individual institution. As indicated above the access information about the protocol needed to access is the format of the URL. However, the additional metadata describing how to located such services would need to be sufficient for journal aggregators to determine if a particular site offered an OpenURL resolver.

RSS & OCS

A common application of Content Syndication is to allow a portals, VLEs and MLEs to display information from regularly updated sources such as news feeds, announcements etc. as indicated in Figure 5. This type of service tends to work using a channel metaphor. A content provider would publish channels to which a portal could subscribe. The description of the channel contains information on how often the channel should be checked for updates. Channels are really just URLs from which the information can be downloaded either as XML or HTML content. A known URL is published from which a list of the channels can be located. Two formats exist for this list of channels.

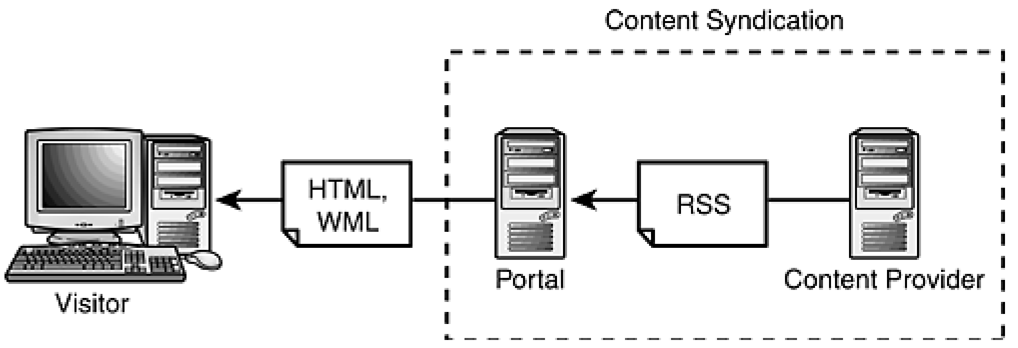

Figure 5 – RSS Content Syndication

The Open Content Syndication Directory Format (OCS) is an XML format which provides a means for a publisher list all the information channels they provide. For each channel, the XML gives some descriptive information which states how often the channel should be checked for updates, as well as a description of the nature of the content using Dublin Core. Each channel has a list of formats in which the information can be obtained: a channel can provide the same information in a number of different formats to enable portals to choose the format that suits the application best, or a format which the application can handle. The format of each channel can be as simple as a plain text feed or a html formatted text feed. However, Open Content Syndication can also act as a container for richer news feeds such as RSS or other XML based formats.

RSS (RDF Site Summary, also know as Rich Site Summary) is an XML format for syndicating news feeds and other channels of published information. Like Open Content Syndication Directory Format it provides a means of identifying channels of information and providing a description of that channels content and how often the information should be checked for updates. RSS uses RDF (Resource Description Format) as a basis for its description elements. RSS has been used for a variety of applications including discussion threads, job listings, homes for sale (multiple listings services), sports scores, document cataloguing.

There are a number of directories emerging, facilitating the location of content services. The most prominent service at present can be found at www.xmltree.com which provides a directory of a number of different XML services. The access metadata needed to access a particular news feed is the URL of the OCS or RSS file which gives the full description.

Web Services

Overview

WebServices are an upcoming technology and framework for distributed systems. They are built around a business model of software companies renting services rather than selling applications. Typical services could include language translation services, location services and of course retail based services. Content Syndication is clearly another area which could be provided by Web Services.

The client application sends requests to the server over the HTTP protocol. The requests issued are fairly similar to function calls within a software program and in many cases are forms of Remote Procedural Call (RPC). In the WebService case, the particular function called and its parameters are encoded either as a HTTP URL using the GET protocol, or more often as an XML message using the HTTP POST. The server responds to the call by returning an XML document. Most WebServices use a specification called SOAP (Simple Object Access Protocol) to define the XML messages used.

The access metadata required for such services can be quite sophisticated. The important information is the URL of the service and the structures required for the request and response messages. A fairly rich metadata format for describing this structure has been developed by the W3C call WSDL (Web Service Description Language). This defines the structure of both the request and response messages using XML schema and also how request and response messages are paired. It also provides for binding to a number of different transports and encoding such as SOAP over HTTP. See Appendix 3 – Protocol Descriptions for further details.

Two WebServices which have potential use in the JISC IE are OAI and ZiNG.

OAI

The Open Archives Initiative defines a protocol for accessing repositories of metadata. Unlike the distributed search mechanism, discovery in OAI takes place by searching a single index which spans multiple repositories. The index is updated periodically by harvesting the records for the multiple repositories. The OAI protocol defines a mechanism for performing this harvesting. It defines six operations which can be submitted by both HTTP GET and HTTP POST mechanisms and the XML responses:

- GetRecord – retrieves a record from the repository given its identifier
- Identify – returns a service description for the repository
- ListIdentifiers – returns a list of identifiers of the records in the repository specified by date range or naming a particular subset
- ListMetadataFormats – lists the record metadata formats that the repository supports
- ListRecords – returns a list of records from the repository specified by date range or naming a particular subset

- <u>ListSets</u> – returns a list of the names of subsets which can be used in ListIdentifiers or ListRecords

The service access metadata required to describe an OAI repository is essentially the base URL to which the OAI requests should be sent.

ZiNG – SRU/SRW

ZiNG is an initiative of various Z39.50 developers to investigate ways of lowering the barrier to implementing Z39.50 solutions. One component of ZiNG is a lightweight webservice based on Z39.50. At present this service only provides a single function call – SearchRetreive. The parameters of SearchRetreive are:

- Query – either a Boolean query or a reference to a previously sent query
- StartRecord – the number of the first record to return
- MaximumRecords – maximum number of records to return
- RecordSchema – the schema in which to return records

The server responds with the appropriate list of records, and optionally a hint as to how to reference that query in subsequent requests. This web services is specifed in two forms SRU which uses an HTTP encoded URL to pass the parameters and returns an XML document and SRW in which the requests and responses are encoded in SOAP.

<u>Service Level Description Schemas</u>

Explain

Explain is part of the Z39.50 protocol specification, it is intended to allow a client to query the available databases and capabilities of those databases. The Explain service is implemented as an additional database on a Z39.50 server with a well known name "IR-Explain-1". A client hence received information by performing Z39.50 queries on this database to retrieve appropriate records. There are 17 different categories of record in this database:

- TargetInfo – Details about the server. There is one such record in the Explain database
- DatabaseInfo – Details about each individual database on the server
- SchemaInfo – Descriptive information of record schemas supported by at least one database on the server
- TagSetInfo – Descriptive information of tag sets used in at least one record schema
- RecordSyntaxInfo – Descriptive information of record syntaxes available from at least one database on the server
- AttributeSetInfo – Descriptive information of attribute set used by at least one database on the server
- TermListInfo – Descriptive information of terms lists (i.e. controlled vocabulary taxonomies) used by at least one database on the server
- ExtendServicesInfo – Descriptive information of record syntaxes available from at least one database on the server
- AttributeDetails – A record for each database indicating what search attributes are used (plus alternatives providing finer or coarser access to the database)

- TermListDetails – For each term list, detailed information on the terms available (sort order etc.)
- ElementSetDetails – For each record syntax on a specified database, a record describing the element sets available.
- RetrievalRecordDetails – For each record schema on a specified database, a record describing the element tags available.
- SortDetails – For each database, the sort orders available.
- Processing – Records suggesting how the client should processing records received.
- VariantSetInfo – Descriptive information of record syntaxes available from at least one database on the server.
- UnitInfo – Descriptive information of record syntaxes available from at least one database on the server.
- CategoryList – A list of the Explain categories which are supported by the server.

An overview of how these all link together is given in Figure 6. Overall much of the information in an Explain database gives extremely details information pertaining to the Z39.50 protocol itself. This report will merely outline the location and basic access metadata which is contained with the TargetInfo and DatabaseInfo categories.

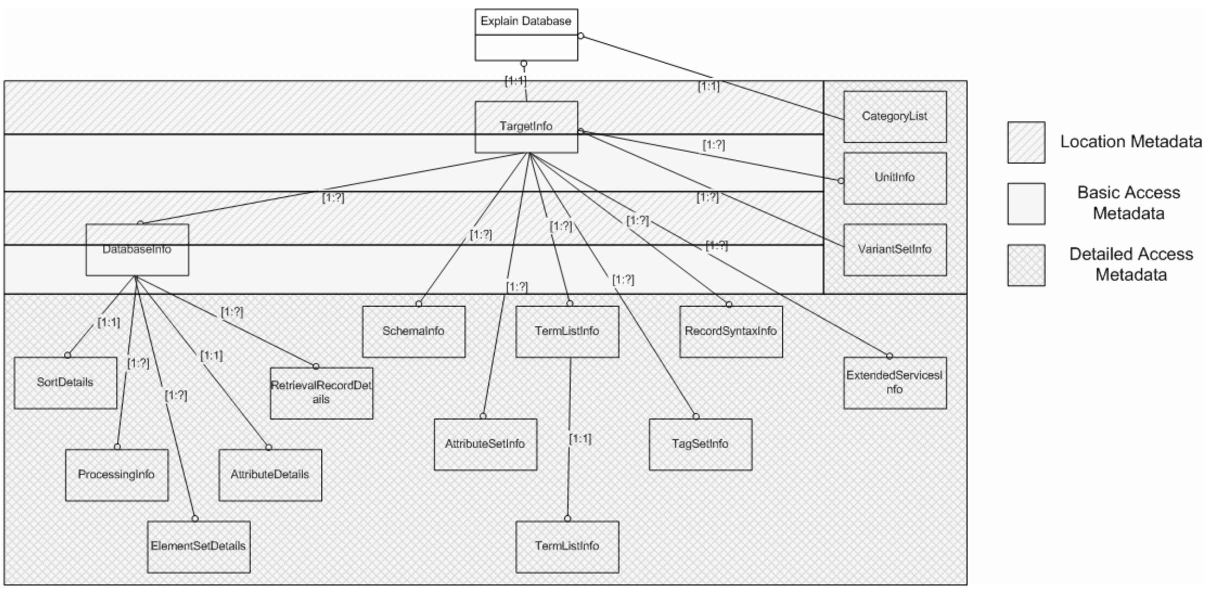

Figure 6 - Explain Database Schema

The TargetInfo record contains the following data:

- A name for the target (only one), in human readable text.
- Recent news of interest to people using this target, in human readable text.
- An icon used to represent this target (in machine presentable form).
- Whether named results sets are supported.
- Whether multiple databases can be searched in one search request.
- The maximum number of concurrent result sets supported.
- The maximum size (in records) of a result set.
- The maximum number of terms allowed in one search request.
- A timeout interval after which the target will trigger an event if no activity has occurred.

- A "welcome" message from the target to be displayed by the origin.
- Contact information for the organization supporting this target.
- A description of the target, in human readable text.
- A set of nicknames or alternate names by which the target is known.
- Restrictions pertaining to this target, in human readable text.
- A payment address (e.g. business office) for the organization supporting this target.
- Hours of operation.
- A list of supported database combinations.
- Internet address and Port number.
- Languages supported for message strings.
- The following elements, where each object listed is supported for one or more databases. (To determine which are supported for a particular database, retrieve the record for that database.)
    - Which query-types are supported, and details for each supported type.
    - Diagnostic sets supported.
    - Attribute sets supported.
    - Schemas supported.
    - Record syntaxes supported.
    - Resource challenges supported.
    - Access challenges supported.
    - Cost information.
    - Variant sets supported.
    - element set names supported.
    - Unit systems supported.

The DatabaseInfo record contains the following information:

- Full database name (only one).
- Whether this is an Explain database (possibly for a different server).
- A list of short (or alternate) names for the database.
- An icon used to represent this database (in machine presentable form).
- Whether there is charge to access this database.
- Whether this database is currently available for access.
- A human-readable name or title for the database (as opposed to the database name, which is typically a short string not meant to be human-readable, and not variable by language.)
- A list of keywords for the database.
- A description of the database, in human readable text.
- Associated databases: those that the target allows (and possibly encourages) to be searched in combination with this database.
- Sub-databases that make up this conceptual single database.
- Any disclaimers concerning this database, in human readable text.
- News about this database, in human readable text.
- A record count for the database (and whether the count is accurate or an estimate).
- A description of the default order in which records are presented, in human readable text.
- An estimate of the average record size (in bytes).
- A maximum record size (in bytes).

- Hours of operation that this database is available.
- Best time to access this database, in human readable text.
- Time of last update of this database.
- Update cycle/interval for this database.
- Coverage dates of this database, in human readable text.
- Whether this database contains proprietary information.
- A description of copyright issues relating to this database, in human readable text.
- A notice concerning copyright which the target expects the origin to display to the user if possible, in human readable text.
- Description and contact information for the database producer, database supplier, and for how to submit material for inclusion in this database, in human readable text.
- Which query-types are supported for this database, and details for each supported type.
- Diagnostic sets supported for this database.
- Attribute sets supported for this database.
- Schemas defined for this database.
- Record syntaxes supported by this database.
- Resource reports supported for this database.
- Text describing access control for this database, in human readable text.
- Costing information related to this database, in both machine readable format, and in human readable text, for connect, present, and search.
- Variant sets supported for this database.
- Element set names supported for this database, with names and descriptions given in human readable text.
- Unit systems supported for this database.

Some of the more interesting inclusions in this description are the occurrence of an icon to represent the database and/or target and suggested welcome text and news to present to the user. One of the main obstacles to Content Syndication is that content providers are concerned about losing their branding which is essential if they are to sell their services. The inclusions of icons (or logos) and content provider messages in the service metadata allows content providers to maintain their branding when engaging in content syndication scenarios

The main obstacles to using Explain as a generic service registry are

- It is designed to handle Z39.50 service access metadata only, although could be extended to other types of services by adding additional record categories.
- It is a fairly complex mechanism – there are very few Z39.50 client or servers which currently implement the Explain service
- The specification defines an Explain database which only contains a description for the local server, although it is easy to expand this to an Explain database detailing numerous remote servers.

