Open Geospatial Consortium Inc.

Date: 2005-05-20

Reference number of this OGC™ project document: **OGC 04-021r3**

Version: 2.0.0 with Corregendum

Category: OGC™ Implementation Specification

Editors: Douglas Nebert, Arliss Whiteside

OGC™ Catalogue Services Specification

Warning

The technical corrigendum included in Annex E of this document describes the changes in this revision.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC™ Publicly Available Standard
Document subtype:	Implementation Specification
Document stage:	Adopted Revision
Document language:	English

Contents Page

Figures

Page

Tables

i. Preface

This document explains how Catalogue Services version 2.0 are organised and implemented for the discovery and retrieval of spatial data and services metadata. The prior public version of this specification was 1.1.1. Catalogue Services version 2.0 supersedes and deprecates version 1.1.1.

This revision of this document has been significantly improved, largely based on the comments on document 04-021 received by the Revison Working Group.

ii. Submitting organizations

The following organizations submitted the original document or its revisions to the Open GIS Consortium, Inc. in response to the OGC Request 6, Core Task Force, Catalogue Working Group, A Request for Proposals: OpenGIS® Catalogue Interface (OpenGIS® Project Document Number 98-001r2):

BAE SYSTEMS Mission Solutions (formerly Marconi Integrated Systems, Inc.)
Blue Angel Technologies, Inc.
Environmental Systems Research Institute (ESRI)
Geomatics Canada (Canada Centre for Remote Sensing (CCRS))
Intergraph Corporation
MITRE
Oracle Corporation
U.S. Federal Geographic Data Committee (FGDC)
U.S. National Aeronautics and Space Administration (NASA)
U.S. National Imagery and Mapping Agency (NIMA)

Contributing Entities
The submitting entities were grateful for the contributions from the following companies in the development and revision of this Interface Specification:

Compusult, Limited
Con terra GmbH
Cubewerx
Galdos Systems, Inc
GEODAN IT bv
Hammon, Jensen, Wallen & Associates, Inc (HJW)
Ionic Software, sa
JRC (Joint Research Centre), European Commission
SICAD GEOMATICS
Traverse Technologies

iii. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Person	Company	Address	Phone	Email
Doug Nebert	U.S. Federal Geographic Data Committee	USGS National Center, Mail Stop 590 12201 Sunrise Valley Drive Reston, VA 20192	+1 703-648-4151 fax: +1 703-648-5755	ddnebert@usgs.gov
Arliss Whiteside	BAE SYSTEMS Mission Solutions	10920 Technology Dr. San Diego, CA 92127-1608 USA	+1 858-592-1608	Arliss.Whiteside@baesystems.com
Peter Vretanos	CubeWerx, Inc.	5 Wexford Blvd. Toronto, Ontario M1R 1K9 Canada	+1 416-701-1985	pvretano@cubewerx.com
Louis Reich	NASA (Computer Sciences Corp)		+1 301-794-2195	louis.i.reich@gsfc.nasa.gov
Richard Martell	Galdos Systems Inc.	1155 W Pender St, Suite 200, Vancouver, BC V6E 2P4 Canada	+1 604 484-2750	rmartell@galdosinc.com
Uwe Voges	con terra GmbH	Martin-Luther-King-Weg 24 Münster 48155 Germany	+49 251 74 74 402	voges@conterra.de

iv. Revision history

Date	Release	Editor	Primary clauses modified	Description
12Aug1999	1.0	Nebert	N/A	Original Specification entitled "Catalogue Interface Implementation Specification" OGC Document 00-034
28Mar2001	1.1	Nebert	Made fine-grain CORBA and OLE/COM Annexes to Informative, added abstract conformance test suite, fixed coarse-grain CORBA IDL	Document only made available to OGC membership pending passage of Version 2.0. (OGC Document 01-040)
11Nov2002	1.1.1	Nebert, Katz,	State diagram changes, renamed specification and changed WWW Profile to Z39.50 Profile, added introductory words as required for new format	Document primarily reflects conversion to newer OGC/ISO document format
22Dec2003	2.0.0	Nebert	All	Reorganised and largely rewrote document.
6Mar2004	2.0.0	Nebert	Clauses 6,7, 9,10,11	Edited CORBA, Z39.50, and HTTP to reflect changes in General Model, other minor revisions to document
29Mar2004	2.0.0	Whiteside	All	
14Apr2004	2.0.0	Whiteside	All	
11May2004	2.0.0	Nebert	Merge updates on Clauses 1-5, 6-8, 9, 10, and 11 from separate editors	Responded to all RWG review comments.
20May2005	2.0.1		See Annex E	Technical corrigendum 1

v. Changes to the OGC™ Abstract Specification

The OGC™ Abstract Specification requires minor changes to accommodate the technical contents of this document to reflect the use of catalogue services to search for and retrieve any type of information object (data, service instance, service type, schema, style description, etc.) based on its properties described in "metadata."

vi. Future work

Future work may include expansion of definitions of ordering capabilities, standing orders or queries, and transactional interfaces. Individual Application Profiles (see Clause 11) will be submitted as separate but dependent specifications.

Foreword

This version deprecates and replaces Catalogue Service Specification 1.1.1 (OGC Document 02-087r3).

This document, through its implementation profiles, references several external standards and specifications as dependencies:

a) Common Object Request Broker Architecture (CORBA/IIOP), Version 2.X, The Object Management Group (OMG): http://www.omg.org

b) Information and documentation -- Information retrieval (Z39.50) -- Application service definition and protocol specification: http://www.iso.ch/iso/en/CatalogDetailPage.CatalogDetail?CSNUMBER=27446&ICS1=35&ICS2=240&ICS3=30

c) ISO/IEC TR 10000-1:1998. *Information Technology – Framework and taxonomy of International Standardised Profiles – Part 1: General principles and documentation framework*. Technical Report, JTC 1. Fourth edition. Available [online]: <http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/c030726_ISO_IEC_TR_10000-1_1998(E).zip>.

d) ISO/IEC 10746-2:1996. *Information Technology – Open Distributed Processing – Reference Model: Foundations*. Common text with ITU-T Recommendation X.902. Available [online]: <http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/s018836_ISO_IEC_107462_1996(E).zip>.

e) Unified Modeling Language (UML) Version 1.3, The Object Management Group (OMG): http://www.omg.org/cgi-bin/doc?formal/00-03-01

f) The Extensible Markup Language (XML), World Wide Web Consortium, http://www.w3.org/TR/1998/REC-xml-19980210

Annex A, the Abstract Conformance Test Suite, is normative to this specification and shall be implemented when a computing environment requires catalogue services. All other annexes are informative and provide background information, such as terminology and alternative implementation approaches.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium, Inc. shall not be held responsible for identifying any or all such patent rights.

Introduction

This document specifies the interfaces between clients and catalogue services, through the presentation of abstract and implementation-specific models. Catalogue services support the ability to publish and search collections of descriptive information (metadata) for data, services, and related information objects. Metadata in catalogues represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software. Catalogue services are required to support the discovery and binding to registered information resources within an information community.

OGC™ Catalogue Services Specification

1 Scope

This OGC™ document specifies the interfaces, bindings, and a framework for defining application profiles required to publish and access digital catalogues of metadata for geospatial data, services, and related resource information. Metadata act as generalised properties that can be queried and returned through catalogue services for resource evaluation and, in many cases, invocation or retrieval of the referenced resource. Catalogue services support the use of one of several identified query languages to find and return results using well-known content models (metadata schemas) and encodings. This OpenGIS® document is applicable to the implementation of interfaces on catalogues of a variety of information resources.

The target audience for this specification is the community of software developers who are:

a) Implementers of OGC compliant Catalogue servers

b) Implementers of OGC compliant Catalogue clients

2 Conformance

Abstract conformance to the mandatory catalogue service interfaces is described in Annex A. In a given community, a test suite should include test metadata records with a variety of element values and a series of queries that would return correct and properly formatted results. Test data and queries may be included in associated Application Profiles.

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this part of OGC 03-108. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

ANSI/NISO Z39.50-2003, Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (revision of Z39.50-1995)
http://www.iso.ch/iso/en/CatalogDetailPage.CatalogDetail?CSNUMBER=27446&ICS1=35&ICS2=240&ICS3=30

IETF RFC 2045 (November 1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, Freed, N. and Borenstein N., eds., <http://www.ietf.org/rfc/rfc2045.txt>

IETF RFC 2141 (May 1997), *URN Syntax*, R. Moats <http://www.ietf.org/rfc/rfc2141.txt>

IETF RFC 2396 (August 1998), *Uniform Resource Identifiers (URI): Generic Syntax*, Berners-Lee, T., Fielding, N., and Masinter, L., eds., <http://www.ietf.org/rfc/rfc2396.txt>

IETF RFC 2616 (June 1999), *Hypertext Transfer Protocol – HTTP/1.1*, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., eds., <http://www.ietf.org/rfc/rfc2616.txt>

IANA, Internet Assigned Numbers Authority, MIME Media Types, available at <http://www.iana.org/assignments/media-types/>

CORBA/IIOP, Common Object Request Broker Architecture, Version 2.X, The Object Management Group (OMG): http://www.omg.org

ISO/IEC TR 10000-1:1998. *Information Technology – Framework and taxonomy of International Standardised Profiles – Part 1: General principles and documentation framework*. Technical Report, JTC 1. Fourth edition. Available [online]: <http://www.iso.ch/iso/en/ittf/ PubliclyAvailableStandards/c030726_ISO_IEC_TR_10000-1_1998(E).zip>.

ISO/IEC 10746-2:1996. *Information Technology – Open Distributed Processing – Reference Model: Foundations*. Common text with ITU-T Recommendation X.902. Available [online]: <http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/s018836_ISO_IEC_107462_ 1996(E).zip>.

ISO 4217:2001, *Codes for the representation of currencies and funds*

ISO 8601:2000(E), *Data elements and interchange formats - Information interchange - Representation of dates and times*.

ISO 11180, *Postal addressing*

ISO 19106:2003, *Geographic Information – Profiles*

ISO 19115:2003, *Geographic Information – Metadata*

ISO/DIS 19119, *Geographic Information – Services*

OASIS/ebXML Registry Services Specification v2.5

OGC 99-054, OpenGIS® Simple Features Specification for CORBA

OGC 99-113, OGC Abstract Specification Topic 13: Catalogue Services

OGC 02-006, OGC Abstract Specification Topic 12: OpenGIS Service Architecture

OGC 02-059, Filter Encoding Implementation Specification

OGC 04-016r2, OWS Common Implementation Specification, January 2004

NOTE This document is currently an OGC Recommendation Paper, but will become an Implementation Specification with or prior to acceptance of the first Implementation Specification that normatively references this document.

This OWS Common Implementation Specification contains a list of normative references that are also applicable to this Implementation Specification.

OMG UML, Unified Modeling Language, Version 1.3, The Object Management Group (OMG): http://www.omg.org/cgi-bin/doc?formal/00-03-01

W3C Recommendation January 1999, *Namespaces In XML*, http://www.w3.org/TR/2000/REC-xml-names.

W3C Recommendation 6 October 2000, *Extensible Markup Language (XML) 1.0* (Second Edition), http://www.w3.org/TR/REC-xml

W3C Recommendation 2 May 2001: *XML Schema Part 0: Primer,* http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

W3C Recommendation 2 May 2001: *XML Schema Part 1: Structures,* http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

W3C Recommendation 2 May 2001: *XML Schema Part 2: Datatypes,* http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

W3C Recommendation (24 June 2003): SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/SOAP/

In addition to this document, this specification includes several normative XML Schema files. These are posted online at the URL http://schemas.opengis.net/. These XML Schema files are also bundled with this document. In the event of a discrepancy between the bundled and online versions of the XML Schema files, the online files shall be considered authoritative.

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply:

4.1
client
software component that can invoke an **operation** from a **server**

4.2
data clearinghouse
collection of institutions providing digital data, which can be searched through a single interface using a common metadata standard [ISO 19115]

4.3
data level
stratum within a set of layered levels in which data is recorded that conforms to definitions of types found at the application model level [ISO 19101]

4.4
dataset series
collection of datasets sharing the same product specification [ISO 19113, ISO 19114, ISO 19115]

4.5
feature catalogue
catalogue containing definitions and descriptions of the feature types, feature attributes, and feature relationships occurring in one or more sets of geographic data, together with any feature operations that may be applied [ISO 19101, ISO 19110]

4.6
geographic dataset
dataset with a spatial aspect [ISO 19115]

4.7
geographic information
information concerning phenomena implicitly or explicitly associated with a location relative to the Earth [ISO 19128 draft]

4.8
identifier
a character string that may be composed of numbers and characters that is exchanged between the client and the server with respect to a specific identity of a resource

4.9
interface
named set of operations that characterize the behaviour of an entity [ISO 19119]

4.10
metadata dataset
metadata describing a specific dataset [ISO 19101]

4.11
metadata entity
group of metadata elements and other metadata entities describing the same aspect of data

NOTE 1 A metadata entity may contain one or more metadata entities.

NOTE 2 A metadata entity is equivalent to a class in UML terminology [ISO 19115].

4.12
metadata schema
conceptual schema describing metadata

NOTE ISO 19115 describes a standard for a metadata schema. [ISO 19101]

4.13
metadata section
subset of metadata that defines a collection of related metadata entities and elements [ISO 19115]

4.14
operation
specification of a transformation or query that an object may be called to execute [ISO 19119]

4.15
parameter
variable whose name and value are included in an operation **request** or **response**

4.16
profile
set of one or more base standards and - where applicable - the identification of chosen clauses, classes, subsets, options and parameters of those base standards that are necessary for accomplishing a particular function [ISO 19101, ISO 19106]

4.17
qualified name
name that is prefixed with its naming context

EXAMPLE The qualified name for the road no attribute in class Road defined in the Roadmap schema is RoadMap.Road.road_no. [ISO 19118].

4.18
reporting group
data with common characteristics forming a subset of a dataset

NOTE 1 Common characteristics can include belonging to an identified feature type, feature attribute or feature relationship; sharing data collection criteria; sharing original source; or being within a specified geographic or temporal extent.

NOTE 2 A reporting group can be as small as a feature instance, an attribute value, or a single feature relationship. [ISO 19109, ISO 19113].

4.19
request
invocation of an **operation** by a **client**

4.20
response
result of an **operation,** returned from a **server** to a **client**

4.21
schema
formal description of a model [ISO 19101, ISO 19103, ISO 19109, ISO 19118]

4.22
server
service instance
a particular instance of a **service** [ISO 19119 edited]

4.23
service
distinct part of the functionality that is provided by an entity through interfaces [ISO 19119]

capability which a service provider entity makes available to a service user entity at the interface between those entities [ISO 19104 terms repository]

4.24
service interface
shared boundary between an automated system or human being and another automated system or human being [ISO 19101]

4.25
service metadata
metadata describing the **operations** and **geographic information** available at a **server** [ISO 19128 draft]

4.26
state
condition that persists for a period

NOTE The value of a particular feature attribute describes a condition of the feature [ISO 19108].