Explain Lite

Explain Lite is an XML format performing much the same purpose as the Explain database. It was designed by the EU funded ONE and ONE-2 projects as a simpler alternative to complex Explain database. The complete schema is shown in Figure 7. The metadata for each server (referred to as host) are

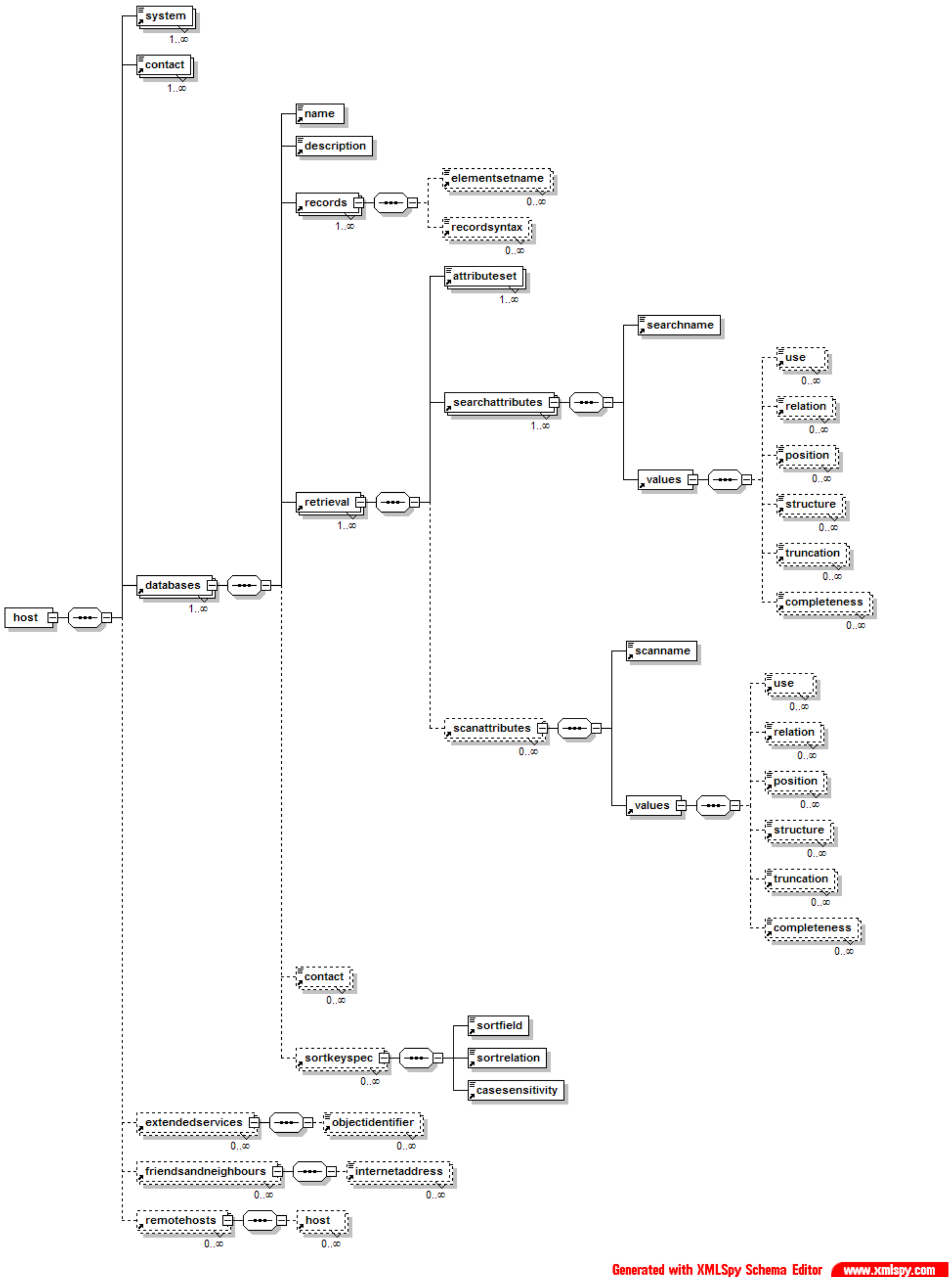

Figure 7 - Explain-Lite Record Schema

- System – includes XML attributes for a description, the internet address and port of the server, whether the server requires a username and password, and a URL for additional information about the server
- Contact – includes XML attributes for the name of the contact, description of contact (e.g. technical equiries, billing etc.), postal address, e-mail and phone
- Database – includes sub-elements for the name of the database, description of the database and Z39.50 specific service access metadata
- FriendsAndNeighbours – the internet address and port of other Z39.50 servers which have some relation to this server (e.g. other servers run by the same organisation or containing similar or related data)
- RemoteHosts – Explain-Lite descriptions of other Z39.50 servers which have some relation to this server (e.g. other servers run by the same organisation or containing similar or related data)

Explain-Lite only defines a schema for describing Z39.50 servers (although the XML could be extended to accommodate other types of services). Explain-lite does not define any mechanism for retrieving the description – it merely allows configuration information to be retrieve by a client via an previous agreed mechanism such as during the Z39.50 initialisation or by downloading the XML from a known URL. The service location metadata does not include enough information for searching a database of Explain-Lite data (e.g. no ability to search for services by keyword, subject, location etc.)

Explain--

Explain—is the current working name for an emerging development from a subset of the ZIG addressing many of the same issues as Explain-Lite but attempting to retain some of the richness of description inherent in Z39.50 Explain. Its overall schema is still very much in development but the current schema at the time of writing is given in Figure 8. There are four top level elements:

- serverInfo – service access metadata giving connection details about the server such as ip address, port, and protocol (e.g. Z39.50 or ZiNG SRW/SRU)
- databaseInfo – service location metadata describing the database
- metaInfo – general provenance of the explain—data
- indexInfo – Z39.50 service access specific information about the available attributes for searching
- recordInfo – Z39.50 service access specific information about the available attributes for searching

Explain-- only defines a schema for describing Z39.50 servers and derivatives such as ZiNG SRU and SRW (although the XML could be extended to accommodate other types of services). Explain-- is still defining issues about how the information is retrieved and distributed. However, it also includes an attribute set for search a Z39.50 database of Explain-- records. This currently allows searching on host ip name, ip port, database name and record syntaxes supported, as well a querying for the settings of the server hosting the Explain—database.

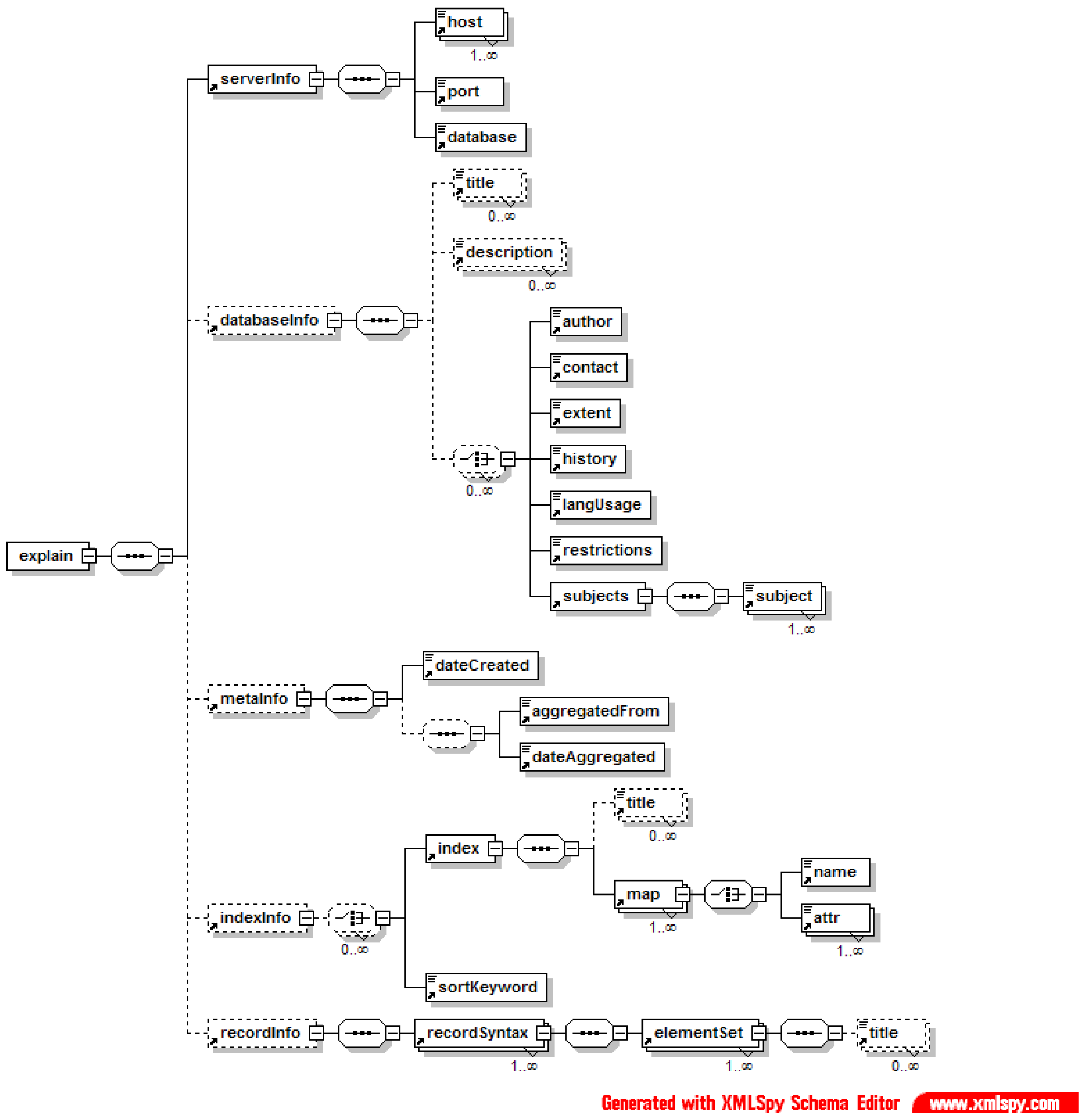

Figure 8 - Explain-- Schema

GILS Locator Records

Global Information Locator Service is a Z39.50 application profile for locating information resources, typically used within US Government for locating Government information resources. It is designed for describing any information resource, not only electronic services and therefore has a number of elements which are more pertinent to Collection Level Descriptions rather than service descriptions. The information contained with a GILS record are:

- Title – (Not Repeatable) identifies title of the resource.
- Originator – (Repeatable) identifies the information resource originator.

- Contributor – (Repeatable) used if there are names associated with the resource in addition to the Orignator, such as personal author, corporate author, co-author, or a conference or meeting name.
- Date Of Publication – (Not Repeatable) the discrete creation date in which the described resource was published or updated. Not used on resources that are published continuously such as dynamic databases.
- Place of Publication – (Not Repeatable) the city or town where the described resource was published.
- Language of Resource – (Repeatable) indicates the language(s) of the described resource.
- Abstract – (Not Repeatable) presents a narrative description of the information resource.
- Controlled Subject Index – (Repeatable) descriptive terms from controlled vocabulary to describing the resource.
- Subject Terms Uncontrolled – (Not Repeatable) descriptive terms to aid users in locating resources of potential interest, not drawn from a formally registered controlled vocabulary source.
- Spatial Domain – (Not Repeatable) identifies the geographic area domain of the data set or information resource. Can be both geographic names and coordinates defining the bounds of coverage.
- Time Period – (Repeatable) provides time frames associated with the information resource.
- Availability – (Repeatable) describes how the information resource is made available. This includes:
    - Medium (Not Repeatable) the material type of the resource, e.g. cassette, kit, computer database, computer file.
    - Distributor – (Not Repeatable) provides information about the distributor, including Name, Organisation, Street Address, City, Post Code, Country, Network Address, Hours of Service, Telephone and Fax.
    - Resource Description – (Repeatable) identifies the resource as it is known to the distributor.
    - Order Process – (Not Repeatable) provides information on how to obtain the information resource from this distributor including cost.
    - Technical Prerequisites – (Not Repeatable) describes any technical prerequisites for use of the information resource as made available by this distributor.
    - Available Time Period – (Repeatable) provides the time period reference for the when the information resource is available from this distributor.
    - Available Linkage – (Repeatable) provides the information needed to contact an automated system made available by this distributor (such as a URL)
- Sources of Data – (Not Repeatable) identifies the primary sources or providers of data.
- Methodology – (Not Repeatable) identifies any specialized tools, techniques, or methodology used to produce this information resource.
- Access Constraints – (Not Repeatable) describes any constraints or legal prerequisites for accessing the information resource.
- Use Constraints – (Not Repeatable) any constraints or legal prerequisites for using the information resource.
- Point of Contact – (Not Repeatable) identifies an organization, and a person where appropriate, serving as the point of contact for the resource (including Name,

Organisation, Street Address, City, Post Code, Country, Network Address, Hours of Service, Telephone and Fax).

- Supplemental Information – (Not Repeatable) additional information about the resource.
- Purpose – (Not Repeatable) describes why the information resource is offered.
- Cross Reference – (Repeatable) identifies other locator record or related information resources likely to be of interest.
- Schedule Number – (Not Repeatable) records the identifier associated with the information resource for records management purposes.
- Control Identifier – (Not Repeatable) defined by the information provider and is used to distinguish this locator record from all other GILS Core locator records.
- Original Control Identifier – (Not Repeatable) used by the record source to refer to another GILS locator record from which this locator record was derived.
- Record source – (Not Repeatable) identifies the organization that created or last modified this locator record.
- Language of Record – (Not Repeatable) indicates the language of the locator record.
- Date of Last Modification – (Not Repeatable) identifies the latest date on which this locator record was created or modified.
- Record Review Date – (Not Repeatable) identifies a date assigned by the Record Source for review of the GILS Record.

GILS records are typically located via Z39.50 and GILS also defines a search profile and attribute set for searching GILS registries. The GILS record itself can be represented in both GRS.1 and XML and GILS also defined crosswalks to other formats such as MARC.

GILS per se contains little access level metadata beyond access to a given URL or identifying a Z39.50 service via a Z39.50 URL (but this would not give other information as to the Z39.50 configuration). However, it would be possible to extend the GILS schema to accommodate this. However, one advantage over Explain and Explain-Lite is the introduction of tracking metadata indicating the currency of the data and how often it is reviewed, plus a mechanism for tracking the provenance of derived records.

WebClarity Resource Registry Schema

WebClarity Resource Registry is a commercial development of Sea Change Corporation. Sea Change produce a number of Z39.50 client products including a Windows based client and a web based client. They have built up a large collection of the configuration data for a variety of world-wide Z39.50 servers which they ship with their clients to offer pre-configured access a wide variety of sources. The WebClarity Resource Registry provides a globally accessible database of this information through a web interface and SOAP, with plans for Z39.50, LDAP and UDDI access.

The data in the Resource Registry is retrieved as XML records. The schema for these records is shown in Figure 9. In many aspects the record schema is an extension of a subset of the GILS schema The data in these records consists of:

- Title – title of the server. Can support both a main title and alternate titles.
- Originator – originator of the data (as in GILS). Can support a main originator and alternates.
- LanguageOfResource – language the data of the service is in.

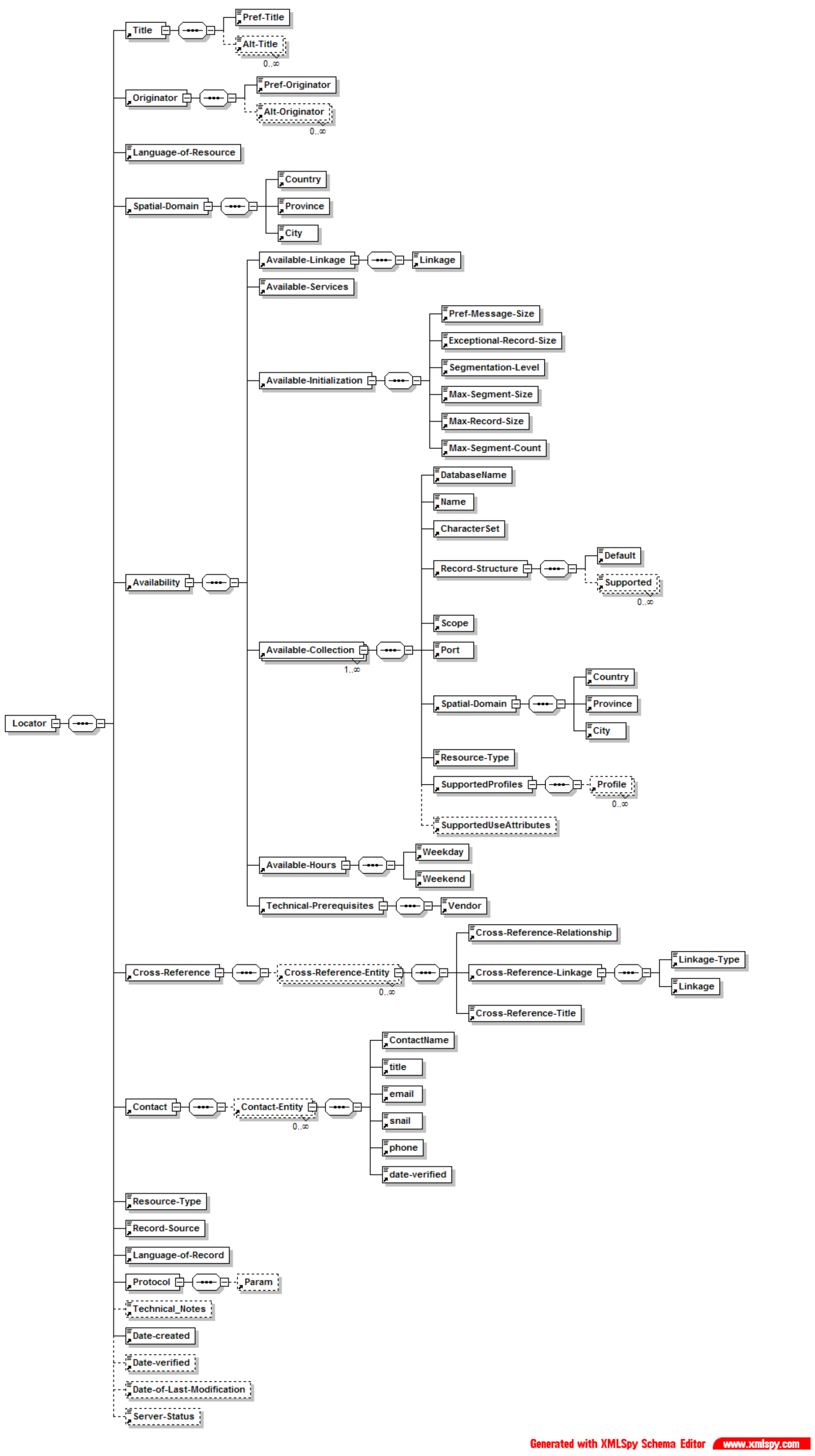

Figure 9 - WebClarity Record Schema

- SpatialDomain – the spatial domain that the data or service covers.
- Availability – this combines both availability of the service (such as hours of access, cost etc.) with service access metadata describing the Z39.50 configuration needed to access the databases on the server.
- CrossReference – identifies other server which have some relationship with the service (such as other resources offered by the same supplier, or services offering related data).
- Contact – contact addresses for the server (such as enquiries, technical, billing etc.)
- ResourceType – specifies the resource type of the host. The options are "Academic", "National", "Public", "Geospatial", "Government", "State Library", "Consortia", "Corporate" and "Other".
- Record-Source – specifies provenance of the record.
- TechnicalNotes – additional technical information for the service.
- ServerStatus – used internally for recording whether the service was available when last checked (values are live, unknown and dead).
- Protocol – specifies the servers protocol (current only Z39.50 is valid).
- LanguageOfRecord – language of the locator record.
- DateCreated – date the locator record was originally created.
- DateVerified – date the locator record was last verified.
- DateOfLastModification – date of the last change to the locator record.

The WebClarity Resource Registry currently only contains service access metadata pertaining to Z39.50 servers but indicates the intention to cover other service types in the future. Like GILS it offers a mechanism for tracking the concurrency of the service description.

UDDI Schema

Universal Description, Discovery and Integration of Web Services (UDDI) is an initiative lead by Ariba, IBM and Microsoft for developing a global registry of web services. It consists of three components: a schema for describing the services (covering both location and access metadata); a SOAP based application programming interface for accessing and publishing to registries containing UDDI schema records; and a global public registry.

In its current form the metadata model for a UDDI record is quite straightforward and its use of XML as the carrier of this information is such that it can easily be extended. There are four main metadata entities: businessEntity for describing businesses and organisations; publisherAssertions for describing the relationships between organisations; businessService for describing the different services a business or organisation may offer; bindingTemplates to describe the technical interfaces available for such a service and tModels to describe the protocols involved in using those services. The relationships between these are illustrated in Figure 10.

Within an academic context, the businessEntity could typically describe a service aggregator such as OCLC FirstSearch (it could also describe a University, or a library, museum etc.). In this case, the businessEntity could offer a number of services: WorldCat, ArticleFirst, ContentsFirst in the case of OCLC FirstSearch or different online courses in the case of an University offering online courses. The businessService describes the nature of these services. Each service may be available via different technologies as illustrated by the bindingTemplates, for instance WorldCat may have three bindingTemplates: one representing a telnet interface, one a web interface and the third a Z39.50 interface – all to the same service.

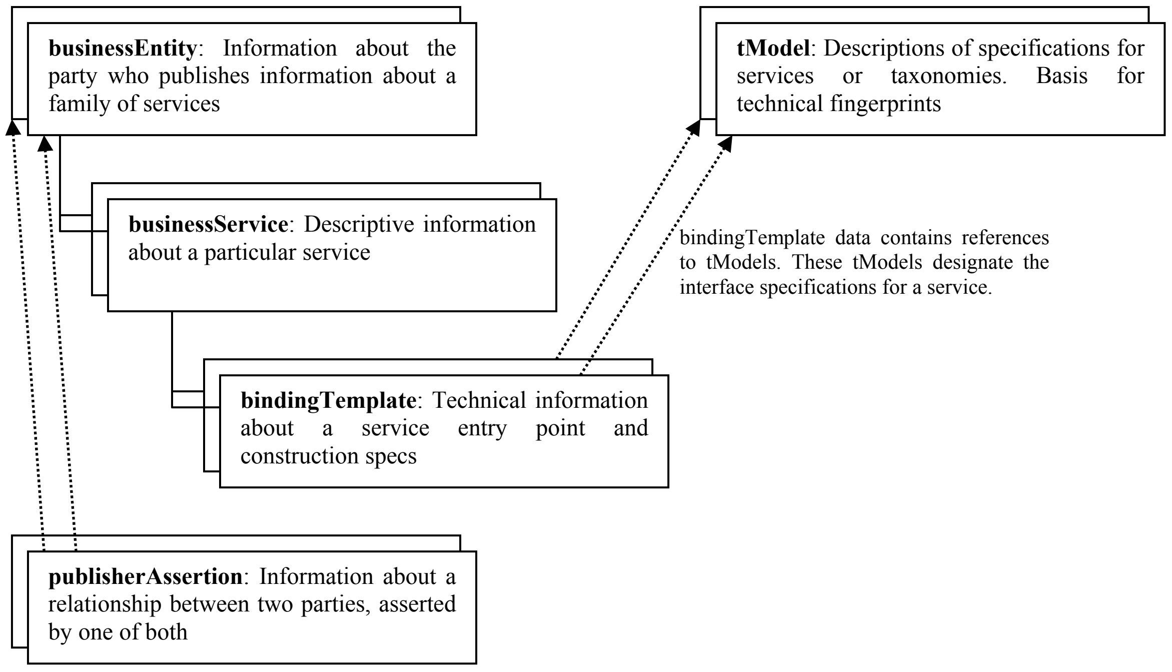
businessEntity: Information about the party who publishes information about a family of services
tModel: Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
businessService: Descriptive information about a particular service
bindingTemplate data contains references to tModels. These tModels designate the interface specifications for a service.
bindingTemplate: Technical information about a service entry point and construction specs
publisherAssertion: Information about a relationship between two parties, asserted by one of both

Figure 10 - UDDI Record Schema

The businessEntity currently contains the following information:

- Name – the name of the business or organisation
- Description – a description of the business or organisation
- Contacts – a list of contact names and details (addresses, telephone etc.). Each contact is given a "useType" which designates the role which that contact plays. Typical suggestions include technical contact, sales contact etc. Roles defined in Collection Level Description work such as that conducted by the RSLP could be included here (e.g. collector, owner etc.)
- IdentifierBag – this contains a list of identifiers. Each identifier also includes details of the type of identifier. Typical examples from the UDDI specification would include tax identifiers. Other more pertinent example for use in JISC developments may include the standard library identifiers assigned by the Library of Congress for use in MARC records[1] or the UCAS codes for universities and courses.
- CategoryBag – a list of categories defining the nature of the business. Each categorization includes the authority for that categorization. Currently defined authorities in UDDI include NAICS (North American Industry Classification System) and SIC (Standard Industry Codes) for industry classification, UNSPC (Universal Standard Products and Services) for product and service classification and GeoWeb for geographical location. Other authorities could include subject classification such as LCSH (Library of Congress Subject Headings) or Conspectus.
- DicscoveryURL – a list of URLs from which additional information can be found. This could for example link to collection level information. The first discoveryURL

[1] http://lcweb.loc.gov/cgi-bin/zgate?ACTION=INIT&FORM_HOST_PORT=/prod/www/data/z3950/loctr05.html,rs20.loc.gov,8210&CI=085311

specified is a URL to the UDDI registry so that the UDDI data can be retrieved via the HTTP protocol as well as UDDI.
- Operator – identifies the registry from which this record originates
- authorizedName – identifies the user who registered this record

In addition the publisherAssertion allows relationships between specified between organisations. Hence, a University Library could be modelled as a businessEntity, with a "is a library of" relationship with a businessEntity representing the University itself.

The businessService contains the following data:

- Name – The name of the service.
- Description – a description of the service
- CategoryBag – a list of categories defining the nature of the service, using the same structure and authorities as for the CategoryBag in the businessEntity.
- Operator – identifies the registry from which this record originates
- authorizedName – identifies the user who registered this record

In the case of a library, for instance, services might include its online catalogue, interlibrary loan ordering facilities etc.

The bindingTemplate contains technical information pertinent to that service (such as the host's internet address) and a link to a tModel, which describes the protocol involved to interact with that service. tModels form a separately maintained list of registered protocols such as telnet, Z39.50 Whois++, LDAP etc. Currently the tModels do little more than provide well-known names for protocols (such as telnet, z3950, ldap etc.). However, the tModel metadata does include a list of URLs at which additional information on that protocol can be found. Currently these link to human readable documents (the Z39.50 tModel includes a link to the Z39.50 Maintenance Agency, for example). However the ultimate plan is for the tModels to include links to machine-readable definitions, such as those specified in WSDL (see Appendix 3 – Protocol Descriptions)

The links between all these entities are maintained by giving each entity a unique identifier using the OMG's (Object Management Group) algorithm for generating GUIDs (Globally Unique Identifiers). Each entity then uses these to indicate the other entities to which it is related. UDDI also provides mechanisms to return URLs from which information from the UDDI registry can be retrieved and viewed by standard web browsers or other clients which do not support UDDI (although this does not provide any search capabilities) and to return URLs providing richer metadata about the service or business.

OAI Identify

The Opens Archive Initiative has a mechanism, Identify for retrieving descriptive information about an OAI repository. The response to an identify request is given in Figure 11.

The response consists of the following information:

- responseDate – the date and time that the OAI repository sent this response
- requestURL – the original URL sent to the repository that prompted this response

- repositoryName – the name of the repository
- baseURL – the base URL to which OAI requests should be sent
- protocolVersion – the version of the OAI protocol supported by the repository
- adminEmail – an e-mail contact for administrative purposes
- description – a detailed description for the repository

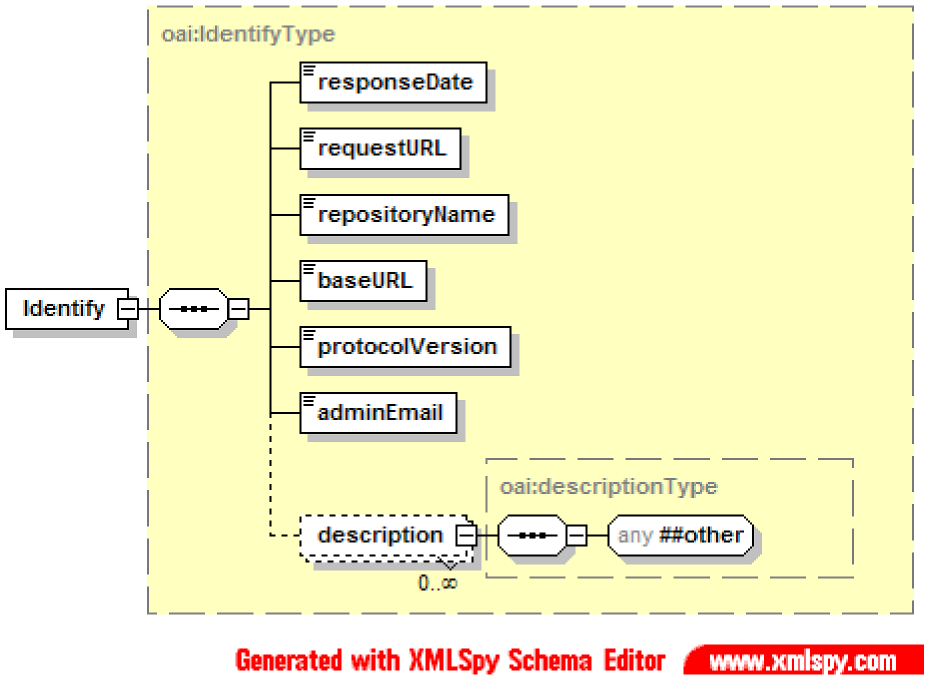

Figure 11 - OAI Identify Schema

The description element provides an extensible mechanism for more detailed descriptive information – new descriptionTypes can be defined for different types of OAI repositories. At the moment, only two descriptionType is defined by the OAI specification: for ePrint archives, the schema for this is defined in Figure 12; and that for archives that implement the OAI standard for unique record idenfiers, the schema for this is defined in Figure 13.

The data in the ePrint archive description is as follows:

- content – textual description and/or URL link to textual description of the content of the archive
- metadataPolicy – textual description and/or URL link to textual description of the metadata policy of the archive
- dataPolicy – textual description and/or URL link to textual description of the data policy of the archive
- submissionPolicy - – textual description and/or URL link to textual description of the submission policy of the archive
- comment – additional information about the archive

The OAI identifier schema in Figure 13 defines how unique record identifiers are constructed. Each identifier is of the form:

{scheme}{delimiter}{repositoryIdentifier}{delimiter}{generated record id}

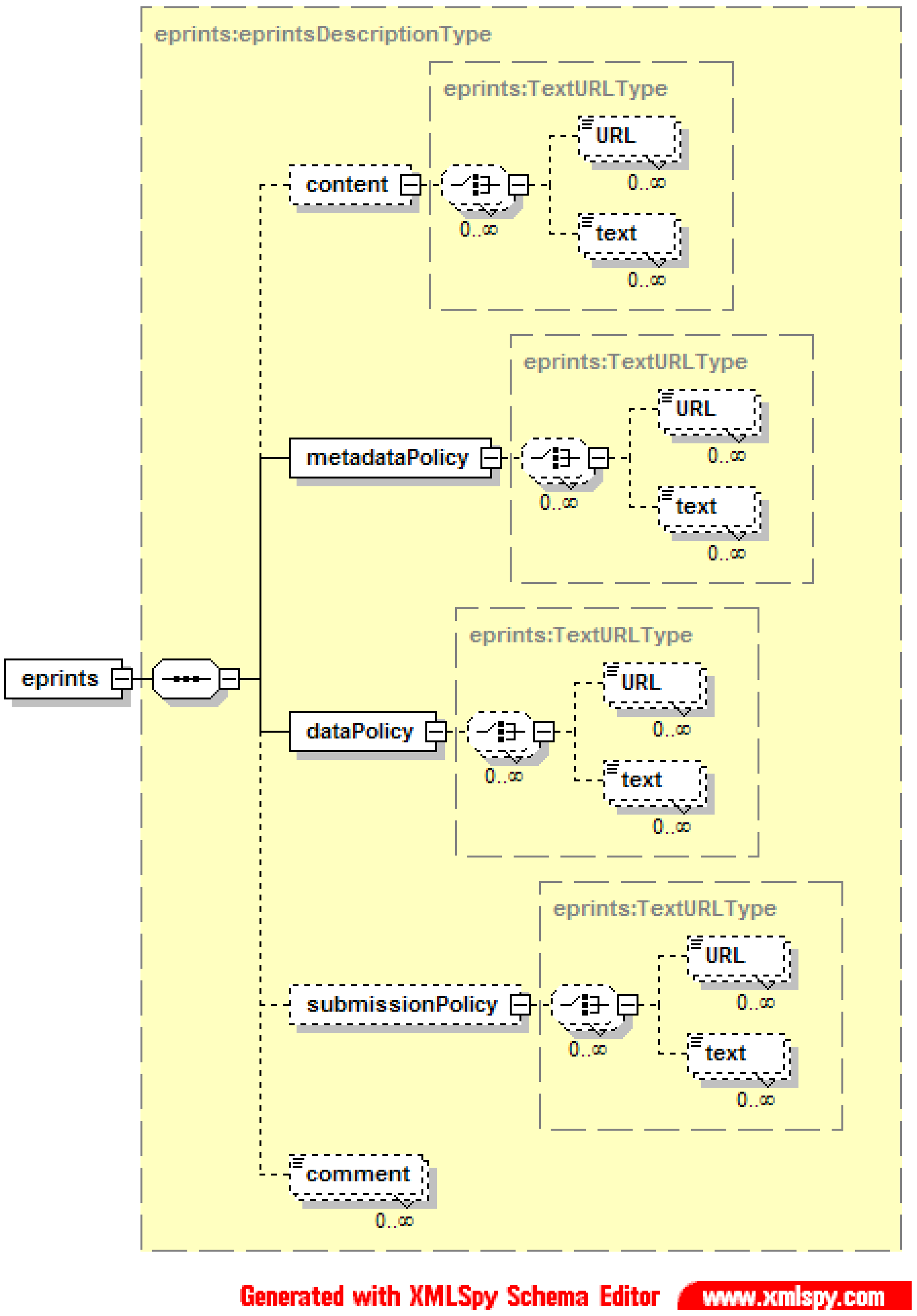

Figure 12 - ePrints descriptionType schema

A sample identifier is also specified.