4.27
transfer protocol
common set of rules for defining interactions between distributed systems [ISO 19118]

4.28
version
version of an Implementation Specification (document) and XML Schemas to which the requested operation conforms

NOTE An OWS Implementation Specification version may specify XML Schemas against which an XML encoded operation request or response must conform and should be validated.

5 Conventions

5.1 Symbols (and abbreviated terms)

Some frequently used abbreviated terms:

CORBA	Common Object Request Broker Architecture
DCP	Distributed Computing Platform
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
ISO	International Organization for Standardization
KVP	Keyword Value Pair
MIME	Multipurpose Internet Mail Extensions
OGC	Open GIS Consortium, also referred to as OpenGIS®
TBD	To Be Determined
TBR	To Be Reviewed
UML	Unified Modeling Language
XML	Extensible Markup Language
Z39.50	Service definition for information search and retrieval, also known as ISO 23950

5.2 UML notation

Some of the diagrams in this document are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this document are described in Figure 1, below.

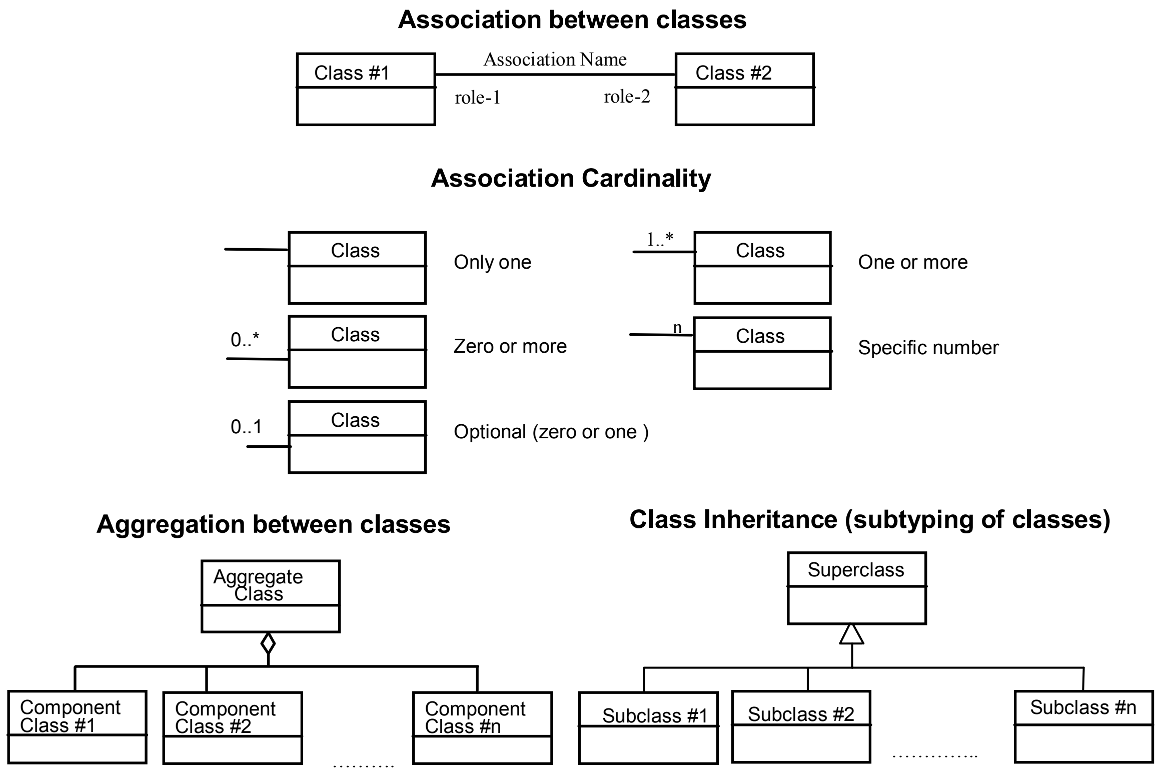

Figure 1 — UML notations

In these UML class diagrams, the class boxes with a light background are the primary classes being shown in this diagram, often the classes from one UML package. The class boxes with a gray background are other classes used by these primary classes, usually classes from other packages.

In this diagram, the following stereotypes of UML classes are used:

a) <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.

b) <<Type>> A stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A Type class may have attributes and associations.

c) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.

d) <<CodeList>> A flexible enumeration that uses string values for expressing a list of potential values. If the list alternatives are completely known, an enumeration shall be used; if the only likely alternatives are known, a code list shall be used.

e) <<Enumeration>> A data type whose instances form a list of alternative literal values. Enumeration means a short list of well-understood potential values within a class.

In this document, the following standard data types are used:

f) CharacterString – A sequence of characters

g) Boolean – A value specifying TRUE or FALSE

h) Integer – An integer number

i) Identifier – Unique identifier of an object

j) URI – An identifier of a resource that provides more information

k) URL – An identifier of an on-line resource that can be electronically accessed

5.3 Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 04-016r2].

6 Catalogue abstract information model

6.1 Introduction

The abstract information model specifies a BNF grammar for a minimal query language, a set of core queryable attributes (names,definitions,conceptual datatypes), and a common record format that defines the minimal set of elements that should be returned in the brief and summary element sets

The geospatial community is a very broad-based community that works in many different operational environments, as shown in the information discovery continuum in Figure 2. On one extreme there are tightly coupled systems dedicated to well defined functions in a tightly controlled environment. At the other extreme are Web based services that know nothing about the client. This document provides a specification that is applicable to the full range of catalogue operating environments.

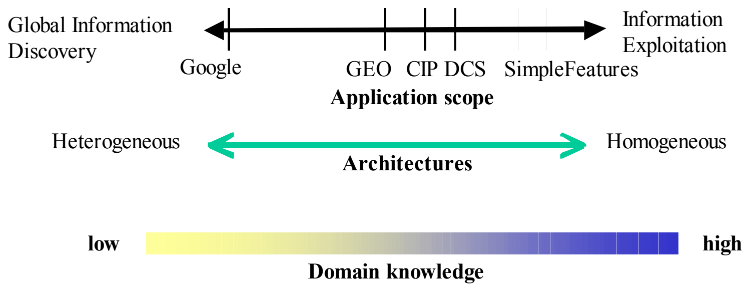

Figure 2 — Information discovery continuum

6.2 Query language support

6.2.1 Introduction

The query capabilities of the OGC General Catalogue Model provide a minimum set of data types and query operations that can be assumed of OGC Compliant Catalogue implementations. In addition, these Query Capabilities provide a high degree of flexibility enabling alternate styles of query, result presentation, and the potential support of any geo-enabled query language. This flexibility is provided by the query operation that contains the parameters needed to select the query result presentation style and to provide a query expression which includes the actual query with an identification of the query language used. The query operation, query expression, and other related operations are further discussed in Clause 7.2.4.

The interoperability goal is supported by the specification of a minimal abstract query (predicate) language, which must be supported by all compliant OpenGIS Catalogue Services. This query language supports nested Boolean queries, text matching operations, temporal data types, and geospatial operators. The minimal query language syntax is based on the SQL WHERE clause in the SQL SELECT statement. Implementations of query languages that are transformable to the OGC_Common Catalogue Query Language are the OGC Filter Specification and the CIP and GEO profiles of Z39.50 Type-1 queries.

The minimal query language assists the consumer in the discovery of datasets of interest at all sites supporting the OpenGIS Catalogue Services. The ability to specify alternative query languages allows for evolution and higher levels of interoperability among more tightly coupled communities of Catalogue Service Providers and Consumers.

6.2.2 OGC_Common catalogue query language

This subclause defines the BNF for the OGC_Common Catalogue Query Language. OGC_Common is the query language to be supported by all OGC Catalogue Interfaces in order to support search interoperability.

Assumptions made during the development of OGC_Common Query Language:

a) The query will have a syntax similar to the SQL “Where Clause.”

b) The expressiveness of the query will not require extensions to various current query systems used in geospatial catalogue queries other than the implementation of some geo operators.

c) The query language is extensible.

d) OGC_Common supports both tight and loose queries. A tight query is defined where if a catalogue doesn’t support an attribute/column specified in the query, no entity/row can match the query and the null set is returned. In a loose query, if an attribute is undefined, it is assumed to match.

BNF definition of OGC_Common Query Language:

```
<SQL terminal character> ::= <SQL language character>
<SQL language character> ::= <simple Latin letter>
                           | <digit>
                           | <SQL special character>
<simple Latin letter> ::= <simple Latin upper case letter>
```

```
                          | <simple Latin lower case letter>
<simple Latin upper case letter> ::=
     A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
   | P | Q | R | S | T | U | V | W | X | Y | Z
<simple Latin lower case letter> ::=
     a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
   | p | q | r | s | t | u | v | w | x | y | z
<digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<SQL special character> ::= <space>
                          | <double quote>
                          | <percent>
                          | <ampersand>
                          | <quote>
                          | <left paren>
                          | <right paren>
                          | <asterisk>
                          | <plus sign>
                          | <comma>
                          | <minus sign>
                          | <period>
                          | <solidus>
                          | <colon>
                          | <semicolon>
                          | <less than operator>
                          | <equals operator>
                          | <greater than operator>
                          | <question mark>
                          | <left bracket>
                          | <right bracket>
                          | <circumflex>
                          | <underscore>
                          | <vertical bar>
                          | <left brace>
                          | <right brace>
<space> ::= /*space character in character set in use
              In ASCII it would be 40*/
<double quote> ::= "
<percent> ::= %
<ampersand> ::= &
<quote> ::= '
<left paren> ::= (
<right paren> ::= )
<asterisk> ::= *
<plus sign> ::= +
<comma> ::= ,
<minus sign> ::= -
<period> ::= .
<solidus> ::= /
<colon> ::= :
<semicolon> ::= ;
<less than operator> ::= <
<equals operator> ::= =
<greater than operator> ::= >
<question mark> ::= ?
<left bracket> ::= [
<right bracket> ::= ]
```

```
<circumflex> ::= ^
<underscore> ::= _
<vertical bar> ::= |
<left brace> ::={
<right brace> ::=}
<separator> ::= { <comment> | <space> | <newline> }...
/* The next section of the BNF defines the tokens available to the
   language. I have deleted the concepts of bit string, hex string and
national character string literal. Keywords have been added to support the
geo literals. */
<token> ::= <nondelimiter token>
          | <delimiter token>
<nondelimiter token> ::= <regular identifier>
                       | <key word>
                       | <unsigned numeric literal>
<regular identifier> ::= <identifier body>
<identifier body> ::=
<identifier start> [ { <underscore> | <identifier part> }... ]
<identifier start> ::= <simple latin letter>
<identifier part> ::= <identifier start>
                    | <digit>
<key word> ::= <reserved word>
<reserved word> ::= AND | POINT | LINESTRING | POLYGON | MULTIPOINT
                   | MULTILINESTRING | MULTIPOLYGON | EMPTY | DATE
                   | TIME | TIMESTAMP | FALSE| TRUE | UNKNOWN | LIKE
                   | MINUTE | MONTH | NOT | NULL
<unsigned numeric literal> ::= <exact numeric literal>
                             | <approximate numeric literal>
<exact numeric literal> ::=
    <unsigned integer> [<period>[<unsigned integer> ] ]
    | <period> <unsigned integer>
<unsigned integer> ::= <digit>...
<approximate numeric literal> ::= <mantissa> E <exponent>
<mantissa> ::= <exact numeric literal>
<exponent> ::= <signed integer>
<signed integer> ::= [ <sign> ] <unsigned integer>
<sign> ::= <plus sign> | <minus sign>
<character string literal> ::=
   <quote> [ <character representation>... ] <quote>
<character representation> ::= <nonquote character> | <quote symbol>
<quote symbol> ::= <quote><quote>
/*End of non delimiter tokens*/
/* I have limited the delimiter tokens by eliminating, interval strings
and delimited identifiers BNF and simplifying the legal character set to
the characters to a single set so no identification of character set would
be needed decision. */
<delimiter token> ::= <character string literal>
                    | <SQL special character>
                    | <not equals operator>
                    | <greater than or equals operator>
                    | <less than or equals operator>
                    | <concatenation operator>
                    | <double greater than operator>
                    | <right arrow>
                    | <left bracket>
                    | <right bracket>
<character string literal> ::=
   <quote> [ <character representation>... ] <quote>
<character representation> ::= <nonquote character> | <quote symbol>
<quote symbol> ::= <quote><quote>
<not equals operator> ::= <>
<greater than or equals operator> ::= >=
<less than or equals operator> ::= <=
/*The following section is intended to give context for identifier and
namespaces.  It assumes that the default namespace is specified in the
query request and does not allow any overrides of the namepace */
<identifier> ::=
    <identifier start [ { <colon> | <identifier part> }... ]
<identifier start> ::= <simple Latin letter>
<identifier part> ::= <simple Latin letter> | <digit>
<attribute name> ::= <simple attribute name> | <compound attribute name>
<simple attribute name> ::= <identifier>
<compound attribute name> ::= <identifier><period>
                              [{<identifier><period>}…]
                              <simple attribute name>

/*The rest of the BNF is the real BNF for the query capabilities.*/
<search condition> ::= <boolean value expression>
<boolean value expression> ::= <boolean term>
               | <boolean value expression> OR <boolean term>
<boolean term> ::= <boolean factor>
                 | <boolean term> AND <boolean factor>
<boolean factor> ::= [ NOT ] <boolean primary>
<boolean primary> ::= <predicate> | <routine invocation>
      | <routine invocation>
   | <left paren> <search condition> <right paren>
<predicate> ::= <comparison predicate>
              | <text predicate>
              | <null predicate>
              | <temporal predicate>
              | <classification predicate>
              | <existence_predicate>

/* This set of productions define temporal predicates */
<temporal predicate> ::= <attribute_name> BEFORE <date-time expression>
                       | <attribute_name> BEFORE OR DURING <period>
                       | <attribute_name> DURING <period>
                       | <attribute_name> DURING OR AFTER <period>
                       | <attribute_name> AFTER <date-time expression>
<date-time expression ::= <date-time> | <period>

/* This set of productions enables loose or tight queries. For example the
predicate “cloudcover EXISTS” evaluates as true for all record instances
where the attribute cloudcover is a member of the record schema.
Similarly, the predicate “cloudcover DOESNOTEXIST” evaluates as true for
all record instances where the attribute cloudcover is not a member of the
record schema.*/

<existence_predicate> := <attribute_name> EXISTS
                       | <attribute_name> DOES-NOT-EXIST

<comparison predicate> ::= <attribute name> <comp op> <literal>
<text predicate> ::= <attribute name> [ NOT ] LIKE <character pattern>
<null predicate> ::= <attribute name> IS [ NOT ] NULL
```

```
<character pattern> ::= <character string literal>
      /* In a character pattern the character percent is used as a
wildcard to represent an arbitrary string. This allows LIKE to implement
the effect of many characters matching operations, such as: contains,
begins with, ends with, not contains, not begins with, not ends with, and
so forth. For example:
         attribute like '%contains_this%'
         attribute like 'begins_with_this%'
         attribute like '%ends_with_this'
         attribute like 'd_ve' will match 'dave' or 'dove'
         attribute not like '%will_not_contain_this%'
         attribute not like 'will_not_begin_with_this%'
         attribute not like '%will_not_end_with_this'   */
<comp op> ::= <equals operator>
            | <not equals operator>
            | <less than operator>
            | <greater than operator>
            | <less than or equals operator>
            | <greater than or equals operator>
<literal> ::= <signed numeric literal>
            | <general literal>
<signed numeric literal> ::=
    [<sign> ] <unsigned numeric literal>
<general literal> ::= <character string literal>
                    | <datetime literal>
                    | <boolean literal>
                    | <geography literal
<boolean literal> ::= TRUE
                    | FALSE
                    | UNKNOWN
<routine invocation> ::= | <geoop name><georoutine argument list>
                         | <relgeoop name><relgeoop argument list>
                         | <routine name><argument list>
<routine name> ::= < attribute name>
<geoop name> ::= EQUAL | DISJOINT | INTERSECT | TOUCH | CROSS
               | WITHIN | CONTAINS |OVERLAP | RELATE
<relgeoop name> ::= DWITHIN | BEYOND
<argument list> ::=
   <left paren> [<positional arguments>]  <right paren>
<positional arguments> ::=
   <argument> [ { <comma> <argument> }... ]
<argument> ::= <literal> | <attribute name>
<georoutine argument list> ::=
<left paren><attribute name><comma><geometry literal><right paren>
<relgeoop argument list> ::= <left paren>
   <attribute name><comma><geometry literal><comma><tolerance>
   <right paren>
<tolerance> ::=
   <unsigned numeric literal><comma><distance units>
<distance units> ::= = "feet" | "meters" | "statute miles" |
                       "nautical miles" | "kilometers"
/*this set of units is just an example. The real list of distance unit
must be developed*/
<geometry literal> := <Point Tagged Text>
                    | <LineString Tagged Text>
                    | <Polygon Tagged Text>
                    | <MultiPoint Tagged Text>
                    | <MultiLineString Tagged Text>
                    | <MultiPolygon Tagged Text>
                    | <GeometryCollection Tagged Text>
                    | <Envelope Tagged Text>
<Point Tagged Text> := POINT <Point Text>
<LineString Tagged Text> := LINESTRING <LineString Text>
<Polygon Tagged Text> := POLYGON <Polygon Text>
<MultiPoint Tagged Text> := MULTIPOINT <Multipoint Text>
<MultiLineString Tagged Text> := MULTILINESTRING <MultiLineString Text>
<MultiPolygon Tagged Text> := MULTIPOLYGON <MultiPolygon Text>
<GeometryCollection Tagged Text> :=
   GEOMETRYCOLLECTION <GeometryCollection Text>
<Point Text> := EMPTY | <left paren> <Point> <right paren>
<Point> := <x><space><y>
<x> := numeric literal
<y> := numeric literal
<LineString Text> := EMPTY
| <left paren><Point>{<comma><Point >}…<right paren>
<Polygon Text> := EMPTY
| <left paren><LineString Text>{<comma><LineString Text> }…<right paren>
<Multipoint Text> := EMPTY
| <left paren><Point Text>{<comma><Point Text >}…<right paren>
<MultiLineString Text> := EMPTY
| <left paren><LineString Text>{<comma><LineString Text>}…<right paren>
<MultiPolygon Text> := EMPTY
| <left paren><Polygon Text>{<comma><Polygon Text>}…<right paren>
<GeometryCollection Text> := EMPTY
| <left paren><Geometry Tagged Text>{<comma><Geometry Tagged Text>}…
<right paren>
<Envelope Tagged Text> ::= ENVELOPE <Envelope Text>
<Envelope Text> := EMPTY
| <left paren><WestBoundLongitude><comma>
              <EastBoundLongitude><comma>
              <NorthBoundLatitude><comma>
              <SouthBoundLatitude>< <right paren>
<WestBoundLongitude> := numeric literal
<EastBoundLongitude> := numeric literal
<NorthBoundLatitude> := numeric literal
<SouthBoundLatitude> := numeric literal

<date-time>    ::= <full-date> "T" <UTC-time>

<full_date>    ::= <date-year> "-" <date-month> "-" <date-day>
<date-year>    ::= <digit><digit><digit><digit>
<date-month>   ::= <digit><digit>
<date-day>     ::= <digit><digit>

<UTC-time>     ::= <time-hour> ":" <time-minute> ":" <time-second> "Z"
<time-hour>    ::= <digit><digit>
<time-minute>  ::= <digit><digit>
<time-second>  ::= <digit><digit>[.<digit>...]

<duration>     ::= "P" <dur-date> | <dur-time>
<dur-date>     ::= <dur-day> | <dur-month> | <dur-year> [<dur-time>]
<dur-day>      ::= <digit>... "D"
<dur-month>    ::= <digit>... "M" [<dur-day>]
<dur-year>     ::= <didit>... "Y" [<dur-month>]
```

```
<dur-time>     ::= "T" <dir-hour> | <dur-minute> | <dur-second>
<dur-hour>     ::= <digit>... "H" [<dur-minute>]
<dur-minute>   ::= <digit>... "M" [<dur-second>]
<dur-second>   ::= <digit>... "S"

<period>       ::= <date-time> "/" <date-time>
                 | <date-time> "/" <duration>
                 | <duration> "/" <date-time>
```