The OAI Protocol also has an additional mechanism called GetRecordFormats for retrieving the different types of record formats that the repository supports.

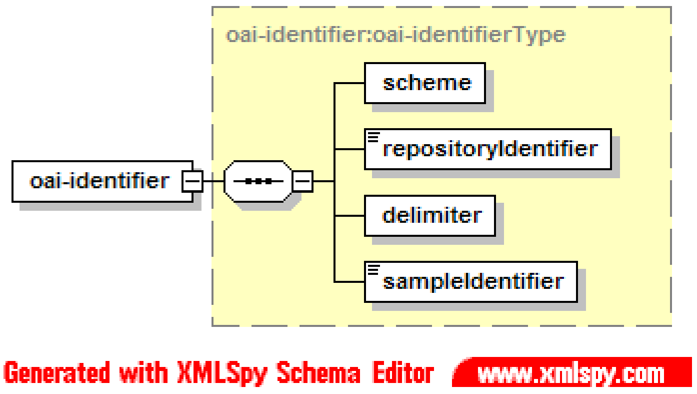
oai-identifier:oai-identifierType
oai-identifier
scheme
repositoryIdentifier
delimiter
sampleIdentifier
Generated with XMLSpy Schema Editor www.xmlspy.com

Figure 13 - OAI identifier schema

ebXML – Registry Information Model

ebXML (e-Business XML) is an OASIS lead standard acting as an umbrella for a variety of standards required for describing electronic business to business transactions. ebXML therefore provides a number of XML formats devised not only for the description and discovery of services but also for describing the processes involved in business to business transactions such as business agreements and contracts that need to be in place before any such transactions can be made.

The ebXML Registry Information Model (currently in draft) defines an object oriented framework for the types of information that may be present in an ebXML registry. Currently an ebXML registry would include the following types of object:

- Collaboration Protocol Agreement (CPA) – represents a technical agreement between two parties on how they plan to communicate with each other using a specific protocol.
- Collaboration Protocol Profile (CPP) – provides information about a Party participating in a Business transaction.
- Process – catalogues a process description document.
- Role – description of a Role in a Collaboration Protocol Profile
- ServiceInterface – an XML description of a service interface
- SoftwareComponent – catalogues a software component (e.g., an EJB or Class library).
- Transport – an XML description of a transport configuration
- UMLModel – an UML model.
- XMLSchema – An an XML schema (DTD, XML Schema, RELAX grammar, etc.).
- ExternalLink – A link to an external object
- ExternalIdentifier – An external identifier
- Association – Defines the association between objects (e.g. parent/child)
- Classification – defines a classification schema
- User – A User
- Organization – An Organization

At present there are no full implementations of ebXML systems. Whilst ebXML is fair richer in its scope it remains to be seen how it fairs in relationship to other emerging XML based

WebService description schemas (such as WSDL, UDDI etc.) which are being developed in conjunction with implementations, and which whilst initially tackling a smaller subset of the issues are encompassing other issues as they develop. At a higher level ebXML defines an object oriented model of a business to business registry which many of the WebService description schemas do conform in many aspects. There are hence plans within ebXML to at least interface with WebService technologies such as UDDI, WSDL. The lack of implementations also means that as regards its Registry Model offers little more than an object oriented view of a service registry (e.g. it does not offer any classification schemes per se, merely a way of modelling a registry which supports external schemes).

GRID

GRID applications require a service discovery mechanism in order to locate GRID nodes on which jobs maybe executed. The current GRID implementations such as GLOBUS use an information service such MDS (Meta-Directory Service) which implements GRIS (GRID Information Service). This currently works via LDAP as both its schema and to provide both its GRIP (GRID Information Protocol) for locating services and its GRRP (GRID Registration Protocol) for publishing services.

In its current form the schema used for describing services are limited to describing computational resources, e.g. specify information such as processing power, available memory, disk space etc. and as such are not suited to more general service discovery. However, expanding this is currently an area of interest from the GRID community. There is, however, a movement towards bringing GRID technologies and WebService technologies closer together (with a road map given in the Open GRID Service Architecture). It is therefore likely that the GRID will embrace existing WebService schemas for these purposes rather than invent its own. It is also possible that the GRID would use UDDI as its GRIP and GRRP provider over LDAP at some point.

Broker Architectures (CORBA, JINI, JXTA etc.)

A number of brokering or peer service frameworks such as CORBA (Common Object Request Broker Architecture), JINI, JXTA (JuXTApose) etc. also offer service publishing and discovery services. However, such services only allow searching on the Application Programming Interfaces available and simple (user defined) key/value pairs. They are really targeted at providing discovery mechanisms within particular applications rather than providing a general purpose service discovery mechanism. Such discovery services are not accessible outside of the specific framework, i.e. only CORBA applications can use the CORBA discovery service etc.

Functional Crosswalks

Table 1gives an overview of the different service location metadata available by the different service description schemas. The most important information is

- Title
- Keyword – to enable searching for services
- Cross-reference – to enable browsing and location of related services
- Currency of record – for management of data
- Provenance of record – for management of data and also resolving disputes over ownership

- Control Identifier – for referencing services at a later date (essential for local caching of service descriptions or bookmarking). UDDI also offers a persistent URL to retrieve the record so that clients which do not support the protocol can still access UDDI descriptions

Of interest is the Explain addition of Icon, Welcome Message and New for branding the service. In order for Content Syndication to be successful the JISC IE would need some similar mechanism in order for Content Providers to remain visible and hence encourage business scenarios with commercial suppliers.

It should be noted that all of these schemas can be extensible. UDDI and OAI Identify have specific mechanisms for allowing third party extensions to the description (and such extensions could later be recommended for inclusion in the official standard). Explain and GILS could be extended although this would need approval of the appropriate standards committees. Explain-Lite is currently a more informal specification and a private JISC IE extension could be applicable, likewise with Explain--. WebClarity Resource Registry Schema is a commercial development and extensions would have to be suggested to Sea Change.

	Explain	**Explain-Lite**	**Explain--**	**WebClarity**	**GILS**	**UDDI**	**OAI Identify**
Title	Name; Nicknames		X		X	X	X
Contact	General; Payment	X	X	General	General; Distributor	*Extensible types*	Admin e-mail
Sources of Data					X	X	
Description	General; Technical; Restrictions; Disclaimer; Copyright; Update/submission information		Description; Author; History; Restrictions; Extent	General; Technical	Abstract; Methodology; Restrictions; Supplemental; Purpose	X	*Extensible description mechanism*
Availability				X	X		
Charges	X				X		
Hours of Operation	General; Best time to access			X	X		
Cross Reference	Associated Databases; Sub-Databases	X		X	X	*UDDI Version 2.0 Only*	
Keywords	X		Subjects		*Both Controlled and Uncontrolled Vocabularies*	X	
Language of Service	X		X	X	X		
Spatial Domain	X			X	X		
Date Coverage	X			X	X		
Currency of Data	Update Cycle; Last Update				Date of Publication		
Language of description	*Multiple language support*				X	*Multiple language support*	
Currency of			Date created,	Date Created;	Source; Last modified;	X	

	Explain	Explain-Lite	Explain--	WebClarity	GILS	UDDI	OAI Identify
Description			Date aggregated	Date Verified; Date modified	Next Review		
Provenance of Description			Aggregated From	X	Last author; Track derivations	Last author	
Control Identifier					X	UUID; Discovery URL	
News	X						
Icon	X						
Welcome Message	X						

Table 1 - Overview of descriptive metadata

Table 2 shows an overview of the service access metadata capabilities of the various schema to describe the various service protocols which would be present in the JISC IE. Explain, Explain-Lite, Explain-- and Web-Clarity are designed for describing Z39.50 configuration only. All three could be extended through the appropriate standards committees or in the case of WebClarity by making suggestions to Sea Change. Sea Change have plans to add support for other protocols but this is not in the current specification. Explain-- have acknowledged that Explain-- might be applicable to non-Z39.50 service but limit their interested to similar services which would be describable via Explain-- without major modifications (such as ZiNG SRW and SRU) . GILS has little access metadata, being more designed for location metadata, although this could be added. OAI Identify is designed only for describing OAI repositories and its access metadata is merely the base URL for sending requests to an OAI server. It could be extended by adding new description types for different protocols but this is distorting the specification somewhat. UDDI provides a mechanism for describing arbitrary protocols. It would be necessary to define appropriate extensions to the bindingTemplate and suitable tModels for Z39.50 configurations (e.g. tModels for different profiles such as the Bath Profile) – Explain-Lite or Explain-- could be appropriate in this capacity.

	Explain	Explain-Lite	Explain--	WebClarity	GILS	UDDI	OAI Identify
Z39.50	X	X	X	X		X	
Single entry HTTP Services (e.g. FirstSearch, RSS, OCS)					X	X	
Multiple entry HTTP Services (e.g. OpenURL)						X	
WebServices (e.g. OAI, ZiNG)			ZiNG SRW/SRU			X	(OAI Only)

Table 2 – Application of Schemas for describing access to different protocols

Service Description Registries

Z39.50

Being a generic search/retrieve protocol, it is not surprising that Z39.50 can lend itself to searching a registry of service descriptions in the same way that it can lend itself to searching any other database. In order to phrase suitable queries for locating services, it is necessary to have an attribute set which includes the appropriate search points required. Many of these (such as title, keyword etc.) are providing in the bib-1 attribute set. Other search points more applicable to service location (such as spatial location) are available in the GILS attribute set (which is also included in the Bib-1 set). Search points particular to the Z39.50 Explain database can be found in the Exp-1 attribute set designed explicitly for Explain. Also relevant is the Explain-- attribute set which defines a functional subset of the Exp-1 set.

The Explain database "IR-Explain-1" is not suitable for a service registry since this is designed just to describe the local Z39.50 server. However, it would be possible to run a more generic Explain database which described multiple systems. This is the proposal behind Explain—'s Explain---1 database which is intended to be a database of Z39.50 servers, not just a description of the databases on the local server. Alternatively it would be possible to run a GILS registry. The US Government runs such a registry for its own information services. It is also possible to provide Z39.50 access to other registries which maybe accessible via other protocols. For instance the WebClarity Resource Registry also offers Z39.50 access as well as access via other protocols.

Z39.50 could also be used to maintain a registry by using the Z39.50 Extended Service "Update". However, there are few implementations of this service available. Commercial available implementations tend to be tied to particular products and tend not to be interoperable. There are also few clients which support this service, and most which do tend to be library based and therefore designed for MARC databases.

UDDI

A client application communicates with the UDDI registry via SOAP. SOAP is a Microsoft proposed standard for allowing client and server applications to communicate using XML messages typically over HTTP. One of the rationales behind SOAP is to enable rapid development of clients and servers. This is vindicated by the rapid development of UDDI software within the first few months after its specification. The World Wide Web Consortium (W3C) are working on a successor to SOAP called XMLP which will include the functionality of SOAP and similar proposals but will be a vendor independent standard. As XMLP is heavily based on SOAP, UDDI will adopt XMLP when it is ratified by the W3C.

The UDDI communications model includes support for the following four main types of interaction:

- Browsing – This allows the discovery of businesses and services given a search template such as business name or the type of service required. There are four functions currently defined in this category: find_business, find_service, find_binding, find_tModel. Each of these functions take either a template businessEntity, businessService, bindingTemplate or tModel respectively using the metadata structures outlined in the previous section and return a list of matching businessEntities, businessServices, bindingTemplates or tModels as appropriate. The query language is currently limited with little support for true Boolean queries although this is being addressed in version 3.0 of the specification.

- Retrieval ("Drill Down") – This allows the retrieval of details of a business, service or protocol given that its unique identifier is known (e.g. by using the browsing functions described above). This would typically be used by mediator software once a service had been located.. There are five functions currently defined in this category: get_businessDetail, get_businessDetailExt, get_serviceDetail, get_tModelDetail, get_bindingDetail. These take a GUID and return a detailed businessEntity, businessService, bindingTemplate or tModel as appropriate. The get_businessDetailExt is used to return additional "private" information not defined within the UDDI specification. Typically Browsing functions are used to return a list of brief information of businesses, services, etc. which includes the GUIDs and brief descriptions. The Retrieval functions are then used to retrieve the full information for the given GUIDs.

- Publishing – These allow the creation, deletion and modification of details stored in the UDDI registry. Typically these would be used by an organisation to maintain its own details. Some application server software may automatically register services using these functions. There are currently eight functions in this category: save_business, save_service, save_tModel, save_binding for saving information (businessEntities, businessServices, tModels and bindingTemplates respectively) and delete_business, delete_service, delete_tModel, delete_binding, for removing previously saved ones from the registry.

- Authentication – These are used alongside the publication functions to ensure that only authorized users can maintain an organisation's entries. There are currently three functions defined in this category: get_authToken is used to authenticate with a server and receive a token which is used when using the other functions – in effect to log on to the server, discard_authToken which is effectively use to log out of the server and get_registeredInfo to retrieve additional information about the current user.

The protocol is a SOAP based protocol which allows rapid development of both client and server applications. There is a growing number of client and server applications as well as toolkits for most major platforms.

A component of the UDDI initiative is to provide a global public registry, so that there is a single well known point of contact for locating services. This is currently being provided by Microsoft, IBM and Hewlett-Packard. The global registry is multi-nodal to provide redundancy and avoid network latency (by having nodes scattered world-wide). The mechanism for replicating data across the nodes has been made part of the UDDI specification as of version 2.0.

The existence of the global UDDI registry does not prevent the existence of private registries – it may be more appropriate for the JISC IE to run its own UDDI registry although some data could be replicated between the private registry and the global one. Details of the relationship between public and private registries and migrating records between them is being ratified in the Version 3.0 specification due to release in 2002.

ebXML – Registry Services Specification

As mentioned above, ebXML (e-Business XML) is an OASIS lead standard acting as an umbrella for a variety of standards required for describing electronic business to business transactions, so provides XML schemas for the description and discovery of services but also for the description of the processes involved in business to business transactions e.g. business agreements and contracts.

The Registry Services Specification defines the Application Programming Interface (API) needed to interact with a registry of ebXML objects (including descriptions of users, organisations, services, protocols, business practices and contracts etc.) for publishing and discovering information. It also defines some of the behaviours of a registry in terms of the life cycle of objects etc.

Whilst ebXML offers a number of other services and features beyond WebService based registries such as UDDI (for example allowing explicit indication of the life span of an object in the registry, explicit access via SQL queries as well as its own defined query interface etc.), there are yet no real implementations of the ebXML Registry Service. This is partly due to its complexity but mainly due to a differing development philosophy. The developers of UDDI are implementing the standard as the standard is being developed with close feedback between implementations and the standard. ebXML however is being developed as a specification first, implementations being developed once the standard as been finalised.

The relationship between ebXML and UDDI is currently volatile (wavering from embracing UDDI within ebXML to developing its own alternative). However, it seems likely that there will be interfaces between ebXML and UDDI.

OAI

Like Z39.50 Explain, the original intent of the OAI-Identify mechanism is to describe the local repository only. It is not really intended for the discovery of OAI repositories. Since OAI has no search capabilities so it would not be appropriate for a general service registry.