6.2.3 Extending the Common Catalogue Query Language

The Common Catalogue Query Language BNF can be extended by adding new predicates, operations, and datatypes. The following discussion is an example of extending the BNF to include a CLASSIFIED-AS operator using the patterns identified in **OASIS/ebXML Registry Services Specification v2.5.** This extension could appear in a protocol binding or an Application Profile.

This specification makes no assumptions about how taxonomies are maintained in a catalogue, or how records are classified according to those taxonomies. Instead, this specification defines a routine, CLASSIFIED-AS, in order to support classification queries based on taxonomies.

The CLASSIFIED-AS routine takes three arguments. The first argument is the abstract entry point who classification is being checked The second argument is the key name string that represents a path expression in the taxonomy. The last argument is the key value string that represents the corresponding path expression containing key values that are the targets of the query. In both cases, the first element of the path expression for the key name argument and key value arguments must be the name of the taxonomy being used. The normal wildcard matching characters, '_' for a single character and '%' for zero or more characters, may be used in the key value expression which is the last argument of the CLASSIFIED_AS routine.

The following set of productions define the CLASSIFIED-AS routine.

```
/* The following example:                                            */
/*                                                                   */
/* RECORD CLASSIFIED AS KEYNAME='/GeoClass/Continent/Country/State'  */
/*      KEYVALUE='/GeoClass/North America/%/Ontario'                 */
/*                                                                   */
/* Will find all records in all the Ontario's in North America.      */

The following are the required BNF specializations:

<classop argument list> ::= <left paren> <entry_point> <comma>
        <keyname path> <comma><keyname value> <right paren>

<entry_point> ::= <identifier>[1]
<keyname path> ::= <solidus><ClassificationName><solidus><identifier>
                   [<solidus><identifier>]…
<classop name> ::= CLASSIFIED_AS
<Classification Name> ::= <identifier>
```

[1] The identifier is a valid value of a Type (core queryable properties) that represents an entire catalogue record. For example, the value DATASET may be used to represent a record in a data catalogue or the value SERVICE may be used to represent a record in a service catalog. In addition this profile has added the value RECORD to indicate all records regardless of type.

```
<keyvalue path> ::= <solidus><path element>[<solidus><path element>]…
<path element> ::= <character pattern>

 <routine invocation> ::= | <geoop name><georoutine argument list>
                         | <relgeoop name><relgeoop argument list>
                         | <routine name><argument list>
                         | <classop><classop argument list>
```

Consider the following example:

```
CLASSIFIED_AS(RECORD, '/GeoClass/Continent/Country/State',
'GeoClass/NorthAmerica/%/Ontario')
```

In this example, we are searching records classified according to the **GeoClass** taxonomy. Specifically, we are looking for all catalogue records classified as *Continent=NorthAmerica*, *Country=**any country*** and *State=Ontario*. Notice how the wildcard character '%' is used to search for any Country node.

Here is the same example encoded using XML:

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
  <ClassifiedAs>
    <EntryPoint>RECORD/<EntryPoint>
      <KeyName>/GeoClass/Continent/Country/State</KeyName>
      <KeyValue>/GeoClass/NorthAmerica/%/Ontario</KeyValue>
  </ClassifiedAs>
</ogc:Filter>
```

In order for catalogue clients to be able to determine which taxonomies are available, a catalogue implementation should advertise the list of available taxonomies in its capabilities document. If a query is executed against a non-existent taxonomy, then an exception should be raised.

6.2.4 Query language realization

Many OGC service operations have the requirement to pass and process a query as a structure to perform a request. There are several query languages and messaging mechanisms identified within OGC specifications. Application Profiles should be explicit about the selected query languages and any features peculiar to a scope of application. The following items should be addressed in the preparation of an Application Profile with respect to query language support:

a) Support for "abstract" query against well-known queryable entry points (e.g. OGC Core). Some specifications promote or require the exposure of well-known field-like objects as common search targets (queryables), allowing interrogation of a service without prior negotiation on information content. The mandatory queryable attributes which must be recognised by all OGC Catalogue Services is discussed in Subclause 6.3.2.

b) Selection of a query language. Some specifications describe one or more query languages that can be supported. Identify the name and version of required query language(s) anticipated by this Application Profile for use.

c) Supported data types (e.g. character, integer, coordinate, date, polygon) and operator types (e.g. inequality, proximity, partial string, spatial, temporal). Query languages may be restricted in their implementation or extended with functions not described in the base specification. This narrative should provide lists or reference documents with the enumerated data types and operator types required by this Application Profile. In addition, any description of special techniques (e.g. supporting joins or associations) that are expected by an Application Profile should be described.

The following subclause uses the Filter Encoding of the BNF to illustrate some of the issues involved in a specific realization of the Common Catalogue Query Language BNF.

6.2.5 OGC filter syntax

6.2.5.1 Introduction

The XML implementation of the BNF in Subclause 6.3.2 may be found in OGC document 02-059, Filter Encoding Implementation Specification. The intent of the XML encoding of the OGC common query language is that it be easily parsable using readily available XML parsers and be easily translatable into a target predicate language such as a SQL where clause or an Xquery predicate.

6.2.5.2 Provider functional extensibility

One feature of the OGC common query language and the XML implementation is that the predicate language is functionally extensible. This means that functions may be added to the filter predicate language without having to change the underlying schema. The relevant schema fragment from 02-059 is:

```
<xsd:element name="Function"
             type="ogc:FunctionType"
             substitutionGroup="ogc:expression"/>
<xsd:complexType name="FunctionType">
    <xsd:complexContent>
      <xsd:extension base="ogc:ExpressionType">
        <xsd:sequence>
          <xsd:element ref="ogc:expression"
                       minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

According to the schema fragment, any function may added to the filter predicate language simply by specifying its name and including zero or more ogc expressions as content of the **<Function>** element which represent the arguments of the function. The following example shows how a function may be called using the filter syntax:

```
<Function name=”MAX”>
   <PropertyName>DEPTH</PropertyName>
</Function>
```

In this example, the MAX() function is invoked to find the maximum value of the property DEPTH.

Any function may be called using the filter syntax as long as the function is advertised in the filter capabilities section (Clause 16 of OGC 02-059) of an OGC capabilities document.

6.2.5.3 Precedence

The XML notation does not provide parentheses to indicate operator precedence as specified in the BNF. The Filter Specification uses the nested structure of the XML notation to indicate this relationship.

In this example, a more complex scalar predicate is encoded using the logical operators AND and OR. The example is equivalent to the expression:

```
((FIELD1=10 OR FIELD1=20) AND (STATUS="VALID").
<Filter xmlns="http://http://www.opengis.net/ogc"
        xmlns:foo="http://foo/">
  <And>
    <Or>
      <PropertyIsEqualTo>
        <PropertyName>foo:FIELD1</PropertyName>
        <Literal>10</Literal>
      </PropertyIsEqualTo>
      <PropertyIsEqualTo>
         <PropertyName>foo:FIELD1</PropertyName>
         <Literal>20</Literal>
      </PropertyIsEqualTo>
    </Or>
    <PropertyIsEqualTo>
      <PropertyName>foo:STATUS</PropertyName>
      <Literal>VALID</Literal>
    </PropertyIsEqualTo>
  </And>
</Filter>
```

6.2.5.4 Tight and loose queries

The following examples show the implementation of tight and loose queries using the Filter Grammer.[2] In both cases the query is directed to a federation of image catalogs that are compliant to an application profile where an optional searchable attribute “cloudcover” is defined as the percentage of the target obscured by clouds.

In Case 1 the investigator wants to get only images he is sure he can use so he requests only images where the cloud cover is less than 5 percent. This is the normal case for querying known schema.

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
   <ogc:PropertyIsLessThan>
      <ogc:PropertyName>cloudcover</ogc:PropertyName>
      <ogc:Literal>5</ogc:Literal>
   </ogc:PropertyIsLessThan>
</ogc:Filter>
```

In Case 2 the investigator is aware that 5% cloud cover is very rare over the target area but he requires only images with less than 5% cloud cover. In this case he wants any images he might be able to use so he requests images which meets his criteria and images where cloud cover in unknown because the catalogue has chosen not to include cloudcover in its searchable attribute set.

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
```

[2] This capability is included in version 1.1 of the OGC Filter Specification which is currently under development

```
   <ogc:Or>
      <ogc:PropertyIsLessThan>
         <ogc:PropertyName>cloudcover</ogc:PropertyName>
         <ogc:Literal>50</ogc:Literal>
      </ogc:PropertyIsLessThan>
      <ogc:PropertyValueDoesNotExist>
         <ogc:PropertyName>cloudcover</ogc:PropertyName>
      </ogc:PropertValueDoesNotExist>
   </ogc:Or>
</ogc:Filter>
```

6.3 Core catalogue schema

6.3.1 Introduction

Metadata structures, relationships, and definitions -- known as conceptual schemas -- exist for multiple information communities. For the purposes of interchange of information within an information community, a metadata schema may be defined that provides a common vocabulary which supports search, retrieval, display, and association between the description and the object being described. Although this specification does not require the use of a specific schema, the adoption of a given schema within an information-sharing community ensures the ability to communicate and discover information.

The geomatics standardization activity under ISO Technical Committee 211 includes a formal schema for geospatial metadata that is intended to apply to all types of information. This metadata standard, ISO 19115:2003 includes a proposal for core metadata elements in common use. ISO Draft Technical Specification 19139 defines a formal encoding and structure of ISO metadata for exchange. Where a catalogue service advertises such application schemas, catalogues that handle geographic dataset descriptions should conform to published metadata standards and encodings, e.g. ISO 19115:2003, and support XML encoding per ISO 19139 or profiles thereof. Service metadata elements should be consistent with ISO 19119.

6.3.2 Core queryable properties

The goal of defining core queryable properties is to enable simple cross-profile discovery, where the same queries can be executed against any catalogue service without modification and without detailed knowledge of the catalogue's information model. This requires a set of general metadata properties that can be used to characterise any resource. The Binding profile must further specify the ID based on the native platform ID types. Binding protocols and application profiles should realize these abstract queryables in their core queryable schemas

Application Profiles may choose to use a single <comma separated list> for a compound datatype or may label each sub-element for clarity and order flexibility.

The current list is shown in Tables 1, 2 and 3.

Table 1 — Common queryable elements

Name	Definition	Data type
Subject [a]	The topic of the content of the resource [b]	CharacterString
Title [a]	A name given to the resource	CharacterString
Abstract [a]	A summary of the content of the resource	CharacterString
AnyText	A target for full-text search of character data types in a catalogue	CharacterString
Format [a]	The physical or digital manifestation of the resource	Codelist: application/xml, text/html, text/plain
Identifier [a]	An unambiguous reference to the resource within a given context	Identifier
Modified [c]	Date on which the resource was last changed	Date-8601
Type [a]	The nature or genre of the content of the resource. Type can include general categories, genres or aggregation levels of content.	Codelist: Dataset, DatasetCollection, Service
BoundingBox [d]	A bounding box for identifying a geographic area of interest	BoundingBox, See Table 2
CRS	Coordinate Reference System (Authority and ID) for the BoundingBox	Identifier [e]
Association	Complete statement of a one-to-one relationship	Association, See Table 3

a Dublin Core Metadata Element Set, version 1.1:ISO Standard 15836-2003 (February 2003)

b Typically, a Subject will be expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

c DCMI metadata term <http://dublincore.org/documents/dcmi-terms/>.

d Same semantics as EX_GeographicBoundingBoxclass in ISO 19115.

e If not supplied, the BoundingBox CRS is a Geographic CRS with the Greenwich prime meridian.

Table 2 — Composition of compound element “BoundingBox”

Name	Definition	Data type
WestBoundLongitude	Western-most coordinate of the limit of the dataset extent, expressed in longitude in decimal degrees (positive east)	numeric
SouthBoundLatitude	Southern-most coordinate of the limit of the dataset extent, expressed in latitude in decimal degrees (positive north)	numeric
EastBoundLongitude	Eastern-most coordinate of the limit of the dataset extent, expressed in longitude in decimal degrees (positive east)	numeric
NorthBoundLatitude	Northern-most, coordinate of the limit of the dataset extent, expressed in latitude in decimal degrees (positive north)	numeric

Table 3 — Composition of compound element "Association"

Name	Definition	Data type
Target	Referenced resource	Identifier
Source	Referencing resource	Identifier
Relation	The name of the description of the relationship	CharacterString or Identifier

6.3.3 Core returnable properties

A set of core properties returned from a metadata search is encouraged to permit the minimal implementation of a catalogue service independent of a companion application profile and to permit the use of metadata returned from different systems and protocol bindings. The core metadata is returned as a request for the Common Element Set. The Common Element Set is a new group of public metadata elements, expressed using the nomenclature and syntax of Dublin Core Metadata, ISO 15836, but including proper qualifiers to disambiguate certain fields of information. Table 4 provides some interpretation of Dublin Core elements and qualifiers in the context of metadata for geospatial data and services.

Table 4 — List of common returnable properties

Dublin Core metadata element name	**Term used in OGC queryables**	**Description** *("Resource" means the thing being described in the metadata)*
dc:title	Title	A name given to the resource. Also known as "Name".
dc:creator		An entity primarily responsible for making the content of the resource.
dc:subject	Subject	A topic of the content of the resource. This is a place where a Topic Category or other taxonomy could be applied.
dct:abstract	Abstract	An account of the content of the resource. This is also known as the "Abstract" in other aspects of OGC, FGDC, and ISO metadata.
dc:publisher		An entity responsible for making the resource available. This would equate to the Distributor in ISO and FGDC metadata.
dc:contributor		An entity responsible for making contributions to the content of the resource.
dc:date	Modified	A date of a creation or update event of the metadata resource.
dc:type	Type	The nature or genre of the content of the resource.
dc:format	Format	The physical or digital manifestation of the resource.
dc:identifier	Identifier	An unambiguous reference to the resource within a given context.
dc:source	Source	A reference to a resource from which the present resource is derived.
dc:language		A language of the intellectual content of the resource.
dc:relation	Relation, Source, Target	A reference to a related resource.
dct:spatial	Envelope, CRS	The spatial extent or scope of the content of the resource.
dc:rights		Information about rights held in and over the resource.

The core elements are recommended for a response but do not need to be populated. The support for a common syntax for the returnable properties as a "common" Summary Element Set is defined in the protocol binding clauses. All data types for the above elements except for date are CharacterString The following is an example of a 'qualified' core metadata set expressed in XML as per the guidance of the Dublin Core Metadata Initiative.

```
<?xml version="1.0" encoding="UTF-8"?>
<cat:Record
  xmlns:cat="http://www.opengis.net/cat"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dct="http://purl.org/dc/terms/"
  xmlns:csw="http://www.opengis.net/cat/csw"
  xmlns:iso19115TopicCategory="http://www.isotc211.org/"
  xmlns:dcmiBox="http://dublincore.org/documents/2000/07/11/dcmi-box/">

   <dc:creator>U.S. Geological Survey</dc:creator>
   <dc:creator>U.S. Geological Survey</dc:creator>
      <dc:contributor>State of Texas</dc:contributor>
      <dc:publisher>U.S. Geological Survey</dc:publisher>
      <dc:subject>Elevation, Hypsography, and Contours</dc:subject>
      <iso19115TopicCategory>Elevation</iso19115TopicCategory>
      <dct:abstract>Elevation data collected for the National Elevation
Dataset (NED) based on 30m horizontal and 15m vertical accuracy.
      </dct:abstract>
      <dc:identifier>f264-77d2-09ce-aa39-f0f0</dc:identifier>
      <dc:relation>g</dc:relation>
      <dc:source>h</dc:source>
      <dc:rights>i</dc:rights>
      <dc:format>j</dc:format>
      <dc:type>Service</dc:type>
      <dc:title>National Elevation Mapping Service for Texas</dc:title>
      <dct:modified>2004-03-01</dct:modified>
      <dct:spatial>
        <Box projection="EPSG:4326" name="Geographic">
          <northlimit>34.353</northlimit>
          <eastlimit>-96.223</eastlimit>
          <southlimit>28.229</southlimit>
          <westlimit>-108.44</westlimit>
        </Box>
      </dct:spatial>
      <dc:language>en</dc:language>
</cat:Record>
```

6.3.4 Information structure and semantics

Some services that implement OGC Implementation Specifications expect a rigid syntax for the information resources to be returned, whereas others do not. This subclause allows an Application Profile to be specific about what information content, syntax, and semantics are to be communicated over the service. The following items should be addressed in an Application Profile.

a) Identify information resource types that can be requested. In the case of a catalogue service, the information resources being described by the metadata may include geographic data, imagery, services, controlled vocabularies, or schemas among a wide variety of possible types. This subclause allows the community to specify or generalise the resource types being described in metadata for their scope of application.

b) Identify a public reference for the information being returned by the service (e.g. ISO 19115:2003 "Geographic Information – Metadata "). Include any semantic resources including data content model, dictionary, feature type catalog, code lists, authorities, taxonomies, etc.

c) Identify named groups of properties (element sets) that may be requested of the service (e.g. "brief," "summary," or "full") and the valid format (syntax) for each element set. Identify valid schema(s) with respect to a given format to assist in the validation of response messages.

d) Specialise the core queryable properties list by making some optional queryable attributes mandatory, deleting other optional attributes and adding queryabable attributes that should be standard across all profile users

e) Optional mapping of queryable and retrievable properties against other public metadata models or tags.

f) Expected response/results syntax and content Message syntax and schemas (e.g. brief/full, individual elements).

7 General catalogue interface model

7.1 Introduction

The General Catalogue Interface Model provides a set of abstract service interfaces that support the discovery, access, maintenance and organization of catalogues of geospatial information and related resources. The interfaces specified are intended to allow users or application software to find information that exists in multiple distributed computing environments, including the World Wide Web (WWW) environment. All behaviour requiring sessions is expressed by a dynamic model of conversation state and state transitions. The model expresses the states and messages that trigger the changes in state.

Implementation design guidance is included in the protocol binding clauses of this specification. Each protocol binding includes a mapping from the general interfaces, operations, and parameters specified in this clause to the constructs available in a chosen protocol. Application profiles are intended to further document implementation choices.

An Application Profile is predicated on the existence of one protocol binding in the base specification. In the case of the Catalogue Services Specification, a profile could reference CORBA, Z39.50, or HTTP protocol bindings. In most, but not all, protocol bindings, there may be restrictions or refinements on implementation of the General Model agreed within an implementation community. This subclause provides an overview of the portions of the General Catalogue Model that are realised by Protocol Bindings and Application Profiles.

Figure 3 shows the Reference Architecture assumed for development of the OGC Catalogue Interface. The architecture is a multi-tier arrangement of clients and servers. To provide a context, the architecture shows more than just catalogue interfaces. The bold lines illustrate the scope of OGC Catalogue and Features interfaces.

The Application Client shown in Figure 3 interfaces with the Catalogue Service using the OGC Catalogue Interface. The Catalogue Service may draw on one of three sources to respond to the Catalogue Service request: a Metadata Repository local to the Catalogue Service, a Resource service, or another Catalogue Service,. The interface to the local Metadata Repository is internal to the Catalogue Service. The interface to the Resource service can be a private or OGC Interface. The interface between Catalogue Services is the OGC Catalogue Interface. In this case, an Catalogue Service is acting as both a client and server. Data returned from an OGC Catalogue Service query is processed by the requesting Catalogue Service to return the data appropriate to the original Catalogue request. See Annex B for more about Distributed Searching.

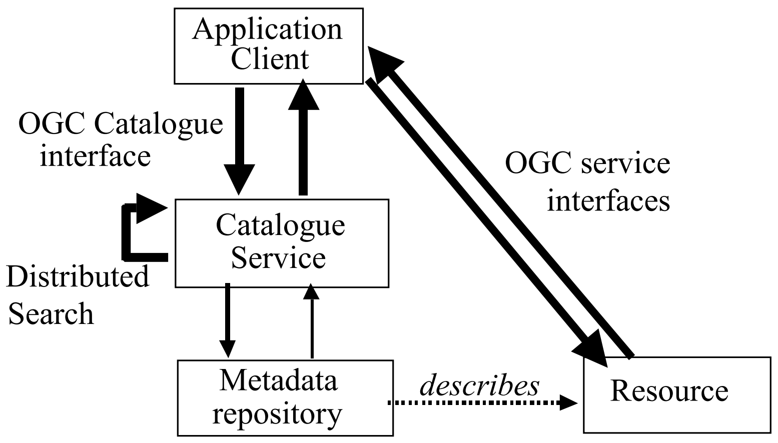

Figure 3 — Reference model architecture

7.2 Interface definitions

7.2.1 Overview

Figure 4 is a general UML model of OGC catalogue service interfaces, in the form of a class diagram. Operation signatures have been suppressed in this figure for simplicity but are described in detail below. This model shows the Catalogue Service class plus five other classes with which that class are associated. A Catalogue Service is a realization of an OGC Service. Each instance of the Catalogue Service class is associated with two or more of these other classes, depending on the abilities included in that service instance. Each of these other classes defines one or several related operations that can be included in a Catalogue Service class instance. The Catalogue Service class directly includes only the serviceTypeID attribute, with a fixed value for the service type.

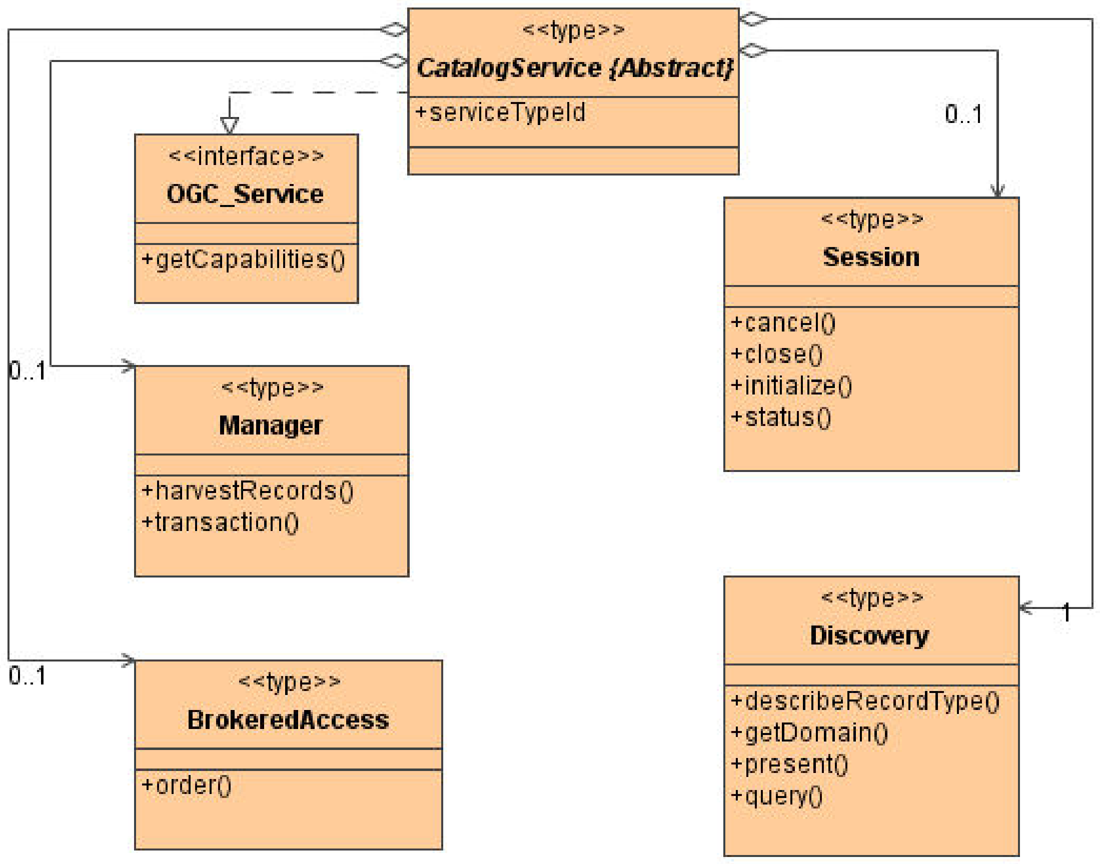

Figure 4 — General OGC catalogue UML static model

In Figure 4 an instance of the `CatalogueService` type is a composite object that is a high-level characterization of a catalogue service. Its constituent objects are themselves components that provide functional behaviours to address particular areas of concern. A protocol binding may realise specific configurations of these components to serve different purposes (e.g. a read-only catalogue for discovery, a transactional catalogue for discovery and publication, or a 'stateful' catalogue that supports session management).