OAI's strength is in allowing local indexes to maintain data from multiple repositories. As such if a local copy of a registry (accessed using another protocol such as Z39.50 or UDDI) were required it could be an appropriate mechanism for maintaining the local copy. However UDDI has its own mechanisms for replication (in the Version 2 specification) and in Version 3 is looking at subscription based services for local registries. Z39.50 has an Extended Service – Export – which could perform a similar need (although this is not well implemented).

LDAP

The LDAP protocol (Lightweight Directory Access Protocol) is a protocol for querying and maintaining X.500 based directories. Such directories are primarily aimed at identifying resources within organisations (such as users, computers, printers, servers etc.). However, LDAP is not aimed at service description per se. Its primary use has been for use managing networks such as by Novell Netware and Microsoft Active Directory. It has also been used for providing e-mail white page style directories.

It has a number of limitations which make it less than ideal for service level descriptions:

- It has a limited schema model primarily designed for resource description rather than service description.
- It has limited search capabilities based on keyname/value pairs and no support for Boolean operators.
- It is aimed at providing a local resource, as such it is not scaleable. In particular, it does not provide for distributed models nor does it allow authorised users to register new resources (it works on the principle of administrators maintaining the content).

It may be useful for a service registry to provide LDAP access for querying for compatibility reasons, but LDAP would not be suitable as the primary protocol for managing a service registry.

WebClarity Resource Registry

The WebClarity Resource Registry is a commercial development of Sea Change Corporation, who produce a number of Z39.50 clients for Windows and web browsers. They have built up a large collection of the configuration data for a variety of world-wide Z39.50 servers which they ship with their clients to offer pre-configured access a wide variety of sources. The WebClarity Resource Registry provides a globally accessible database of this information which is currently accessible via a Web Interface and also through a proprietary SOAP protocol which has also been implemented in the new versions of their client software. The web interface can also be used by registered users to maintain the details of their own Z39.50 servers. Sea Change currently plan to provide access to the Resource Registry via Z39.50, LDAP and UDDI. However access to the registry is on a subscription basis.

Products

WebClarity

(http://www.webclarity.info/registry.html)

As mentioned Sea Change maintain a subscription based service containing registered Z39.50 servers which is searchable via the web and a SOAP API with plans for Z39.50, UDDI and LDAP access. The software that they use for the global Resource Registry is also available as a standalone product for local use. At present the WebClarity product only uses a proprietary SOAP based protocol accessibly from Sea Change's own Z39.50 clients (such as BookWhere), so would only be viable when the other planned protocols are supported. Also at present WebClarity is designed to describe Z39.50 servers only.

OAI

The OAI Protocol is aimed solely at harvesting metadata deliberately leaving open what use is subsequently made of any harvested metadata. Most software which supports the OAI protocol currently is designed for navigating e-print archives, and so is not applicable to service registries. However, the OAI protocol is designed to be simple to implement and therefore should be fairly straight forward to add to an existing service registry implementation.

UDDI

Global Registry

UDDI consists of both a protocol and schema definition and an initiative to run a global public registry. Currently Microsoft, IBM and Hewlett-Packard are all signed up to run nodes of the distributed public UDDI registry. Anyone can sign up to register businesses and services, although all additions to the registry can be tracked. The intention is to let social pressures encourage business to add entries into the registry (so that they can participate in the WebService arena) and also social pressures to ensure that businesses police their own representation in the registry (e.g. if someone masquerades as a business) rather than centralised policing (in effect in a very similar manner in which the web presence of businesses work). To assist any disputes all entries in the registry can be tracked back to the person who registered them and the node of the registry where the records were created.

A number of software companies offer client tools for accessing an UDDI registry, and many also offer UDDI registry servers for running private UDDI registries.

Microsoft

(http://msdn.microsoft.com)
Microsoft offer a number of tools within their .NET suite of software:

- Visual Studio .NET has the ability to search a UDDI registry for WebServices, and then to been the COM object for communicating with the server from a client application from a WSDL description. It also has the ability to generate WSDL descriptions from code. It currently supports UDDI version 2.0.
- UDDI SDK – is a small download implementing UDDI version 2.0 for Visual Studio 6. It includes a sample visual basic application for navigating a UDDI registry
- WebServices for Microsoft Office – is a optional download for Microsoft Office which allows the discovery of WebServices from a UDDI registry and subsequent use of those services from within the Visual Basic macro language in Microsoft Office products.
- UDDI Registry – is a beta implementation of a version 1.0 UDDI registry. This runs on a Microsoft SQL Database (a runtime version of MSSQL is included for use on a machine on which MSSQL is not previously installed). It also includes a web interface for searching and publishing UDDI records. Currently this requires Microsoft .NET Server beta 3 to run. Authorisation to add records to the UDDI registry is based on Windows NT authentication, hence it is not possible to register as a publisher through the web interface and is really intended for intranet use where strict control over who can publish to the registry is required.

IBM

(http://www.ibm.com/developerWorks)
IBM have a number of UDDI based developments, primarily under their WebSphere suite of products. These include:

- Web Services Toolkit – a collection of tools for developing an deploying java based Web Services. These include tools for handling SOAP messaging (developed in conjunction with the Apache community), a UDDI client library for java, and a java library for manipulating and parsing WSDL documents.

- Business Explorer for Web Services – a client for navigating UDDI registries
- WebSphere Studio Application Developer (currently in beta) – A complete Java development environment which include tools for generating WSDL and UDDI descriptions, compiling client JavaBeans which interact with the WebServices from WSDL and UDDI descriptions, and tools deploying, running and debugging WebServices running on IBMs WebSphere Application Server.
- Private UDDI Registry (currently in preview) – A private UDDI registry implementing version 1.0 of the standard. This requires DB2 to run. It includes a web interface which can either allow anyone to register to be a publisher, or can be locked to just centrally authorised publishers.
- Web Services Gateway, Web Services Hosting – Various additions to the IBM WebSphere Application Server range which support the running of WebServices, dynamic publishing of WebServices to UDDI when a server is deployed and gateways for controlling access to WebServices through firewalls etc.

jUDDI

(http://www.juddi.org)
jUDDI is an OpenSource Java implementation of both a UDDI client and a UDDI server, which currently supports version 2.0 of the standard. The server supports different backends for databases currently including a simple file structure, mySQL, PostGres, Oracle and Access. It is primarily developed by Bowman. It does not offer a web interface for accessing or publishing to the registry.

JAXR/Java WebServices ToolKit

(http://java.sun.com/webservices/downloads/webservicespack.html)
JAXR (Java extenstion API for Accessing Registries) is a Sun approved extension to Java offering a generic application programming interface for accessing registries. Registered providers translate the generic API to specific protocols (such as UDDI, ebXML, LDAP etc.). It is currently in a release candidate form and included as part of the Java WebService ToolKit. The ToolKit also includes a UDDI version 1.0 registry server.

Borland WebServices ToolKit

(http://www.borland.com/jbuilder/webservices/)
Borland offer an additional add-on for their JBuilder Java Development environment which provides tools for generating WSDL descriptions, building Java Bean clients to WebServices from WSDL description and locating services and WSDL descriptions from a UDDI registry.

Systinet WASP

(http://www.systinet.com/)
Systinet (previously called Idoox) offer a suite of tools called WASP. This consists of the following:

- WASP Developer – is a development environment for WebServices with support for WSDL and UDDI.

- WASP Server – is a server for deploying and running WebServices. It comes it two versions. Lite is free for commercial use, whilst Advanced is free for development and testing only.
- WASP UDDI – is a UDDI server supporting UDDI version 2.0. It also supports a web interface for administering the registry as well as searching and publishing. It can allow users to register to be able to publish either automatically or after centralised approval. It is free for development and testing only.

Z39.50

There are a number of Z39.50 servers on the market – mostly as additional components to library systems. Very few servers currently support Explain, notable exceptions being Cheshire (http://cheshire.lib.berkeley.edu) and Index Data's servers (http://www.indexdata.dk). Both these, however, primarily support the Explain-IR-1 database which is meant for describing databases on the local server rather than servers in general.

In terms of support for GILS, most generic Z39.50 toolkits should be able to support GILS records and the GILS attribute set for searching. However, the main GILS implementation is the OpenSource ASF (Advanced SearchFramework) software (http://asf.gils.net).

Recommendations

Overall UDDI has advantages over the other technologies available, primarily since it has been designed to handle service descriptions for multiple protocols whereas many of the others have been designed for a particular protocol or purpose only. Also UDDI specified both the schema and the registry protocol, whereas others simple specify a schema for describing a local server only.

The main alternative at present is the WebClarity Resource Registry. This has the advantage of potentiall offering a number of different protocols for interaction (currently SOAP, Web and Z39.50 with UDDI and LDAP planned). This also has a large number of Z39.50 targets already in its public registry. However, it currently can only describe Z39.50 targets (and other protocols are planned). The major obstacle to WebClarity is that this is a commercial product from a single supplier.

UDDI is currently a de facto standard but with support from a large proportion of the industry with about 200 members advising its development. It is planned to go to a standards body (such as ISO or OASIS) in 2002. The growing support is leading to a number of emerging toolkits and with very good current support on both Microsoft and Java platforms.

UDDI has a mechanism for describing different protocols: a serviceBinding identifies the protocol used via a tModel and also links to specific configuration information. However the exact tModels and serviceBinding descriptions for the various services in the JISC IE will need to be worked out. Some of the other specific protocol service schemas could be used here. For example the serviceBinding to a Z39.50 server could contain configuration data in Explain-Lite, or a serviceBinding to a OAI respository could contain data in OAI-Identify. This information could be harvested automatically from Z39.50 Servers which support Explain/Explain-Lite (or via Explain generation robots such as IndexData ZSpy or GateZed) or from OAI archives.

There are a few areas in the location metadata where UDDI has deficiencies, namely branding information (such as icons/logos or messages to display in clients) and detailed provenance and currency information about the record (UDDI has some but fairly limited when compared to WebClarity and GILS). However, UDDI has a mechanism for linking to extended information via discoveryURLs where such information could be found. The discoverURL mechanism can also be used for linking with other descriptive/location services such as Collection Level Description registries or other search interfaces providing navigational links to reaources.

One issue to be resolved would be whether the JISC IE should have a private UDDI registry or use the Global Registry (possibly using a tModel to indicate a JISC IE approved service). A private registry would provide more control although would involve issues as to how and if to replicate with the Global UDDI registry. A JISC IE private registry would need multiple nodes fore resilience. Local institutions may also wish to run a local mirror node. The UDDI specification includes details on the replication service for such multiple node configurations.

Another issue would be whether to provide gateways into the UDDI registry for other protocols. A web interface would, of course, be required both for allowing service providers to maintain their own entries, but also to allow casual browsing of JISC IE resources. Most UDDI products provide this facility. Other gateways may include Z39.50 to UDDI and LDAP to UDDI (ala the WebClarity product).

Appendix 1 – Ownership of Service Descriptions

An outstanding issue with UDDI Version 2.0 is that of the ownership of descriptive records. At present, anyone can register as a publisher and create records in the public UDDI registry. There are no central controls or policing as the quality of the records, or that the publisher of the record has the authority to publish the business or service. The intent is that social pressures will control most of this – organisations will wish to publish themselves in order to participate in the WebService arena and will also check that no-one is misrepresenting them. In case of disputes, all records returned from the registry indicate the node at which they were created and the user who created them. In effect, resembling the way in which organisations police web sites. One result of this is that various organisations that maintain deirectories (of businesses, or public/government agencies) have published their directories into UDDI, although they themselves do not officially represent the organisations whose descriptions they have published.

There is also the issue of trust – that the service advertised is legitimate and comes from the agency it claims. In UDDI Version 2.0 there are no controls on the public registry, however, this does resemble the situation with services offered through websites. However, UDDI Version 3.0 is introducing the ability to sign UDDI registries using certificates so that some degree of trust can be placed in UDDI that the business advertising the services are who they claim to be.

Neither of these issues necessarily apply to private UDDI registries where authorisation to publish records can be tightly controlled by the organisation running the registry, except, of course, in the case where the registry is populated with records accessed from the public UDDI registry.

Appendix 2 – Collection Level Descriptions

The general overview of Collection Level Descriptions is show in Figure 14. There is some similarity with the UDDI model in that if we simplify Collector, Owner and Administrator into a single entity (namely that of an Agent), Agents roughly map to BusinessEntities, Collections to BusinessServices and Locations to BindingTemplates.

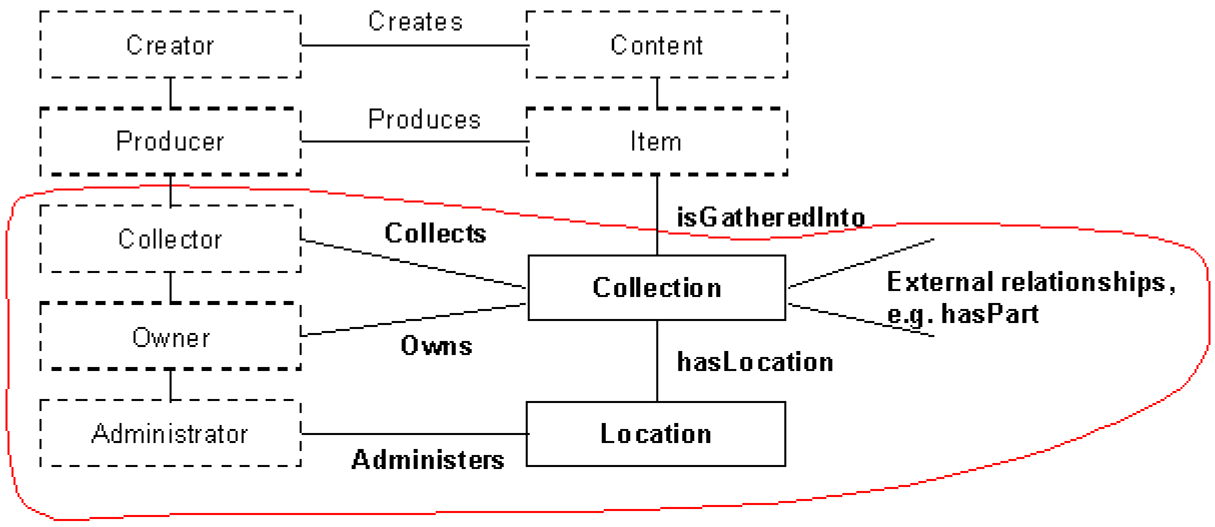

Figure 14 - Collection Level Description Overview

In general there is not a simple relationship between Collection Level Descriptions and Service Level Descriptions. For example, a BusinessService can only have one owning BusinessEntity, whilst a Collection has two Agents (with differing roles); and a bindingTemplate belongs to the same BusinessEntity as its owning Service whilst a Location can belong to a different Agent (with the role of Adminitrator) than its owning Collection. In general there are other issues with a mapping of Collection Level Description and Service Level Descriptions:

- A Service may offer access to a number of different collections (e.g. OCLC FirstSearch)
- A collection may be spread across a number of different services (e.g. an archive may be split amongst different benefactees)
- A collection may provided through multiple services run by third parties.
- Collections may form a hierarchy of sub-collections, whereas most service level descriptions do not support an hierarchy. It is not necessarily the case that a sub-collection would be accessible through the same service as its parent (e.g. COPAC)
- A collection may not be accessible through any electronic service (e.g. uncatalogued resources)
- Service Level Descriptions may describe other services not associated with collections (such as authentication services, user profiling services, data manipulation services, learning resources etc.)

One solution to this problem would be to have different registries for Service and Collection Level Descriptions. However, despite the differences, there is a high level of overlap both in the descriptive metadata and it the entities being described (many of the the same organisations would feature in both registries). An alternative solution would therefore be to derive a combined schema – as overview how how such a schema may look is given in Figure 15. In this case the registry would act as both a Collection Level Description registry and a Service Level Description registry. However, although it would use a unified database, it would be performing two logically distinct roles.