The associated classes shown in this figure are mandatory or optional for implementation as indicated by the association multiplicity in the UML diagram. Therefore, a compliant catalogue service shall implement the OGC_Service, CatalogueService, and Discovery classes. An application profile or protocol binding can implement additional classes associated with the Catalogue Service class. A catalogue implementation shall recognise all operations defined within each included class, and shall generate a message indicating when a particular operation is not implemented.

The protocol binding clauses of this specification provide more detail on the implementation of these conceptual interfaces. For example, the names of the classes and operations in this general UML model are changed in some of the protocol bindings. The names of some operation parameters are also changed in some protocol bindings.

Application Profiles may further specialise the implementation of these interfaces and their operations, including adding classes. In general, however, the interfaces and operations described here shall have the same semantics and granularity of interaction regardless of the protocol binding used.

The Catalogue Service class can be associated with the:

a) OGC_Service class, which provides the getCapabilities operation that retrieves catalogue service metadata. This class is always realised by the Catalogue Service class, and is thus always implemented by a Catalogue Service implementation.

b) Discovery class, which provides four operations for client discovery of resources registered in a catalogue. This class has a required association from the Catalogue Service class, and is thus always implemented by a Catalogue Service implementation. The "query" operation searches the catalogued metadata and produces a result set containing references to all the resources that satisfy the query. This operation optionally returns metadata for some or all of the found result set. The "present" operation returns selected metadata for some or all of the resources referenced in a specific previous result set. The describeRecordType operation retrieves the type definition used by metadata of one or more registered resource types. The getDomain operation retrieves information about the valid values of one or more named metadata properties.

c) Session class, which provides four operations for interactive sessions between a server and a client. This class has an optional association from the Catalogue Service class; this interface is implemented by the Catalogue Service implementation. The "initialise" operation initiates an interactive session, and the "close" operation terminates a session. The "status" operation retrieves the current status of a previously initiated operation, and the "cancel" operation terminates a previously initiated operation.

d) Manager class, which provides two operations for inserting, updating, and deleting the metadata by which resources are registered in a catalogue. This class has an optional association from the Catalogue Service class; this interface is implemented by the Catalogue Service implementation. The transaction operation performs a specified set of "insert", "update", and "delete" actions on metadata items stored by a Catalogue Service implementation—this enables a "push" style of publication. The harvestRecords operation requests the Catalogue Service to retrieve resource metadata from a specified location, often on a regular basis—this behaviour reflects a 'pull' style of publication.

e) Brokered Access class, which provides the "order" operation for ordering an identified resource that is registered in a catalogue but is not directly accessible to the client. This class has an optional association from the Catalogue Service class; this interface is implemented by the Catalogue Service implementation.

The five classes that can be associated with the Catalogue Service class allow different OGC catalogue services to provide significantly different abilities. A particular protocol binding is used by each Application Profile and a particular set of these catalogue service classes is specified by each Application Profile.

Each of the catalogue classes is described further in the following subclauses. These subclauses discuss the operations and parameters of each operation in this general model. Specific protocol bindings or application profiles can define additional parameters. For example, the HTTP Protocol Binding adds the Service, Request, and Version parameters to all operation requests to be consistent with other OGC Web Services.

7.2.2 Catalogue Service class

The Catalogue Service class provides the foundation for an OGC catalogue service. The Catalogue Service class directly includes only the serviceTypeID attribute, as specified in Table 5. In most cases, this attribute will not be directly visible to catalogue clients.

Table 5 — Attribute of Catalogue Service class

Name	Definition	Data type	Multiplicity
serviceTypeID	Identification of catalogue service type	URI, as specified in IETF RFC 2396	One (Mandatory)

7.2.3 OGC_Service class

7.2.3.1 Introduction

The OGC_Service class allows clients to retrieve service metadata by providing the getCapabilities operation. This class is always realised by the Catalogue Service class, and is thus always implemented by a Catalogue Service instance.

NOTE This getCapabilities operation corresponds to CatalogueService.explainServer operation in OGC Catalogue version 1.1.1.

7.2.3.2 getCapabilities operation

The getCapabilities operation is more completely specified in Figure 5.

Table 6 — Definition of getCapabilities operation

Definition	Allows clients to retrieve service metadata describing Catalogue Service instance
Receives	Optional identifier(s) of requested parts of the complete service metadata document
Returns	Service metadata document for Catalogue Service instance. Some document contents depend on the set of classes that are associated with the Catalogue Service class, as defined by the specific protocol binding, and on other details of that protocol binding. Other document contents depend on the types of data defined by the specific application profile, and on other details of that profile.
Exceptions	Invalid Parameter Value, Missing Parameter Value
Pre-conditions	None
Post-conditions	Service metadata document returned to requesting client, either complete or including selected parts

Figure 5 provides a UML model of the OGC_Service class that shows the complete signature of the getCapabilities operation, plus classes for the getCapabilities operation request and the ServiceMetadata operation response. The abstract GetCapabilities and Service Metadata classes are specialised by each service that uses the OGC_Service class. The detailed contents of both the CatalogGetCapabilities and CatalogueServiceMetadata classes depend on the protocol binding, and perhaps also on the Application Profile, and are thus not detailed here.

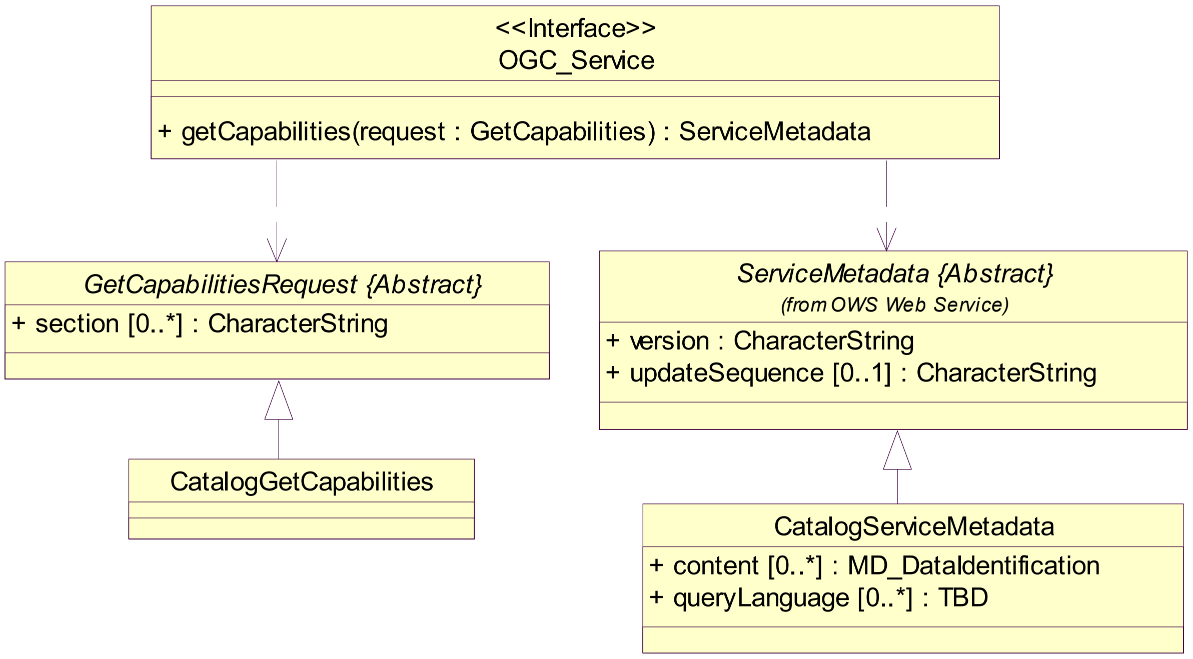
<<Interface>>
OGC_Service
+ getCapabilities(request : GetCapabilities) : ServiceMetadata
GetCapabilitiesRequest {Abstract}
+ section [0..*] : CharacterString
ServiceMetadata {Abstract}
(from OWS Web Service)
+ version : CharacterString
+ updateSequence [0..1] : CharacterString
CatalogGetCapabilities
CatalogServiceMetadata
+ content [0..*] : MD_DataIdentification
+ queryLanguage [0..*] : TBD

Figure 5 — getCapabilities operation UML static model

The GetCapabilities operation request includes one “section” attribute listed and defined in Table 7.

Table 7 — UML attibute in getCapabilities operation request

Name	Definition	Data type and value	Optionality and use
section	Name of requested section in complete service metadata document	Character String type, not empty Allowed values specified by each Application Profile	Zero or more (Optional) Return complete service metadata document when omitted

The normal GetCapabilities operation response is a service metadata document that includes the “section” attibutes listed and defined in Table 8, as selected by the “section” attribute in the operation request.

Table 8 — UML attibutes in getCapabilities operation normal response

Name	Definition	Data type	Optionality and use
ServiceIdentification	Metadata about this specific server	SV_ServiceIdentification in ISO 19119	Zero or one (Optional) Include when requested
ServiceProvider	Metadata about the organization operating this server	SV_ ServiceProvider in ISO 19119	Zero or one (Optional) Include when requested
OperationMetadata	Metadata about an operations specified by this service, including the URL(s) for operation requests	SV_OperationMetadata in ISO 19119	Zero or more (Optional) Include when requested Repeated for each operation implemented by this server
Content	Metadata about a collection or type of resource cataloged by this server	MD_DataIdentification in ISO 19115 (adapted)	Zero or more (Optional) Include when requested Repeated for each collection and type of resources cataloged
QueryLanguage	Metadata about a query lanquage supported by this server, specifying the query abilities implemented	Character string	Zero or more (Optional) Include when requested Repeated for each query lanquage implemented by this server

NOTE 1 The term "Capabilities XML" document was previously used for what is here called "service metadata" document. The term "service metadata" is now used because it is more descriptive and is compliant with OGC Abstract Specification Topic 12 (ISO 19119).

NOTE 2 This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.4 Discovery class

7.2.4.1 Introduction

The Discovery class allows clients to discover resources registered in a catalogue, by providing four operations named "query", "present", describeRecordType, and getDomain. This class has a required association from the Catalogue Service class, and is thus always implemented by all Catalogue Service implementations. The Session class can be included with the Discovery class, in associations with the Catalogue Service class.

The "query" and "present" operations may be executed in a session or stateful context. If a session context exists, the dynamic model in Subclause 7.4.3 defines the states of the session and the allowed transitions between states. When the "query" and "present" state does not include a session between a server and a client, any "memory or shared information" between the client and the server may be based on private understandings or features available in the protocol binding. The describeRecordType and getDomain operations do not require a session context. If a session context exists, the simple request-response dynamic model shown in Subclause 7.4.4 would apply.

7.2.4.2 "query" operation

The "query" operation is more completely specified in Table 9. Figure 6 provides a UML model of the "query" operation that shows the complete Discovery class with the QueryRequest and QueryResponse classes and the classes they use. This UML diagram does not detail the <<CodeList>> stereotyped classes named ResourceType, QueryScope, ResultType, ElementSet, ResponseSchema, ReturnFormat, SortOrder, and QueryLanguage. The operation request includes the attibutes and association role names listed and defined in Table 10 through Table 13. The normal operation response includes the attibutes and association role names listed and defined in Table 14.

NOTE The query operation corresponds to Discovery.query in OGC Catalogue version 1.1.1.

Table 9 — Definition of "query" operation

Definition	Allows clients to ask a catalogue to execute a query that searches the catalogued metadata and produces a result set containing (zero or more) references to all the registered resources that satisfy the query. The server may maintain the result set for subsequent retrieval requests.
Receives	Specifications of query constraints and of metadata to be returned
Returns	Number of items in result set, and/or selected metadata for some or all of the result set. The client can specify the maximum number of records for which metadata is returned. When metadata return is requested, the service implementation shall first sort the result set as specified by the client. Most of the metadata returned depends on the metadata requested and on the types of data defined by the specific Application Profile.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Nonexistant collection or type
Pre-conditions	The client knows the schema of the information model that the catalogue supports and can thus form valid query expressions.
Post-conditions	Response returned to requesting client, containing number of items in result set and/or selected metadata for some or all of result set

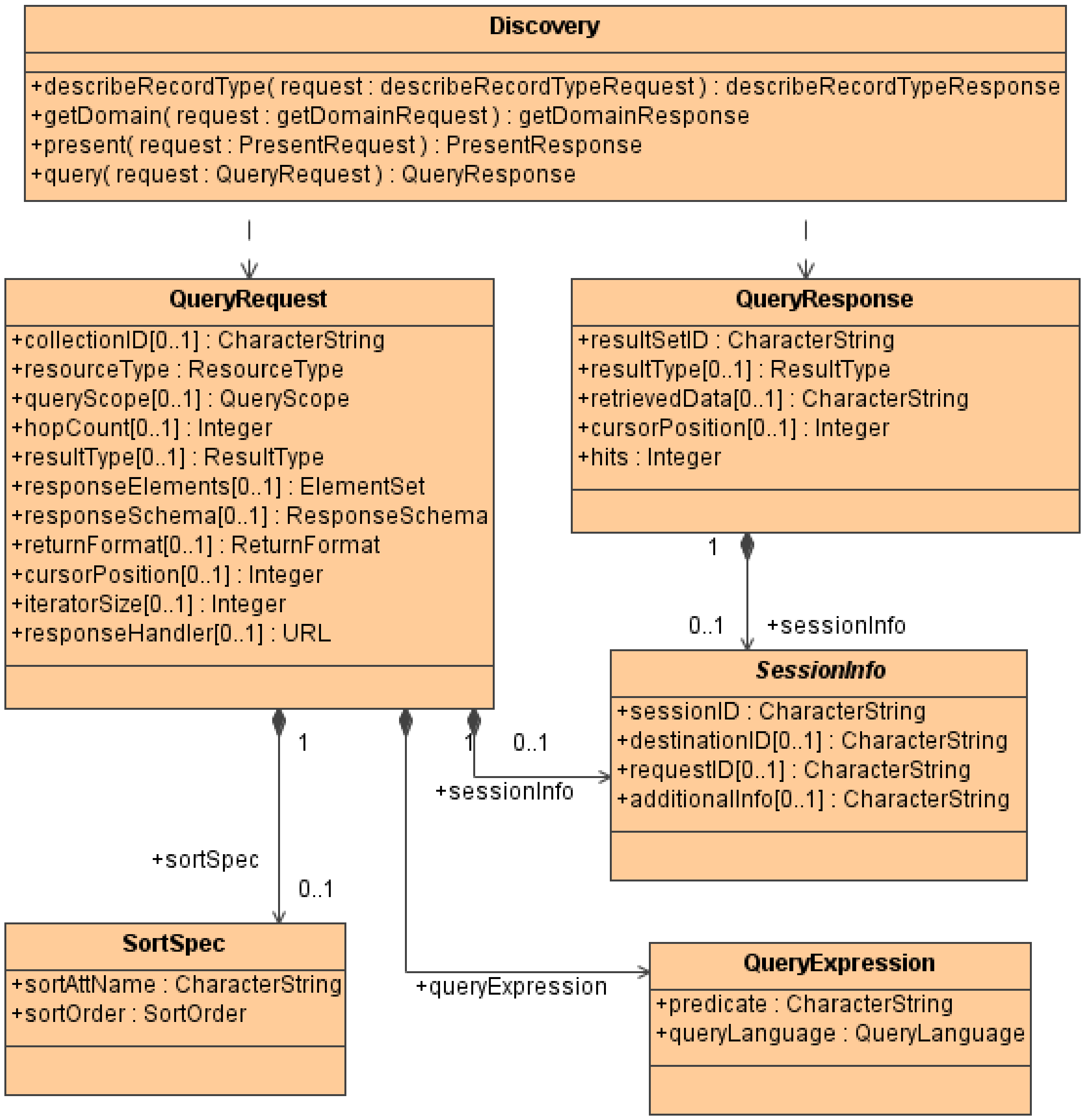
Discovery
+describeRecordType(request : describeRecordTypeRequest) : describeRecordTypeResponse
+getDomain(request : getDomainRequest) : getDomainResponse
+present(request : PresentRequest) : PresentResponse
+query(request : QueryRequest) : QueryResponse
QueryRequest
+collectionID[0..1] : CharacterString
+resourceType : ResourceType
+queryScope[0..1] : QueryScope
+hopCount[0..1] : Integer
+resultType[0..1] : ResultType
+responseElements[0..1] : ElementSet
+responseSchema[0..1] : ResponseSchema
+returnFormat[0..1] : ReturnFormat
+cursorPosition[0..1] : Integer
+iteratorSize[0..1] : Integer
+responseHandler[0..1] : URL
QueryResponse
+resultSetID : CharacterString
+resultType[0..1] : ResultType
+retrievedData[0..1] : CharacterString
+cursorPosition[0..1] : Integer
+hits : Integer
1
0..1
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString
1
1
0..1
+sessionInfo
+sortSpec
0..1
SortSpec
+sortAttName : CharacterString
+sortOrder : SortOrder
+queryExpression
QueryExpression
+predicate : CharacterString
+queryLanguage : QueryLanguage

Figure 6 — “query” operation UML static model

Table 10 — UML attributes and roles in "query" operation request

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
queryExpression	The query language and predicate expressing query constraints	QueryExpression, See Table 13	One (Mandatory)
collectionID	Specifies the search space for this query. Search space can be all catalogue holdings, a previously named result set, or a named subspace of the catalogue holdings	Character String type, not empty Specific values that may be referenced are application profile or protocol binding dependent	Zero or one (Conditional) Include when required by protocol binding, otherwise optional
resourceType	A catalogue may contain references to several different resource types. This parameter provides for the selection of one of those types for processing	CodeList type [a]	One (Mandatory)
queryScope	Scope of this query.	CodeList type with allowed values of "local" and "distributed"	Zero or one (Optional) Default value is"local"
hopCount	Maximum number of message hops before distributed search is terminated. Each catalogue decrements value by one when request is received, and does not forward request if hopCount=0.	Non-negative integer	Zero or one (optional) Default value is "2" Included only when queryScope has value "distributed"
resultType	Specifies how client wants result set presented and the behaviour of the catalogue as to when a response is sent	CodeList Type [b]	Zero or one (Conditional) Default values specified by protocol binding or application profile
ResponseElements[C]	Specifies set name or list of metadata elements to be returned in the context of a specific metadata structure	Either a list of elements as name/type pairs OR CodeList type named ElementSet with allowed values of "brief," "summary" "full" and "browse"	Zero or one (Optional) Default value is "brief"

Name	Definition	Data type and value	Optionality and use
ResponseSchema[C]	The name of the “well-known” or advertised (in the capabilities) schema of the response	Code List type with one mandatory value of “OGCCORE” that represents the core catalogue schema. Other values may be defined by application profiles. Examples of such values might be: “FGDC”, “ISO-19119”, ISO-19139”, ANZLIC	Zero or one (Optional). If the parameter is not specified then the default value is “OGCCORE”.
sortSpec	SortField provides sorting information to the server for formatting data returned to the client	SortSpec, See Table 11	Zero or one (Optional) Default is sorted on ID in descending order
returnFormat	Specifies format (MIME or Internet media type) for returning result set metadata	CodeList type XML HTML TXT	Zero or one (Optional) Default is “XML” Include when results to be returned
cursorPosition	First result set resource to be returned for this operation request	Positive integer	Zero or one (Optional) Default is “1” Include when results to be returned
iteratorSize	Specifies maximum number of result set resources to be returned	Non-negative integer	Zero or one (Optional) Default is “10” Include when results to be returned
responseHandler	Network location to which the response will be forwarded when operation has been completed, for asynchronour requests	URL	Zero or one (Optional) If not included, process request synchronously

a Values and defintitions of resourceType codes:

Data set – the lowest level packaging of Features that have been catalogued

Data set collection – a grouping of data sets that have commonality (ISO 19115: data set series)

Service – a set of interfaces that provide access to or operations on data (e.g. catalogue service)

b Values and definition of resultType codes and behaviours in session based environments:

validate - the QueryResponse is returned as soon as QueryRequest has been determined to be valid. Query processing continues after the QueryResponse is returned.. Reasons for failure are provided in the diagnostic of QueryResponse.

resultSetID - the QueryResponse is returned as soon as the resultSetID is available and the query has completed processing.

hits- the QueryResponse is returned as soon as the query has completed processing and the number of hits has been determined. Metadata records are not returned in the QueryResponse

results - the QueryResponse is returned as soon as the query has completed processing and the results have been formatted for return. Metadata records are returned in the QueryResponse

Name	Definition	Data type and value	Optionality and use
c The information model of this specification is the core catalogue schema defined in subclause 6.3. It represents the common part of the information model which all application profiles must support. This specifiecation only supports 'OGCCORE' as the value of the 'responseSchema' parameter and a value of "brief", "summary" or "full" for the value of the 'responseElements' parameter. Additional values for the responseSchema and responseElements parameters may be defined by application profiles.			

Table 11 — UML attibutes in SortSpec data type

Name	Definition	Data type and value	Optionality and use
sortAttName	Identifies the result set attribute type to be sorted on	Character String	Zero or one (Optional) Default is ID as defined by server
sortOrder	How the attributes are to be ordered by the sort	Code List type with allowed values of "ascending" and "descending"	Zero or one (Optional) Default is descending

Table 12 — UML attibutes in SessionInfo data type

Name	Definition	Data type and value	Optionality and use
sessionID	Unique identifier for this client/server session. The value is assigned in response to an "initiate" request. All further messages within that session will contain that identifier in the sessionID parameter.	CharacterString	One (Mandatory)
destinationID	Identifies the target for this message. It can identify a server, service, or a process within a service, or a list of services to which messages may be sent in a distributed environment.	CharacterString	Zero or one (Conditional) As required by protocol binding or distributed search environment
requestID	Uniquely identifies this message. In the case of a request message, this identifier can be used to monitor and control the processing resulting from the request message.	CharacterString	Zero or one (Optional)
additionalInfo	This parameter provides a means of passing additional data that may only be relevant within the context of a specific message exchange.	CharacterString	Zero or one (Optional)

Table 13 — UML attibutes in QueryExpression data type

Name	Definition	Data type and value	Optionality
queryLanguage	Specifies the predicate language and version used in a query expression	Code List, known values of "OGC_Common", "Filter, Type-1"	One (Mandatory)
predicate	The constraint expression for selecting entries from a catalogue	CharacterString	One (Mandatory)

Table 14 — UML attributes and role "query" operation normal response

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol bindingand/or result sets are persistent
resultSetID	Identifier of the Result Set generated for the query. Further query, present and cancel requests for this Result Set will supply this value through the collectionID parameter	Character String must be defined by protocol binding and may be further defined by application profile	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
resultType	How the server responded to the query request.	CodeList type with allowed values of "dataset", "datasetcollection" and "service	Zero or one (Optional)
retrievedData	A subset of the results of this query request, organised and formatted as specified in the presentation, messageFormat, and sortField parameters.	ReturnData Set of resource descriptions/records	Zero or one (Conditional) Include when resultType = Results
cursorPosition	Last result set resource returned for this operation request.	Positive integer	Zero or one (Conditional) Include when results returned
hits	Number of entries in the result set.	Non-negative integer	One (Mandatory)

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.4.3 "present" operation

The "present" operation is more completely specified in Table 15. Figure 7 provides a UML model of the "present" operation that shows the complete Discovery class with the PresentRequest and PresentResponse classes and the class they use. This UML diagram does not detail the <<CodeList>> stereotyped classes named PresentationSetType, ResponseSchema, ReturnFormat, and SortOrder. The operation request includes the attibutes and association role name listed and defined in Table 16. The normal operation response includes the attibutes and association role name listed and defined in Table 17.

Table 15 — Definition of “present” operation

Definition	Allows clients to retrieve selected metadata for some or all of the resources referenced in a specific previous result set or a list of resource identifiers. This operation can be used repetitively to retrieve more of the result set, each time retrieving metadata for a maximum number of the resources listed, starting at a specified position.
Receives	Specifications of sorting and of metadata to be returned, optionally including maximum number of records for which metadata is to be returned
Returns	Metadata document containing selected metadata for some or all of the specific result set, after it is sorted as specified by the client. Most of the metadata returned depends on the metadata requested and on the types of data defined by the specific Application Profile
Exceptions	Missing Parameter Value, Invalid Parameter Value , Unrecognized collection identifier.
Pre-conditions	Client has previously performed “search” operation, and the server has provided a result set identifier that the client can use to perform the present operation.
Post-conditions	Metadata document returned to requesting client, containing selected metadata for some or all of sorted result set

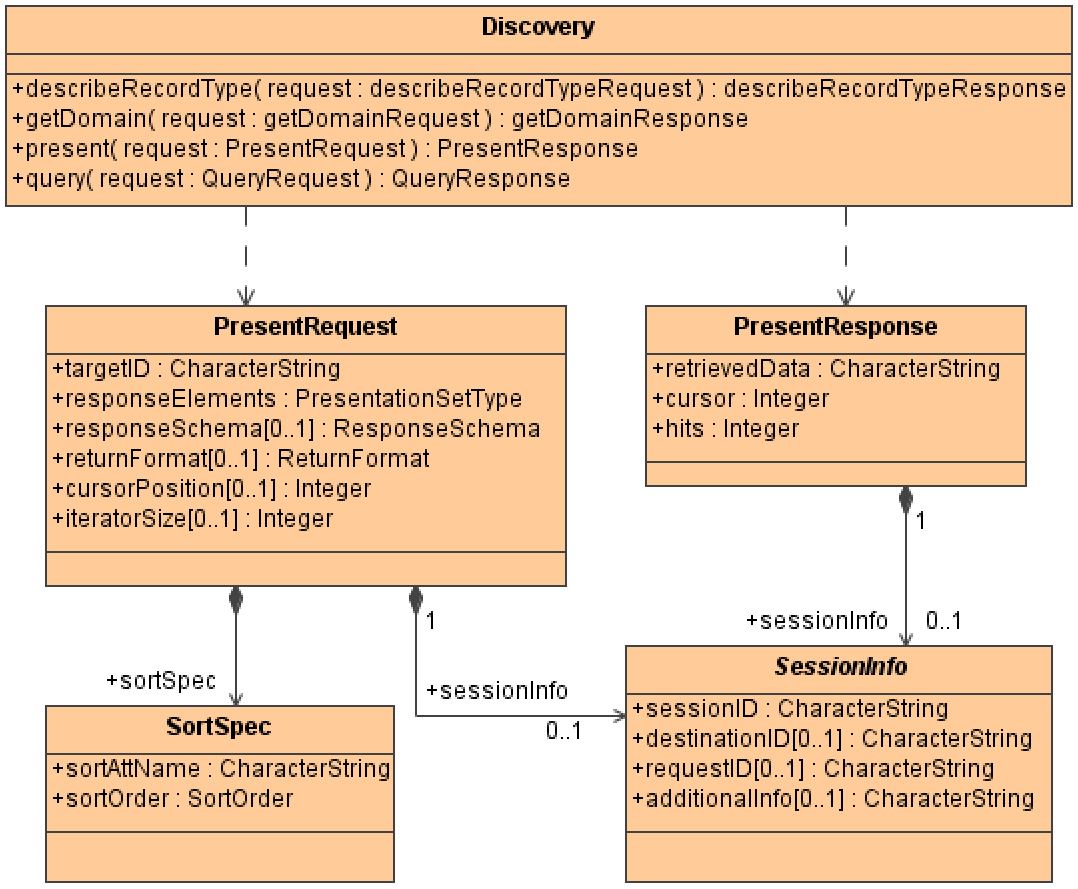
Discovery
+describeRecordType(request : describeRecordTypeRequest) : describeRecordTypeResponse
+getDomain(request : getDomainRequest) : getDomainResponse
+present(request : PresentRequest) : PresentResponse
+query(request : QueryRequest) : QueryResponse
PresentRequest
+targetID : CharacterString
+responseElements : PresentationSetType
+responseSchema[0..1] : ResponseSchema
+returnFormat[0..1] : ReturnFormat
+cursorPosition[0..1] : Integer
+iteratorSize[0..1] : Integer
PresentResponse
+retrievedData : CharacterString
+cursor : Integer
+hits : Integer
1
+sessionInfo
0..1
1
+sortSpec
+sessionInfo
0..1
SortSpec
+sortAttName : CharacterString
+sortOrder : SortOrder
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString

Figure 7 — “present” operation UML static model

Table 16 — UML attributes and role in “present” operation request

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol bindingand/or result sets are persistent
targetID	Unique identifier of result set from which resources to be returned or of resources to be returned	Either resultSetID OR list of resource IDs	One (Mandatory)
responseElements	Specifies set name or list of metadata elements to be returned in the context of a specific metadata structure.	Either a list of elements as name/type pairs OR Codelist type named PresentationSetType with allowed values of “brief,” “summary” “full” and “browse”	Zero or one (Optional) Default is “Brief” Include when results returned
responseSchema	The name of the “well-known” or advertised (in the capabilities) schema of the response	Code List type with allowed values of “FGDC”,”ISO-19115”, ISO19139”, ANZLIC	Zero or one (Optional)
sortSpec	SortField provides sorting information to the server for formatting data returned to the client	SortSpec , See Table 11	Zero or one (Optional) Default is sorted on ID in descending order
returnFormat	Specifies format (MIME or Internet media type) for returning result set metadata.	CodeList type with allowed values of XML HTML TXT	Zero or one (Optional) Default is “XML”
cursorPosition	First result set resource to be returned for this operation request.	Positive integer	Zero or one (Optional) Default is “1”
iteratorSize	Specifies maximum number of result set resources to be returned.	Non-negative integer	Zero or one (Optional) Default is “10”

Table 17 — UML attributes and role in "present" operation normal response

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol bindingand/or result sets are persistent
retrievedData	A subset of the results of this query request, organised and formatted as specified in the presentation, messageFormat, and sortField parameters.	Record type for the catalogue or collection.	One (Mandatory)
cursor	Last result set resource returned for this operation request.	Non-negative integer	One (Mandatory)
hits	Number of entries in the result set.	Non-negative integer	One (Mandatory)?