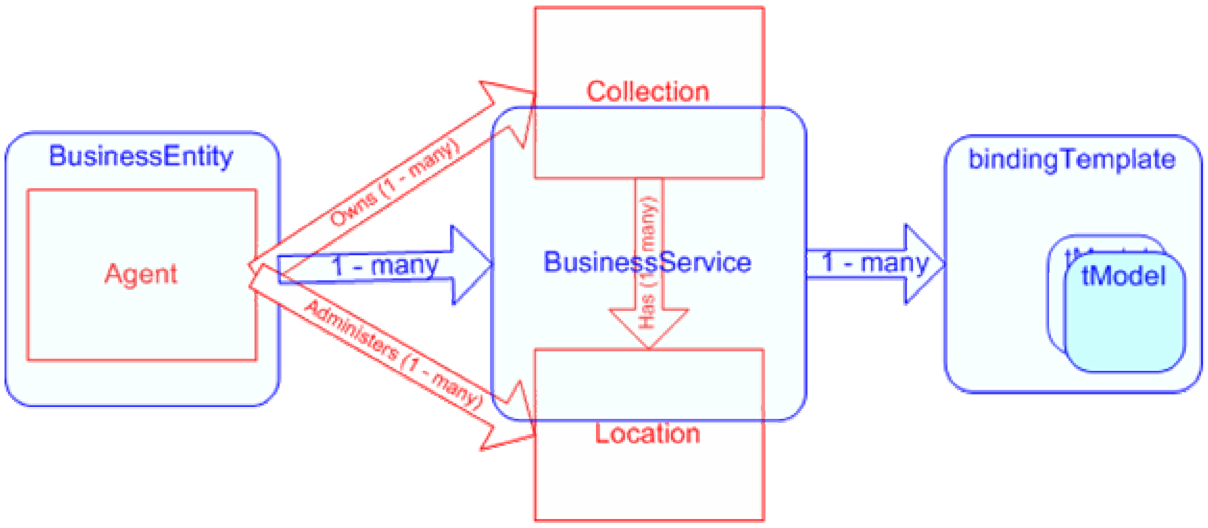

Figure 15 - Overlay of UDDI and Collection Level Schemas

In any case, there needs to be a means of navigating between the two. UDDI offers mechanisms for this via the DiscoveryURL. Any service is automatically associated a persistent URL which can be used to retrieve the UDDI service description. Such a DiscoveryURL could be included in the description of a collection which is associated with that service, thus allowing navigation from a collection description to its relevant services. An UDDI service description can also have other DiscoveryURLs giving further information. Given that collection level description has a similar mechanism to UDDI for generating a persistent URL from which the collection level description can be retrieved, the UDDI record can then contain DiscoveryURLS pointing to relevant collections to that service. A similar mechanism can be used to navigate between organisations in collection level descriptions and businesses in UDDI service level descriptions.

Appendix 3 – Protocol Descriptions

WSDL

A proposed language for such machine-readable descriptions is the Web Services Description Language (WSDL) from IBM and Microsoft. In many aspects this performs the same role as ASN.1 (Abstract Syntax Notation) or IDL (Interface Definition Language) in that it is a language for describing data structures, client-server programming interfaces and communications protocols in a machine-readable way. It differs from ASN.1 and IDL in that it is an XML application. Like ASN.1 and IDL, the code for the client and server interfaces can be generated automatically, thus simplifying the task of the developer attempting to create client and server software which use the protocol defined. There are already development tools which can do this given WSDL definitions obtained from a UDDI registry. These are described below in the sections on Products and Developments. Future generation tools may also be able to dynamically adapt to work with new protocols having obtained the WSDL definition via UDDI.

A WSDL description consists of five sections:

- **Types** – these specify the data structures used in messages between a client and a server. WSDL supports the SOAP defined subset of XML Schema as a type description language.
- **Messages** – this combines the types in messages passed between client and server.
- **Operations** – this combines the Messages into actual interations that can be made on the server. Typically they would match a request message with the corresponding response message i.e. specify the Z39.50 services. WSDL supports four different types: One-way Operation (client to server only), Request-response Operation (client sends request, server responds), Solicit-response Operation (server sends request, client responds) and Notification Operation (server to client only).
- **Binding** – this binds the Port Types onto an actual transport layer such as SOAP, HTTP, SMTP or other transports. Mechanisms for handling state may need to be specified here (as in the case for SOAP) if not explicitly handled by the transport layer (as would be the case with HTTP which uses cookies for maintaining state)
- **Services** – This describes an actual implementation of a Binding giving server details etc.

Examples

The following show example WSDL descriptions of Z39.50, Open Archives Initiative protocol and ZiNG. These are all works in progress and the specifications here should be regarded as drafts.

Z39.50

The following gives a complete of Z39.50 Service Definition with the exception of the Segmentation service. Segmentation is difficult to describe in WSDL due to the need for a single request to a server to return multiple responses from the server. However, the need for such a service where the transport layer does not do this automatically and transparently is debatable.

Types

The types for Z39.50 can be specified using XSD derived from the ASN.1 and XER. It should be possible to generate the XSD automatically thus ensuring the ability for new features of Z39.50 to be transferred into this Service Definition easily.

The current XSD for Z39.50 is available at http://asf.gils.net/xer/ez.xsd

Messages

The WSDL extract below gives all the Request and Response messages defined in Z39.50 and maps this onto the underlying types defined in the XSD above. The exception here is the messages for Segmentation for reasons given below.

```
<definitions targetNamespace="urn:ez3950"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/so
ap/" xmlns:xer="http://asf.gils.net/xer">

  <import namespace="http://asf.gils.net/xer"
location="http://asf.gils.net/xer/ez.xsd"/>


  <message name="initRequest">
      <part type="xer:initRequest"
name="initRequest"/>
  </message>
  <message name="initResponse">
      <part type="xer:initResponse"
name="initResponse"/>
  </message>
  <message name="searchRequest">
      <part type="xer:searchRequest"
name="searchRequest"/>
  </message>
  <message name="searchResponse">
      <part type="xer:searchResponse"
name="searchResponse"/>
  </message>
  <message name="presentRequest">
      <part type="xer:presentRequest"
name="presentRequest"/>
  </message>
  <message name="presentResponse">
      <part type="xer:presentResponse"
name="presentResponse"/>
  </message>
  <message name="sortRequest">
      <part type="xer:sortequest"
name="sortRequest"/>
  </message>
  <message name="sortResponse">
      <part type="xer:sortResponse"
name="sortResponse"/>
  </message>
  <message name="scanRequest">
      <part type="xer:sortRequest"
name="scanRequest"/>
  </message>
  <message name="scanResponse">
      <part type="xer:scanResponse"
name="scanResponse"/>
  </message>
  <message name="deleteRequest">
      <part type="xer:scanRequest"
name="deleteRequest"/>
  </message>
  <message name="deleteResponse">
      <part type="xer:scanResponse"
name="deleteResponse"/>
  </message>
  <message name="accessControlRequest">
      <part type="xer:accessControlRequest"
name="accessControlRequest"/>
  </message>
  <message name="accessControlResponse">
      <part type="xer:accessControlResponse"
name="accessControlResponse"/>
  </message>
  <message
name="triggerResourceControlRequest">
      <part
type="xer:triggerResourceControlRequest"
name="triggerResourceControlRequest"/>
  </message>
  <message name="resourceControlRequest">
      <part type="xer:resourceControlRequest"
name="resourceControlRequest"/>
  </message>
  <message name="resourceControlResponse">
      <part type="xer:resourceControlResponse"
name="resourceControlResponse"/>
  </message>
  <message name="resourceReportRequest">
      <part type="xer:resourceReportRequest"
name="resourceReportRequest"/>
  </message>
  <message name="resourceReportResponse">
      <part type="xer:resourceReportResponse"
name="resourceReportResponse"/>
  </message>
  <message name="extendedServicesRequest">
      <part type="xer:extendedServicesRequest"
name="extendedServicesRequest"/>
  </message>
  <message name="extendedServicesResponse">
      <part
type="xer:extendedServicesResponse"
name="extendedServicesResponse"/>
  </message>
  <message name="close">
      <part type="xer:close" name="close"/>
  </message>
```

Operations

The WSDL extract below groups the messages into the appropriate services (known in WSDL as Operations, grouped under a Port Type). WSDL supports four types of Operation (One-way, Request-response, Solicit-reponse and Notification) of which three are used as indicated (Request-response, Solicit-reponse and Notification).

```
<portType name="ez3950PortTypes">

<!-- Request-response Operations (client
initiated) -->
      <operation name="init">
          <input message="initRequest"/>
          <output message="initResponse"/>
      </operation>
      <operation name="search">
          <input message="searchRequest"/>
          <output message="searchResponse"/>
      </operation>
      <operation name="present">
          <input message="presentRequest"/>
          <output message="presentResponse"/>
      </operation>
      <operation name="sort">
          <input message="sortRequest"/>
          <output message="sortResponse"/>
      </operation>
      <operation name="scan">
          <input message="scanRequest"/>
          <output message="scanResponse"/>
      </operation>
      <operation name="delete">
          <input message="deleteRequest"/>
          <output message="deleteResponse"/>
      </operation>
      <operation name="resourceReport">
          <input
message="resourceReportRequest"/>
          <output
message="resourceReportResponse"/>
      </operation>
      <operation name="extendedServices">
          <input
message="extendedServicesRequest"/>
          <output
message="extendedServicesResponse"/>
      </operation>
      <operation name="close">
          <output message="close"/>
          <input message="close"/>
      </operation>

<!-- Solicit-response Operation (Server
initiated) -->
      <operation name="accessControl">
          <output
message="accessControlResponse"/>
          <input
message="accessControlRequest"/>
      </operation>
      <operation name="resourceControl">
          <output
message="resourceControlResponse"/>
          <input
message="resourceControlRequest"/>
      </operation>
      <operation name="close">
          <output message="close"/>
          <input message="close"/>
      </operation>

<!-- Notification Operations (Server
initiated)-->
      <operation name="segment">
          <output message="segmentRequest"/>
      </operation>

<!-- One-way Operations (Client initiated) -->
      <operation
name="triggerResourceControl">
          <input
message="triggerResourceControlRequest"/>
      </operation>

<!-- Note: Segmentation control is not
supported (do we need it?) -->
  </portType>
</definitions>
```

This describes the full Z39.50 protocol with the exception of Segmentation. Bindings to particular transport layers are given below. State is either handled implicitly by the specific layer (e.g. cookies in the case of HTTP) or specified in the binding definition (e.g. using headers in the case of SOAP)

Transport Bindings for Z39.50

The following give WSDL descriptions on how the above service definition can be bound to a particular transport protocol. The binding may only bind some of the services

BER over TCP/IP

In theory, a binding definition could be defined for Z39.50 encoded using Z39.50 over BER. This would require a suitable WSDL binding namespace being defined for BER.

XER over SOAP

The binging of the above Service Definition using XER over SOAP is given below. The soap:header is used to carry session information in the form of a session id and (in responses from the server) a time to live/timeout value giving the validity of the session id.

```
<definitions targetNamespace="urn:ez3950"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/so
ap/" xmlns:ez="urn:ez3950/portTypes"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema
">

  <message name="soapHeader">
      <part type="xsd:string" name="id"/>
      <part type="xsd:string" name="timeout"/>
  </message>>
  <import namespace="urn:ez3950/portTypes"
location="http://www.lib.ox.ac.uk/jafer/ez3905
0/ez3950-portTypes.wsdl"/>

    <binding name="ez3950SOAPBinding"
type="ez:ez3950PortTypes">
        <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/htt
p"/>
        <operation name="init">
           <soap:operation soapAction=""/>
           <input>
```

```
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="search">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="present">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="sort">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="scan">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="delete">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>
        <operation name="resourceReport">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="extendedService">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation name="close">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>
        <operation name="close">
           <soap:operation soapAction=""/>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
        </operation>

        <operation name="accessControl">
           <soap:operation soapAction=""/>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
        </operation>

        <operation name="resourceControl">
           <soap:operation soapAction=""/>
           <output>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
```

```
              <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
        </operation>

        <operation name="segment">
           <soap:operation soapAction=""/>
           <output>
              <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
              <soap:header
message="soapHeader" part="timeout"
use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
              <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </output>
        </operation>

        <operation
name="triggerResourceControl">
           <soap:operation soapAction=""/>
           <input>
               <soap:header
message="soapHeader" part="id" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
               <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"/>
           </input>
        </operation>

    </binding>
</definitions>
```

Specifying a implementation

A typical WSDL file for combining the Service Definition with a particular Binding to describe a server implementation would look like the following example:

```
<?xml version="1.0"?>

 <definitions name="ez3950"
xmlns:ez="urn:ez3950/SOAPBinding" >

   <import namespace="urn:ez3950/SOAPBinding"
location="http://www.lib.ox.ac.uk/jafer/ez3950
/ez3950-soapbinding.wsdl"/>

   <service name="Oxford University
Libraries">
     <documentation>Z39.50 Server for Oxford
University Libraries</documentation>
     <port name="OLIS"
binding="ez:ez3950SOAPBinding">
       <soap:address
location="http://jafer.las.ox.ac.uk/ez3950"/>
     </port>
   </service>

 </definitions>
```

Profiling

A WSDL binding does not need to bind all specified Port Types to a protocol. In addition a binding can use the part attribute to only bind certain component parts of a message specified in the Port Type. As such a WSDL binding could also be used to specify some aspects of a profile.