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding

7.2.4.4 describeRecordType operation

The describeRecordType operation is more completely specified in Table 18. Figure 8 provides a UML model of the describeRecordType operation that shows the complete Discovery class with the DescribeRecordTypeRequest and DescribeRecordTypeResponse classes and the class they use. The operation request includes the attibutes and association role name listed and defined in Table 19. The normal operation response includes the attibutes and association role name listed and defined in Table 20.

Note The describeRecordType operation corresponds to CG_Discovery.explainCollection operation in OGC Catalogue version 1.1.1.

Table 18 — Definition of describeRecordType operation

Definition	Allows clients to retrieve type definition(s) used by metadata of one or more registered resource types
Receives	Optional identifications of requested record type(s) and of desired format
Returns	Type definition document containing definition(s) of type(s) used by the metadata of one or more registered resource types. This type definition shall include the structure (schema), queryables, element sets, and formats of the metadata used for one or more registered resource types. The contents of the result of this operation depend on the types of metadata that can currently be used by registered resources.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Nonexistent type
Pre-conditions	None
Post-conditions	Type definition document returned to requesting client, containing definition(s) of type(s) used by the metadata of one or more registered resource types

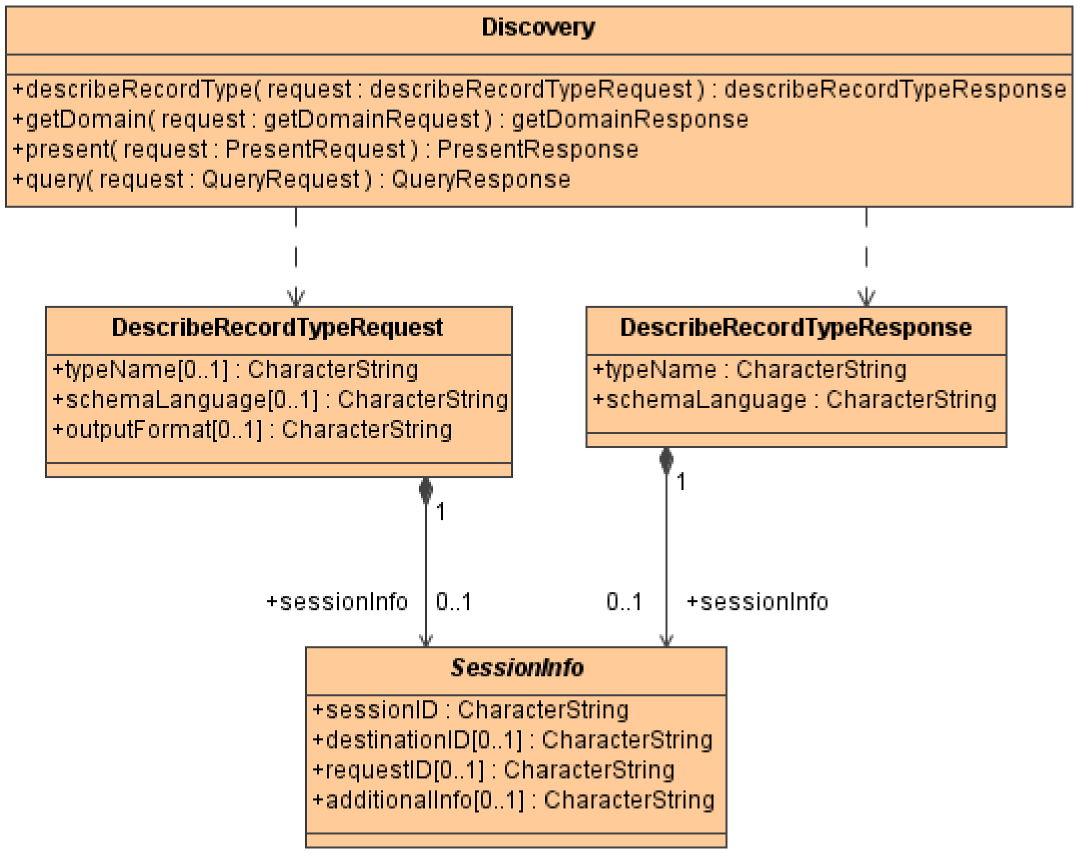
Discovery
+describeRecordType(request : describeRecordTypeRequest) : describeRecordTypeResponse
+getDomain(request : getDomainRequest) : getDomainResponse
+present(request : PresentRequest) : PresentResponse
+query(request : QueryRequest) : QueryResponse
DescribeRecordTypeRequest
+typeName[0..1] : CharacterString
+schemaLanguage[0..1] : CharacterString
+outputFormat[0..1] : CharacterString
DescribeRecordTypeResponse
+typeName : CharacterString
+schemaLanguage : CharacterString
1
+sessionInfo
0..1
1
0..1
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString

Figure 8 — describeRecordType operation UML static model

Table 19 — UML attributes and role in describeRecordType operation request

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
typeName	Name of metadata record type(s) for which type information is to be returned	Character String type Values specified by protocol binding	Zero or more (Optional) Return all types when omitted
schemaLanguage	The schema language of the response message	Character String type Values specified by protocol binding	Zero or one (Optional) Use XML Schema when omitted
outputFormat	Document format for output	Character String type Value is MIME type	Zero or one (Optional) Use application/xml when omitted.

Table 20 — UML attributes and role in describeRecordType operation normal response

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
typeName	Name of metadata record typefor which type information is returned	Character String Values are names of metadata record types	One or more (Mandatory) Include one for each record type to be returned
schemaLanguage	The schema language used to describe the type	Character String. Non-empty Values specified by protocol binding	One (Mandatory).

7.2.4.5 getDomain operation

The getDomain operation is more completely specified in Table 21. Figure 9 provides a UML model of the getDomain operation that shows the complete Discovery class with the GetDomainRequest and GetDomainResponse classes and the class they use. The operation request includes the attibutes listed and defined in Table 22. The normal operation response includes the attibutes and association role name listed and defined in Table 23.

Table 21 — Definition of getValueDomain operation

Definition	Allows clients to retrieve the domain (allowed values) of a metadata property or request parameter at the time the request is invoked. The returned information may be static domain information, but may also be dynamic in that the allowed values are determined at runtime. The operation does a *best attempt* at returning information about a metadata property or request parameter.
Receives	Names of one or more requested metadata properties or request parameters.
Returns	Descriptions of domains of one or more requested metadata properties or request parameters
Exceptions	Missing Parameter Value, Invalid Parameter Name
Pre-conditions	None
Post-conditions	Descriptions of domains returned to requesting client, containing the domain descriptions for all the identified metadata properties or request parameters.

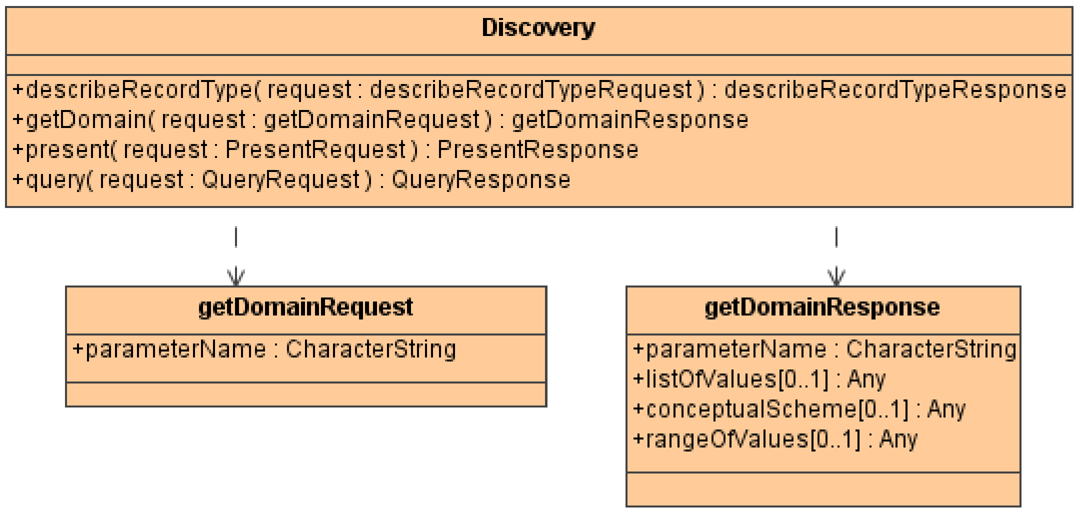
Discovery
+describeRecordType(request : describeRecordTypeRequest) : describeRecordTypeResponse
+getDomain(request : getDomainRequest) : getDomainResponse
+present(request : PresentRequest) : PresentResponse
+query(request : QueryRequest) : QueryResponse
getDomainRequest
+parameterName : CharacterString
getDomainResponse
+parameterName : CharacterString
+listOfValues[0..1] : Any
+conceptualScheme[0..1] : Any
+rangeOfValues[0..1] : Any

Figure 9 — getDomain operation UML static model

Table 22 — UML attibute in getDomain operation request

Name	Definition	Data type and value	Optionality
parameterName	The name of a metadata property or request parameter	Character string. Non-empty Allowed values specified by protocol binding	One (Mandatory)

Table 23 — UML attributes and role in getValueDomain operation normal response

Name	Definition	Data type and value	Optionality and use
parameterName	Name or identifier of metadata property or request parameter	Character String type, not empty	One (Mandatory)
listOfValues	Unordered list of domain values	Data type of list elements depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]
conceptualScheme	Reference to an authoritative list of domain values for the specified parameter	Data type of list of values in the authoritative list depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]
rangeOfValues	Range of domain values expressed by specifying a minimum and maximum value	Data type of the minimum and maximum values depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]
a For any single parameter, only one of listOfValues, conceptualScheme or rangeOfValues should be used to describe the value domain.			

7.2.5 Session class

7.2.5.1 Introduction

The Session class allows use of interactive sessions between a client and a server, by providing four stateful operations named “initiate”, “close”, “status”, and “cancel”. This class encapsulates the operations pertaining to session management. This class has an optional association from the Catalogue Service class, in which case this class is implemented by the Catalogue Service implementation.

NOTE **The four** Session operations are patterned after similar Z39.50 services.

7.2.5.2 “initialize” operation

The “initialize” operation is used to establish a session context with a Catalogue Service and is more completely specified in Table 24. Figure 10 provides a UML model of the “initialize” operation that shows the complete Session class with the InitiateRequest and InitiateResponse classes and the class they use. The operation request includes the association role name listed and defined in Table 25. The normal operation response includes the association role name listed and defined in Table 26.

Table 24 — Definition of “initiaize” operation

Definition	Allows clients to initiate an interactive session with a server, and generates a unique identifier used to track the session
Receives	Identifier of operation request
Returns	Data describing success or failure of this operation, plus session identifier when successful
Exceptions	Missing Parameter Value, Invalid Parameter Value
Pre-conditions	Existence of a Catalogue Service instance that supports sessions
Post-conditions	Result document returned to requesting client

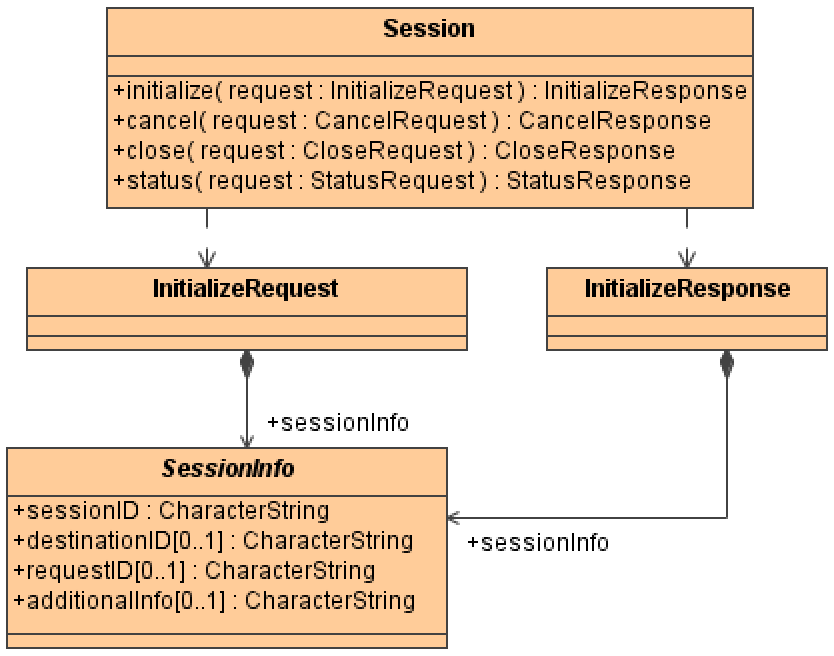
Session
+initialize(request : InitializeRequest) : InitializeResponse
+cancel(request : CancelRequest) : CancelResponse
+close(request : CloseRequest) : CloseResponse
+status(request : StatusRequest) : StatusResponse
InitializeRequest
InitializeResponse
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString
+sessionInfo

Figure 10 — "initialize" operation UML static model

Table 25 — UML role name in "initialize" operation request

Name	Definition	Data type	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

Table 26 — UML role name in "initiaize" operation normal response

Name	Definition	Data type	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.5.3 "close" operation

The "close" operation is more completely specified in Table 27. Figure 11 provides a UML model of the "close" operation that shows the complete Session class with the CloseRequest and CloseResponse classes and the class they use. The operation request includes the association role name listed and defined in Table 28. The normal operation response includes the association role name listed and defined in Table 29.