OAI

WSDL

```
<?xml version="1.0"?>
<definitions name="OAI"
       targetNamespace="http://www.openarchiv
es.org/OAI/1.1/Service"
       xmlns:tns="http://www.openarchives.org
/OAI/1.1/Service"
       xmlns:oai="http://www.openarchives.org
/OAI/1.1"
       xmlns="http://schemas.xmlsoap.org/wsdl
/"
       xmlns:xsd="http://www.w3.org/1999/XMLS
chema"
       xmlns:http="http://schemas.xmlsoap.org
/wsdl/http/"
       xmlns:mime="http://schemas.xmlsoap.org
/wsdl/mime/">

 <service name="OAI-Test">
  <port name="Test" binding="tns:OAI-GET">
   <http:address location="http://test.org"/>
  </port>
 </service>

 <message name="GetRecordRequest">
  <part name="identifier" type="xsd:string" />
  <part name="metadataPrefix"
type="xsd:string" />
 </message>
 <message name="GetRecordResponse">
  <part name="GetRecord"
element="oai:GetRecord"/>
 </message>

 <message name="IdentifyRequest">
 </message>
 <message name="IdentifyResponse">
  <part name="Identify"
element="oai:Identify"/>
 </message>

 <message name="ListIdentifiersRequest">
  <part name="until" type="xsd:string" />
  <part name="from" type="xsd:string" />
  <part name="set" type="xsd:string" />
  <part name="resumptionToken"
type="xsd:string" />
 </message>
 <message name="ListIdentifiersResponse">
  <part name="ListIdentifiers"
element="oai:ListIdentifiers"/>
 </message>


 <message name="ListMetadataFormatsRequest">
  <part name="identifier" type="xsd:string" />
 </message>
 <message name="ListMetadataFormatsResponse">
  <part name="ListMetadataFormats"
element="oai:ListMetadataFormats"/>
 </message>

 <message name="ListRecordsRequest">
  <part name="until" type="xsd:string" />
  <part name="from" type="xsd:string" />
  <part name="set" type="xsd:string" />
  <part name="resumptionToken"
type="xsd:string" />
  <part name="metadataPrefix"
type="xsd:string" />
 </message>
 <message name="ListRecordsResponse">
  <part name="ListRecords"
element="oai:ListRecords"/>
 </message>

<message name="ListSetsRequest">
  <part name="resumptionToken"
type="xsd:string" />
 </message>
 <message name="ListSetsResponse">
  <part name="ListSets"
element="oai:ListSets"/>
 </message>


 <portType name="OAI">
  <operation name="GetRecord">
   <input name="GetRecordRequest"
message="GetRecordRequest"/>
   <output name="GetRecordResponse"
message="GetRecordResponse"/>
  </operation>
  <operation name="Identify">
   <input name="IdentifyRequest"
message="IdentifyRequest"/>
   <output name="IdentifyResponse"
message="IdentifyResponse"/>
  </operation>
  <operation name="ListIdentifiers">
   <input name="ListIdentifiersRequest"
message="ListIdentifiersRequest"/>
   <output name="ListIdentifiersResponse"
message="ListIdentifiersResponse"/>
  </operation>
  <operation name="ListMetadataFormats">
   <input name="ListMetadataFormatsRequest"
message="ListMetadataFormatsRequest"/>
   <output name="ListMetadataFormatsResponse"
message="ListMetadataFormatsResponse"/>
  </operation>
  <operation name="ListRecords">
   <input name="ListRecordsRequest"
message="ListRecordsRequest"/>
   <output name="ListRecordsResponse"
message="ListRecordsResponse"/>
  </operation>
  <operation name="ListSets">
   <input name="ListSetsRequest"
message="ListSetsRequest"/>
   <output name="ListSetsResponse"
message="Response"/>
  </operation>
 </portType>

  <binding name="OAI-GET" type="tns:OAI">
    <http:binding verb="GET"/>
    <operation name="GetRecord">
       <http:operation
location="?verb=GetRecord"/>
       <input name="GetRecordRequest"
message="GetRecordRequest">
         <http:urlEncoded/>
       </input>
       <output name="GetRecordResponse"
message="GetRecordResponse">
            <mime:mimeXml
part="oai:GetRecord"/>
       </output>
    </operation>
    <operation name="Identify">
       <http:operation
location="?verb=Identify"/>
       <input name="IdentifyRequest"
message="IdentifyRequest">
         <http:urlEncoded/>
       </input>
       <output name="IdentifyResponse"
message="IdentifyResponse">
            <mime:mimeXml
part="oai:Identify"/>
       </output>
    </operation>
    <operation name="ListIdentifiers">
       <http:operation
location="?verb=ListIdentifiers"/>
       <input name="ListIdentifiersRequest"
message="ListIdentifiersRequest">
         <http:urlEncoded/>
       </input>
       <output name="ListIdentifiersResponse"
message="ListIdentifiersResponse">
            <mime:mimeXml
part="oai:ListIdentifiers"/>
       </output>
    </operation>
    <operation name="ListMetadataFormats">
       <http:operation
location="?verb=ListMetadataFormats"/>
       <input
name="ListMetadataFormatsRequest"
message="ListMetadataFormatsRequest">
         <http:urlEncoded/>
       </input>
       <output
name="ListMetadataFormatsResponse"
message="ListMetadataFormatsResponse">
            <mime:mimeXml
part="oai:ListMetadataFormats"/>
       </output>
    </operation>
    <operation name="ListRecords">
```

```
       <http:operation
location="?verb=ListRecords"/>
       <input name="ListRecordsRequest"
message="ListRecordsRequest">
         <http:urlEncoded/>
       </input>
       <output name="ListRecordsResponse"
message="ListRecordsResponse">
            <mime:mimeXml
part="oai:ListRecords"/>
       </output>
    </operation>
    <operation name="ListSets">
       <http:operation
location="?verb=ListSets"/>
       <input name="ListSetsRequest"
message="ListSetsRequest">
         <http:urlEncoded/>
       </input>
       <output name="ListSetsResponse"
message="ListSetsResponse">
            <mime:mimeXml
part="oai:ListSets"/>
       </output>
    </operation>
 </binding>

  <binding name="OAI-POST" type="tns:OAI">
    <http:binding verb="POST"/>
    <operation name="GetRecord">
       <http:operation
location="?verb=GetRecord"/>
       <input name="GetRecordRequest"
message="GetRecordRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output name="GetRecordResponse"
message="GetRecordResponse">
            <mime:mimeXml
part="oai:GetRecord"/>
       </output>
    </operation>
    <operation name="Identify">
       <http:operation
location="?verb=Identify"/>
       <input name="IdentifyRequest"
message="IdentifyRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output name="IdentifyResponse"
message="IdentifyResponse">
            <mime:mimeXml
part="oai:Identify"/>
       </output>
    </operation>
    <operation name="ListIdentifiers">
       <http:operation
location="?verb=ListIdentifiers"/>
       <input name="ListIdentifiersRequest"
message="ListIdentifiersRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output name="ListIdentifiersResponse"
message="ListIdentifiersResponse">
            <mime:mimeXml
part="oai:ListIdentifiers"/>
       </output>
    </operation>
    <operation name="ListMetadataFormats">
       <http:operation
location="?verb=ListMetadataFormats"/>
       <input
name="ListMetadataFormatsRequest"
message="ListMetadataFormatsRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output
name="ListMetadataFormatsResponse"
message="ListMetadataFormatsResponse">
            <mime:mimeXml
part="oai:ListMetadataFormats"/>
       </output>
    </operation>
    <operation name="ListRecords">
       <http:operation
location="?verb=ListRecords"/>
       <input name="ListRecordsRequest"
message="ListRecordsRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output name="ListRecordsResponse"
message="ListRecordsResponse">
            <mime:mimeXml
part="oai:ListRecords"/>
       </output>
    </operation>
    <operation name="ListSets">
       <http:operation
location="?verb=ListSets"/>
       <input name="ListSetsRequest"
message="ListSetsRequest">
          <mime:content type="application/x-
www-form-urlencoded"/>
       </input>
       <output name="ListSetsResponse"
message="ListSetsResponse">
            <mime:mimeXml
part="oai:ListSets"/>
       </output>
    </operation>
 </binding>


<types>
<schema
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:oai="http://www.openarchives.org/OAI/1.1
"
targetNamespace="http://www.openarchives.org/O
AI/1.1" elementFormDefault="qualified"
attributeFormDefault="unqualified">
       <!-- define recordType -->
       <!-- a record has a header and a
metadata part -->
       <complexType name="recordType">
              <sequence>
                     <element name="header"
minOccurs="1" maxOccurs="1"
type="oai:headerType"/>
                     <element
name="metadata" minOccurs="0" maxOccurs="1"
type="oai:metadataType"/>
                     <element name="about"
minOccurs="0" maxOccurs="1"
type="oai:aboutType"/>
              </sequence>
              <attribute name="status"
use="optional" type="oai:statusType"/>
       </complexType>
       <!-- define headerType -->
       <!-- a header has a unique identifier
and a datestamp -->
       <complexType name="headerType">
              <sequence>
                     <element
name="identifier" minOccurs="1" maxOccurs="1"
type="anyURI"/>
```

```
                        <element
name="datestamp" minOccurs="1" maxOccurs="1"
type="date"/>
                </sequence>
        </complexType>
        <!-- define metadataType -->
        <!-- metadata must be expressed in XML
that is compliant with another XML Schema -->
        <!-- metadata must be explicitely
qualified in the response -->
        <complexType name="metadataType">
                <sequence>
                        <any
namespace="##other" processContents="lax"/>
                </sequence>
        </complexType>
        <!-- define aboutType -->
        <!-- data "about" the record must be
expressed in XML -->
        <!-- that is compliant with an XML
Schema defined by a community -->
        <complexType name="aboutType">
                <sequence>
                        <any
namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="1"/>
                </sequence>
        </complexType>
        <!-- define statusType -->
        <!-- a record can have a status of
"deleted" -->
        <simpleType name="statusType">
                <restriction base="string">
                        <enumeration
value="deleted"/>
                </restriction>
        </simpleType>
        <annotation>
                <documentation>
    Schema to verify validity of responses to
GetRecord OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="GetRecord"
type="oai:GetRecordType"/>
        <!-- response to GetRecord-request -->
        <complexType name="GetRecordType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element name="record"
minOccurs="0" maxOccurs="1"
type="oai:recordType"/>
                </sequence>
        </complexType>
        <annotation>
                <documentation>
    Schema to verify validity of responses to
Identify OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="Identify"
type="oai:IdentifyType"/>
        <!-- response to Identify-request -->
        <complexType name="IdentifyType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element
name="repositoryName" minOccurs="1"
maxOccurs="1" type="string"/>
                        <element name="baseURL"
minOccurs="1" maxOccurs="1" type="anyURI"/>
                        <element
name="protocolVersion" minOccurs="1"
maxOccurs="1" type="string"/>
                        <element
name="adminEmail" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element
name="description" minOccurs="0"
maxOccurs="unbounded"
type="oai:descriptionType"/>
                </sequence>
        </complexType>
        <complexType name="descriptionType">
                <sequence>
                        <any
namespace="##other" processContents="lax"/>
                </sequence>
        </complexType>
        <annotation>
                <documentation>
    Schema to verify validity of responses to
ListIdentifiers OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="ListIdentifiers
type="oai:ListIdentifiersType"/>
        <!-- response to ListIdentifiers-
request -->
        <!-- records have an optional
"deleted" status -->
        <!-- this response may contain an
optional resumptionToken -->
        <complexType
name="ListIdentifiersType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element
ref="oai:identifier" minOccurs="0"
maxOccurs="unbounded"/>
                        <element
name="resumptionToken" minOccurs="0"
maxOccurs="1" type="string"/>
                </sequence>
        </complexType>
        <element name="identifier">
                <complexType>
                        <simpleContent>
                                <extension
base="anyURI">

        <attribute name="status"
use="optional" type="oai:statusType"/>
                                </extension>
                        </simpleContent>
                </complexType>
        </element>
        <annotation>
                <documentation>
    Schema to verify validity of responses to
ListMetadataFormats OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="ListMetadataFormats"
type="oai:ListMetadataType"/>
        <!-- response to ListMetadataFormats-
request -->
        <complexType name="ListMetadataType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element
name="metadataFormat" minOccurs="0"
maxOccurs="unbounded"
type="oai:metadataFormatType"/>
                </sequence>
        </complexType>
        <complexType
name="metadataFormatType">
                <sequence>
                        <element
name="metadataPrefix" minOccurs="1"
maxOccurs="1" type="oai:metadataPrefixType"/>
                        <element name="schema"
minOccurs="1" maxOccurs="1" type="anyURI"/>
                        <element
name="metadataNamespace" minOccurs="0"
maxOccurs="1" type="anyURI"/>
                </sequence>
        </complexType>
        <simpleType name="metadataPrefixType">
                <restriction base="string">
                        <pattern value="[a-zA-
Z0-9_]+"/>
                </restriction>
        </simpleType>
        <annotation>
                <documentation>
     Schema to verify validity of responses to
ListRecords OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="ListRecords"
type="oai:ListRecordsType"/>
        <!-- response to ListRecords-request -
->
        <!-- this response may contain an
optional resumptionToken -->
        <complexType name="ListRecordsType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element name="record"
minOccurs="0" maxOccurs="unbounded"
type="oai:recordType"/>
                        <element
name="resumptionToken" minOccurs="0"
maxOccurs="1" type="string"/>
                </sequence>
        </complexType>
        <annotation>
                <documentation>
     Schema to verify validity of responses to
ListSets OAI-protocol request.
    This Schema validated at
http://www.w3.org/2001/03/webdata/xsv on 2001-
05-07
    with XSV XSV 1.189/1.95 of 2001/05/07
08:38:12
  </documentation>
        </annotation>
        <element name="ListSets"
type="oai:ListSetsType"/>
        <!-- this response may contain an
optional resumptionToken -->
        <complexType name="ListSetsType">
                <sequence>
                        <element
name="responseDate" minOccurs="1"
maxOccurs="1" type="dateTime"/>
                        <element
name="requestURL" minOccurs="1" maxOccurs="1"
type="anyURI"/>
                        <element name="set"
minOccurs="0" maxOccurs="unbounded"
type="oai:setType"/>
                        <element
name="resumptionToken" minOccurs="0"
maxOccurs="1" type="string"/>
                </sequence>
        </complexType>
        <!-- each set in the list consists of
a setSpec and a pretty name -->
        <complexType name="setType">
                <sequence>
                        <element name="setSpec"
minOccurs="1" maxOccurs="1"
type="oai:setSpecType"/>
                        <element name="setName"
minOccurs="1" maxOccurs="1" type="string"/>
                </sequence>
        </complexType>
        <simpleType name="setSpecType">
                <restriction base="string">
                        <pattern value="([A-Za-
z0-9])+(:[A-Za-z0-9]+)*"/>
                </restriction>
        </simpleType>
</schema>
</types>
</definitions>
```

ZiNG

Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="urn:z3950:ZiNG:P1:ResponseSch
ema1"
xmlns="urn:z3950:ZiNG:P1:ResponseSchema1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/so
ap/" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xsd:element name="response"
type="searchRetrieveResponse"/>
        <xsd:complexType
name="searchRetrieveResponse">
                <xsd:annotation>

        <xsd:documentation>Schema of default
response format
Other response schemas should be an
xsd:extension of zng:searchRetrieveResponse,
i.e. specified using an XML schema such as the
following: <xsd:complexType
name="searchRetrieveResponseTest">

        <xsd:complexContent>

        <xsd:extension
base="zng:searchRetrieveRequest">

        <xsd:element name="sessionID"
type="xsd:string"/>

        <xsd:element name="originalQuery"
type="xsd:string"/>

        </xsd:extension>

        </xsd:complexContent>

        </xsd:complexType>
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element
name="resultSetReference"
type="resultSetReference" nillable="true">
                                <xsd:annotation>

        <xsd:documentation>
Optional Structure (defined below) indicating
a server generated result set identifer (name)
and optionally time to live
</xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                        <xsd:element
name="totalHits" type="xsd:integer"
nillable="false">
                                <xsd:annotation>

        <xsd:documentation>total number of
hits for query
</xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                        <xsd:element
name="records" type="records"
nillable="false">
                                <xsd:annotation>

        <xsd:documentation>Structure (defined
below) for holding the
records</xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                        <xsd:element
name="status" type="status" nillable="false">
                                <xsd:annotation>

        <xsd:documentation>Structure for
specifying the status of the search
</xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType
name="resultSetReference">
                <xsd:annotation>
                        <xsd:documentation>
Structure indicating a server generated result
set identifer (name) and optionally time to
live
</xsd:documentation>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element
name="resultSetName" type="xsd:string">
                                <xsd:annotation>

        <xsd:documentation>
Server generated id/name for refering to the
result set in future CQL queries
                </xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                        <xsd:element
name="resultSetTTL" type="xsd:string"
nillable="true">
                                <xsd:annotation>

        <xsd:documentation>Optional indicator
of how long the server will retain the result
set
                  (In ISO time/date format)
</xsd:documentation>

        </xsd:annotation>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="records">
                <xsd:annotation>
                        <xsd:documentation>
Array structure containing records returned
</xsd:documentation>
                </xsd:annotation>
                <xsd:complexContent>
                        <xsd:extension
base="soap:Array">
                                <xsd:sequence>

        <xsd:element name="record"
type="record">

        <xsd:annotation>

        <xsd:documentation>Structure for
holding a record</xsd:documentation>

        </xsd:annotation>

        </xsd:element>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="record">
                <xsd:annotation>
                        <xsd:documentation>
Structure for representing a record -
suggested types is currently a string for SRW
and XML for SRU
</xsd:documentation>
                </xsd:annotation>
                <xsd:sequence>
```

```
                      <xsd:element
name="schema" type="xsd:string"
nillable="false">
                              <xsd:annotation>

       <xsd:documentation>
Schema of the record. Used to indicate the XML
schema to which the contents of the
 recordData subelement must correspond.
   </xsd:documentation>

       </xsd:annotation>
                      </xsd:element>
                      <xsd:element
name="recordData" type="xsd:anyType">
                              <xsd:annotation>

       <xsd:documentation>Actual record data
returned in the XML schema specified by the
schema element above. Typical schema include:
  Dublin Core:
http://dublincore.org/documents/2001/04/11/dcm
es-xml/dcmes-xml-dtd.dtd
      Onix: onix-international.dtd
(http://www.editeur.com)   OpenArchives MARC:
http://www.openarchives.org/OAI/1.1/oai_marc.x
sd
      Surrogate Diagnostic:
{urn:z3950:ZNG_Prototype1}diagnostic</xsd:docu
mentation>

       </xsd:annotation>
                      </xsd:element>
              </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="status">
              <xsd:annotation>

       <xsd:documentation>Status of the
request
</xsd:documentation>
              </xsd:annotation>
              <xsd:sequence>
                      <xsd:element
name="statusCode" type="xsd:int">
                              <xsd:annotation>

       <xsd:documentation>Status code for
request.
 Codes are:
  0 - success
```

WSDL

```
<?xml version="1.0"?>
<definitions name="ZiNG"
       targetNamespace="urn:z3950:ZiNG:P1:Ser
vice"
       xmlns:tns="urn:z3950:ZiNG:P1:Service"
       xmlns:types="urn:z3950:ZiNG:P1:Respons
eSchema1"
       xmlns="http://schemas.xmlsoap.org/wsdl
/"
       xmlns:xsd="http://www.w3.org/2001/XMLS
chema"
       xmlns:soap="http://schemas.xmlsoap.org
/wsdl/soap/"
       xmlns:http="http://schemas.xmlsoap.org
/wsdl/http/"
       xmlns:mime="http://schemas.xmlsoap.org
/wsdl/mime/">
    <documentation>
       ZiNG Prototype 1

       History:
         2001-06-28  Initial Draft
M. J. Dovey
```

```
  1 - partial success (some surrogate
diagnostics present in records structure)   2
- failure

           </xsd:documentation>

       </xsd:annotation>
                      </xsd:element>
                      <xsd:element
name="diagnostic" type="diagnostic"
nillable="true" minOccurs="0"
maxOccurs="unbounded">
                              <xsd:annotation>

       <xsd:documentation>Optional list of
diagnostic codes</xsd:documentation>

       </xsd:annotation>
                      </xsd:element>
              </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="diagnostic">
              <xsd:annotation>
                      <xsd:documentation>
Diagnostic code structure
(used in status structure for non-surrogate
diagnostics and
 record structure for surrogate diagnostics)
</xsd:documentation>
              </xsd:annotation>
              <xsd:sequence>
                      <xsd:annotation>

       <xsd:documentation>
                  Diagnostic code (confirms to
bib1 diagnostics)
                </xsd:documentation>

       <xsd:documentation>
                  Optional additional
information
                </xsd:documentation>
                      </xsd:annotation>
                      <xsd:element
name="condition" type="xsd:int"/>
                      <xsd:element
name="additionalInformation" type="xsd:string"
nillable="true"/>
              </xsd:sequence>
       </xsd:complexType>
</xsd:schema>
         2001-07-10  Added totalHits to
response       J. Gatenby
         2001-07-12  Corrected xmlns
declarations      M. J. Dovey
         2001-07-15  Added documentation +
corrections M. J. Dovey
         2001-10-05  Changed maximumRecord
documentation on advice from ZIG  M. J. Dovey
         2001-10-12  Updated Types to conform
to latest XML Schema recommendation  M. J.
Dovey
         2002-01-20  Modified to seperate
Response Schema and name changes   M. J. Dovey
    </documentation>

    <xsd:import
namespace="urn:z3950:ZiNG:P1:ResponseSchema1"
location="rs1.xsd" />

       <!-- Service definitions -->
       <service name="ZiNG">
              <xsd:annotation>
                      <xsd:documentation>
        This specifies a sample ZNG service on
localhost:8080 with the SOAP binding.
```

```
        Used for testing WSDL compilation
tools. This will be removed from the final
WSDL description.
            </xsd:documentation>
                </xsd:annotation>
                <port binding="tns:SRW"
name="ZiNG">
                        <soap:address
location="http://localhost:8080/ZiNG"/>
                </port>
        </service>

        <!-- Message Definitions -->
        <message name="searchRetrieveRequest">
                <xsd:annotation>
                        <xsd:documentation>
        SearchRetrieve request message.
        Parts explicitly defined to allow URL
binding

        query:          query in CQL
                        (a superset of the
query part of CCL) or a specified result set
name (syntax to be finalised)
        startRecord:    record from which to
start
                        (default 0)
        maximumRecords: how many records to
return
                         (-1 is interpreted to
mean all records, Default is -1)
        responseSchema: schema of response
format
                         (default is
urn:z3950:ZiNG:P1:ResponseSchema1)
                         responseSchema should
be an xsd:extension of
zng:searchRetrieveResponse e.g. specified
using
                         an XML schema such as
the following:
        recordSchema:    schema of record
format
                         (default is ???)
           </xsd:documentation>
                </xsd:annotation>
                <part name="query"
nullable="true" type="xsd:string"/>
                <part name="startRecord"
type="xsd:int"/>
                <part name="maximumRecords"
type="xsd:int"/>
                <part name="responseSchema"
nullable="true" type="xsd:string"/>
                <part name="recordSchema"
nullable="true" type="xsd:string"/>
        </message>
        <message
name="searchRetrieveResponse">
                <xsd:annotation>
                        <xsd:documentation>
        binding for default SearchRetrieve
response message (schema
zng:searchRetrieveResponse - see types below)
            </xsd:documentation>
                </xsd:annotation>
                <part
name="searchRetrieveResponse"
type="types:searchRetrieveResponse"/>
        </message>

        <!-- Port Types -->
        <portType name="ZiNGPort">
                <xsd:annotation>
                        <xsd:documentation>
        Message pairing for SearchRetrieve
service
      </xsd:documentation>
                </xsd:annotation>
                <operation
name="searchRetrieve" paramOrder="query
startRecord maximumRecords responseSchema
recordSchema">
                        <input
message="tns:searchRetrieveRequest"
name="searchRetrieveRequest"/>
                        <output
message="tns:searchRetrieveResponse"
name="searchRetrieveResponse"/>
                </operation>
        </portType>

        <!-- Bindings to protocols -->
        <binding name="SRW"
type="tns:ZiNGPort">
                <xsd:annotation>
                        <xsd:documentation>
        This binds the ZNG messages to SOAP
over http.

        Encryption: If encryption is required
use SOAP over https.
        Authentication: If authentication is
required use http authentication.
      </xsd:documentation>
                </xsd:annotation>
                <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/htt
p/"/>
                <operation
name="searchRetrieve">
                        <soap:operation
soapAction="" style="rpc"/>
                        <input use="encoded">
                                <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"
namespace="urn:z3950:ZiNG:P1:Service"
parts="query startRecord maximumRecords
recordSchema" use="encoded"/>
                        </input>
                        <output use="encoded">
                                <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap
/encoding/"
namespace="urn:z3950:ZiNG:P1:Service"
parts="query startRecord maximumRecords
recordSchema" use="encoded"/>
                        </output>
                </operation>
        </binding>

        <binding name="SRU"
type="tns:ZiNGPort">
                <xsd:annotation>
                        <xsd:documentation>
        This binds the ZNG messages to URL GET
over http.
        (The URL parameter names are detemined
by the searchRetrieveRequest message parts
        i.e. query startRecord maximumRecords
responseSchema recordSchema)


        Encryption: If encryption is required
use https.
        Authentication: If authentication is
required use http authentication.
      </xsd:documentation>
                </xsd:annotation>
                <http:binding verb="GET"/>
                <operation
name="searchRetrieve">
```

```
                    <http:operation
location=""/>
                    <input>

       <http:urlEncoded/>
                    </input>
                    <output>
```

```
                                   <mime:content
type="text/xml"/>
                           </output>
                    </operation>
             </binding>

</definitions>
```

WSIL

Web Services Inspection Language (WSIL) is a draft standard from IBM and Microsoft. It acts as a containing listing links to WSDL descriptions of WebServices which a particular organisation may make available. A simple example giving the locations of the WSDL descriptions of two services plus the link where the most up to date WSIL file can be located is given below (for the second service a link to its UDDI description is also given):

```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
  <service>
    <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                 location="http://example.com/exampleservice.wsdl" />
  </service>
  <service>
    <description referencedNamespace="urn:uddi-org:api">
       <wsiluddi:serviceDescription location=
                   "http://example.com/uddi/inquiryapi">
         <wsiluddi:serviceKey>
                   52946BB0-BC28-11D5-A432-0004AC49CC1E</wsiluddi:serviceKey>
       </wsiluddi:serviceDescription>
    </description>
  </service>
  <link referencedNamespace=
                   "http://schemas.xmlsoap.org/ws/2001/10/inspection/"
        location="http://example.com/tools/toolservices.wsil"/>
</inspection>
```

The intent is that this document can be placed at a well known place on the organisation's web site (the recommendation is to put this in a file called inspection.wsil located at the root of the web site), so that the services offered by an organisation can be located quickly by accessing that organisations web site. WSIL can also be embedded in the headers of a HTML page.

WSIL is a successor to the proprietary Microsoft DISCO (DISCOvery) XML format that is present in the beta's of Visual Studio .NET.

WSUI

The OASIS Web Services Standard Body is developing an XML language for describing the user interface to Web Services called WSUI (Web Service User Interface). An intelligent client could therefore build its interface dynamically from the WSUI description. The WSUI reference implementation includes a portal which dynamically builds its portlets from WSUI descriptions. In future this could provide the ability for a user to locate content services via UDDI and for a portal server to automatically generate a portal interface dynamically without any programming required by the portal builder.

WSFL et al.

There are also a number of other WebService description languages being developed by the WebService community. These currently include:

- WSFL – Web Services Flow Language, used to describe the process flow of a Web Service.
- WSC -Web Services Choreography, used to describe combinations of web services
- WSEL -Web Services Endpoint Language describes properties of web service implementations

Glossary

API – Application Programming Interface. This provides programmer access to the functionality of a software system.
CGI – Common Gateway Interface. A standard for writing server processes called by a web server. (http://hoohoo.ncsa.uiuc.edu/cgi/overview.html)
CORBA – Common Object Request Broker Architecture. A standard for allowing software running on different computers to interact. (http://www.corba.org)
DNER – The Distributed Network for Electronic Resources. A JISC programme for integrating electronic services. (http://www.jisc.ac.uk/dner).
Dublin Core – a set of common metadata identifiers for describing resources. (http://dublincore.org/)

e-Lib – electronic Libraries. A JISC funded programme for developing electronic library services. (http://www.jisc.ac.uk/elib)
GRID Computing – An initiative to build large scale distributed computing resources (http://www.gridforum.org)
HTTP – HyperText Transfer Protocol. The communication protocol between web browsers and web servers. It also allows message passing between servers. (http://www.w3.org/Protocols/)
HTTPS – Secure HyperText Transfer Protocol. An encrypted form of HTTP used to prevent eavesdropping or hijacking of information. (http://www.w3.org/Protocols/)
ISP – Internet Service Provider. Companies providing dial-up or broadband access to the Internet.

Java – an object oriented programming language and cross-platform environment for developing and running software. The compiled software runs on a Java Virtual Machine and therefore can be run on any platform for which Java has been implemented. Java also benefits from being designed to avoid common programming issues with C++ and earlier languages. Java versions 1.2 and above are collectively referred to as Java 2. (http://www.javasoft.com)

JavaBean – A java based component architecture for building systems. (http://www.javasoft.com/products/javabeans)

JINI – Jini network technology is an open architecture that enables developers to create network-centric services (http://www.jini.org)

JSP – Java Server Pages. A server side scripting language. Has the advantage over other languages of being precompiled. (http://java.sun.com/jsp)
JVM – Java Virtual Machine. The software environment under which a Java program runs. A java program can hence run on any platform for which a JVM is available.

JXTA – JuXTApose. A Java framework for peer to peer processes (http://www.jxta.org).

MIA – MODEL's Information Architecture. A technical overview of the issues involved in implementing the concepts developed in MODELS.

(http:// http://www.ukoln.ac.uk/dlis/models/requirements/arch)
MLE – Managed Learning Environment (http://www.jisc.ac.uk/jciel/mlesg/)

MODELS – A series of workshops run by UKOLN and funded by the JISC to investigate the technical issues surrounding integrated service access. It has provided the basic concepts behind the DNER programme. (http://www.ukoln.ac.uk/dlis/models)
OASIS – the Organization for the Advancement of Structured Information Standards. A non-profit, international consortium that creates interoperable industry specifications based on public standards such as XML and SGML, as well as others that are related to structured information processing. (http://www.oasis-open.org)
OCS – Open Content Syndication. An XML format for publishing information channels. (http://internetalchemy.org/ocs/directory.html)
Perl – A scripting language particular suited to string processing due to powerful regular expression support. (http://www.perl.org)
PHP – HyperText Preprocessor. A scripting language designed for server side web scripting. (http://www.php.net)
RDF – Resource Description Format. A metadata framework for describing resources. (http://www.w3.org/RDF/)

RDN – Resource Discovery Network. An service to provide integrated access to internet resources via subject based gateways. (http://www.rdn.ac.uk)
RSS – RDF Site Summary. An XML format for publishing information channels. (http://www.purl.org/rss/)
Servlet – Java application called by a web server. (http://java.sun.com/products/servlet/)
SMTP – Simple Mail Transport Protocol. The protocol used to deliver internet e-mail. (IETF RFC-821)
SOAP – Simple Object Access Protocol. A Microsoft proposal for allowing client-server communication to work by passing messages in XML (normally over HTTP). It is a proposal from Microsoft. (http://msdn.microsoft.com/soap/)
UDDI – Univeral Description, Discovery and Integration of Web Services. A specification, protocol and registries for describing and locating Web Services. (http://www.uddi.org)

URL – Uniform Resource Location. A mechanism for identifying information on the web. (http://www.w3.org/Addressing/)
VLE – Virtual Learning Environment (http://www.jisc.ac.uk/pub00/req-vle.html).

Web Services – A framework for interoperable distributed computing. (http://www.xml.com/pub/a/2001/04/04/webservices/)
WSDL – Web Service Description Language. An IBM/Microsoft proposal for a language for describing the API for Web Services in XML. (http://msdn.microsoft.com/xml/general/wsdl.asp)
WSUI – Web Service User Interface. An XML language for describing the user interface for a Web Service. (http://www.wsui.org)
WSIL – Web Service Inspection Language. An XML container for links to WSDL description of WebServices offered by an organisation. (http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html)
XHTML – XML HTML. An XML compliant version of HTML. (http://www.w3c.org/MarkUp/)
XML – eXtended Markup Language. A tagged based machine-readable language for describing tree structures based on SGML. (http://www.w3.org/XML/)
XSL – XML Style Language. This covers both XSLT and XSL FO. (http://www.w3c.org/Style/XSL/)

XSL FO – XML Style Language Formatting Objects. An XML typesetting language. Often used during XSLT transforms as an intermediate to a format such as PDF. (http://www.w3c.org/Style/XSL/)
XSLT – XML Style Language Transforms. An XML based language for transforming XML documents. (http://www.w3c.org/Style/XSL/)

References

An Analytical Model of Collections and their Catalogues. Michael Heaney. http://www.ukoln.ac.uk/metadata/rslp/model/

Analysis of the Z39.50 Profile for Access to Digital Collections and the Z39.50 Explain Service for UKOLN. http://www.ukoln.ac.uk/metadata/cld/study/crossnet/zpadc.pdf

The DNER Technical Architecture: scoping the information environment. Andy Powell & Liz Lyon. http://www.ukoln.ac.uk/distributed-systems/dner/arch/dner-arch.html.

ebXML. http://www.ebxml.org

explain--. http://explain.z3950.org

Global Information Location Service. http://www.gils.net

UDDI Organisation. http://www.uddi.org

Global GRID Forum. http://www.ggf.org

Grid Information Services for Distributed Resource Sharing. Karl Czajkowski, Steven Fitzgerald, Ian Foster, Carl Kesselman, Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, 2001

Open Archives Initiative. http://www.openarchives.org

Information Retrieval (Z39.50): Application Service Definition and Protocol Specification
WebServices. http://www.webservices.org

WEBCLARITY RESOURCE REGISTRY - Distributed Directory Service Technology; Technical White Paper. http://www.webclarity.info/whitepapers/registry_white.pdf