Table 27 — Definition of “close” operation

Definition	Allows clients to terminate an interactive session with a server
Receives	Identifiers of session and of operation request
Returns	Optional identities of session or request in acknowledgement.
Exceptions	Missing Parameter Value, Invalid Parameter Value
Pre-conditions	Client has previously initiated identified session
Post-conditions	Optional result document returned to requesting client , and results sets created during the session are deleted and other resources are released.

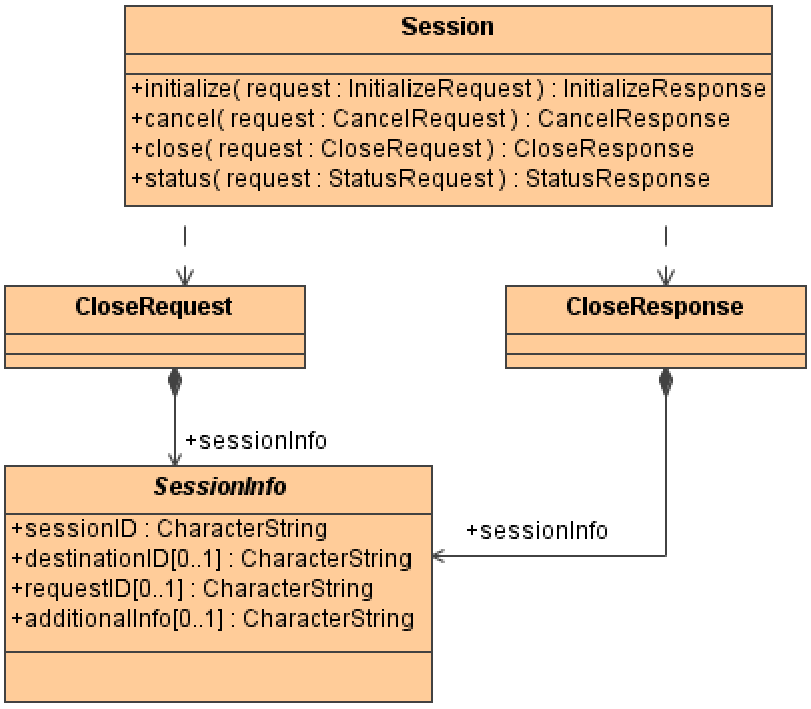
Session
+initialize(request : InitializeRequest) : InitializeResponse
+cancel(request : CancelRequest) : CancelResponse
+close(request : CloseRequest) : CloseResponse
+status(request : StatusRequest) : StatusResponse
CloseRequest
CloseResponse
+sessionInfo
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString

Figure 11 — “close” operation UML static model

Table 28 — UML role name in “close” operation request

Name	Definition	Data type	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

Table 29 — UML role name in "close" operation normal response

Name	Definition	Data type	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.5.4 "status" operation

The "status" operation is more completely specified in Table 30. Figure 12 provides a UML model of the "status" operation that shows the complete Session class with the StatusRequest and StatusResponse classes and the class they use. The operation request includes the attribute and association role name listed and defined in Table 31. The normal operation response includes the attribute and association role name listed and defined in Table 32.

Table 30 — Definition of "status" operation

Definition	Allows clients to retrieve current status of specified previously initiated operation in a session, either currently running or completed
Receives	Identifiers of session, previous operation request, and this operation request
Returns	Session or request IDs
Exceptions	Missing Parameter Value, Invalid Parameter Value
Pre-conditions	Client has previously initiated identified session and operation
Post-conditions	Result document returned to requesting client

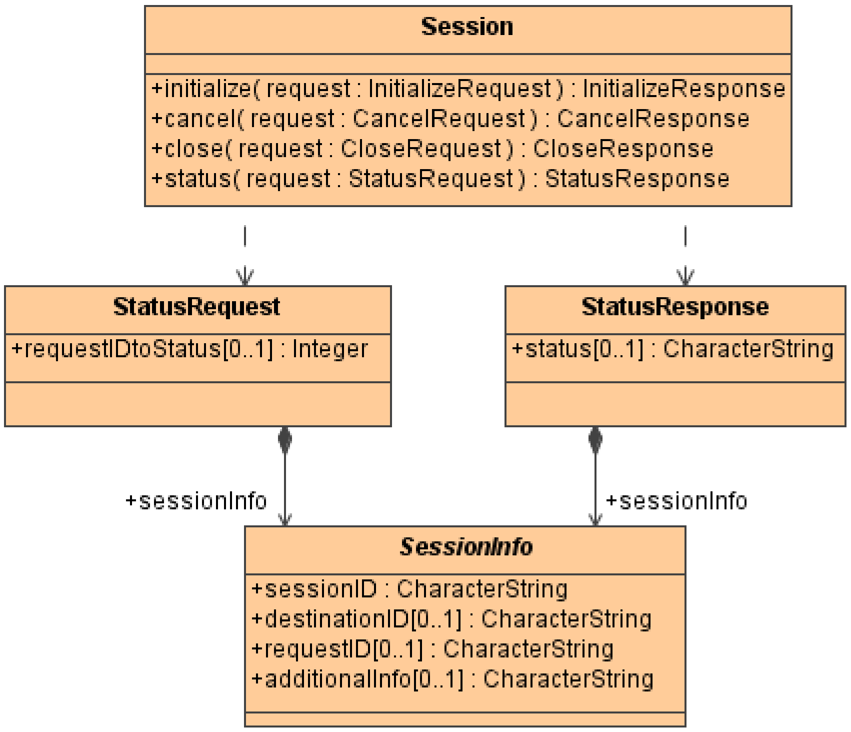
Session
+initialize(request : InitializeRequest) : InitializeResponse
+cancel(request : CancelRequest) : CancelResponse
+close(request : CloseRequest) : CloseResponse
+status(request : StatusRequest) : StatusResponse
StatusRequest
+requestIDtoStatus[0..1] : Integer
StatusResponse
+status[0..1] : CharacterString
+sessionInfo
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString

Figure 12 — "status" operation UML static model

Table 31 — UML attribute and role in "status" operation request

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
requestIDtoStatus	Unique requestID for operation execution to be obtain status of	Integer Value of request to status	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

Table 32 — UML attribute and role in “status” operation normal response

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
status	Message containing the status disposition	CharacterString Values TBD	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.5.5 “cancel” operation

The “cancel” operation is more completely specified in Table 33. Figure 13 provides a UML model of the “cancel” operation that shows the complete Session class with the CancelRequest and CancelResponse classes and the class they use. The operation request includes the attibutes and association role name listed and defined in Table 34. The normal operation response includes the attibutes and association role name listed and defined in Table 35.

Table 33 — Definition of “cancel” operation

Definition	Allows clients to cancel a previously initiated operation in a session, either currently running or completed. Any partial or completed result set from that operation is discarded.
Receives	Identifiers of session, previous operation request, and this operation request
Returns	Data describing success or failure of this operation
Exceptions	Missing Parameter Value, Invalid Parameter Value
Pre-conditions	Client has previously initiated identified session and operation
Post-conditions	Search request is cancelled, current result set erased (resources freed) unless otherwise specified

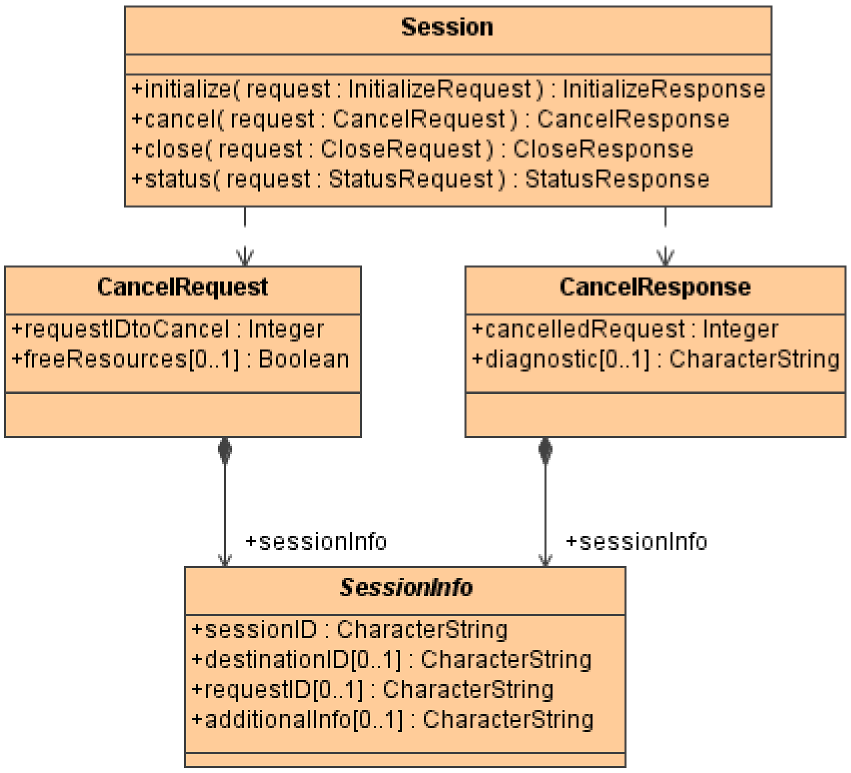
Session
+initialize(request : InitializeRequest) : InitializeResponse
+cancel(request : CancelRequest) : CancelResponse
+close(request : CloseRequest) : CloseResponse
+status(request : StatusRequest) : StatusResponse
CancelRequest
+requestIDtoCancel : Integer
+freeResources[0..1] : Boolean
CancelResponse
+cancelledRequest : Integer
+diagnostic[0..1] : CharacterString
+sessionInfo
+sessionInfo
SessionInfo
+sessionID : CharacterString
+destinationID[0..1] : CharacterString
+requestID[0..1] : CharacterString
+additionalInfo[0..1] : CharacterString

Figure 13 — “cancel” operation UML static model

Table 34 — UML attributes and role in “cancel” operation request

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
requestIDtoCancel	Unique requestID for operation execution to be cancelled	Integer ID of request to cancel	One (Mandatory)
freeResources	If set to FALSE, any partial result set is not deleted until the client terminates the session.	Boolean	Zero or one (Optional) Free resources if omitted

Table 35 — UML attributes and role in "cancel" operation normal response

Name	Definition	Data type and value	Optionality and use
sessionInfo	The core set of parameters required in each message exchanged between a client and server operating in a session context, where these parameters support message routing and session management	SessionInfo, See Table 12	Zero or one (Conditional) Include when sessions are supported/required by protocol binding and/or result sets are persistent
cancelledRequest	Unique requestID for operation that was the target of the cancel request	Integer ID of request cancelled	One (Mandatory)
diagnostic	Text message describing the result of the cancel request	CharacterString Values TBD	Zero or one (Optional)

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.6 Manager class

7.2.6.1 Introduction

The Manager class allows a client to insert, update and/or delete catalogue content. This class has an optional association from the CatalogueService class; it is not required that a catalogue service implement publishing functionality. Two operations are provided: "transaction" and harvestResource.

The "transaction" operation allows a client to formulate a transaction, and send it to the catalogue to be processed. The transaction may contain metadata records and elements of the information model that the catalogue understands. To use the transaction operation, the client must know something about the information model that the catalogue implements.

The "harvestResource" operation, on the other hand, directs the catalogue to retrieve an accessible metadata record and processes it for inclusion in the catalog, perhaps periodically re-fetching the metadata records to refresh the information in the catalog. The client does not need to be aware of the information model of the catalogue when using the "harvestResource" operation, since the catalogue itself is doing the work required to process the information. The client is simply pointing to where the metadata resource to be harvested is.

7.2.6.2 "transaction" operation

The "transaction" operation is more completely specified in Table 36.

Figure 14 provides a UML model of the "transaction" operation that shows the complete Manager class with the TransactionRequest and TransactionResponse classes and the classes they use. The operation request includes the attributes listed and defined in Table 37. The normal operation response includes the attributes listed and defined in Table 38.

Table 36 — Definition of “transaction” operation

Definition	Allows clients to request a specified set of “insert”, “update”, and “delete” actions on the content managed by a Catalogue Service instance.
Receives	Specification of set of “insert”, “update”, and “delete” actions, plus an optional identifier. At least one action shall be included.
Returns	A summary of the transaction results that identifies newly created entries when applicable. Most contents of the result depend on the types of data defined by the specific protocol binding and Application Profile.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Transaction Failed
Pre-conditions	User is authorized to modify catalogue contents
Post-conditions	Catalogue entries are inserted, updated, and/or deleted as requested, and the integrity and consistency of catalogue contents are preserved..

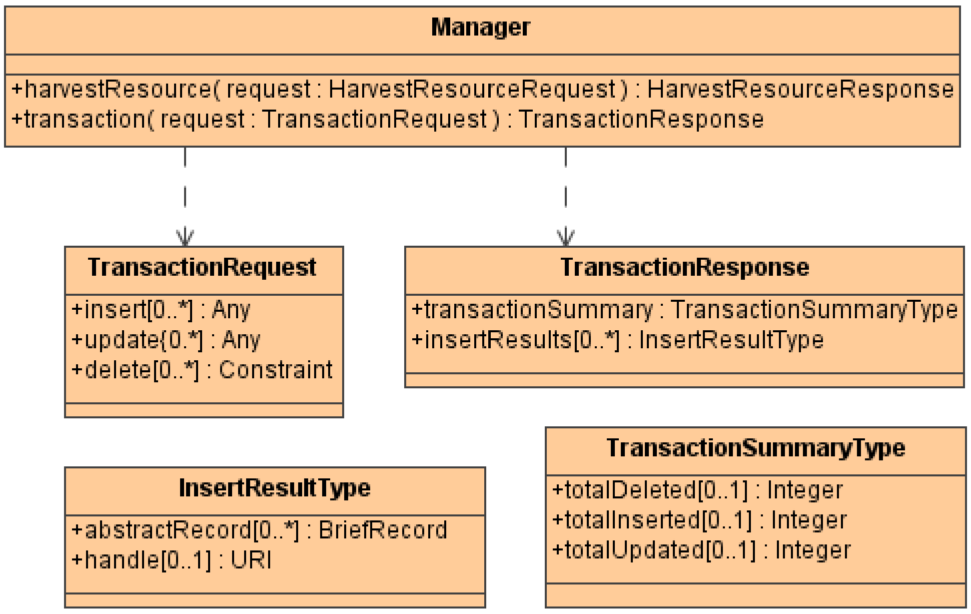
Manager
+harvestResource(request : HarvestResourceRequest) : HarvestResourceResponse
+transaction(request : TransactionRequest) : TransactionResponse
TransactionRequest
+insert[0..*] : Any
+update{0.*] : Any
+delete[0..*] : Constraint
TransactionResponse
+transactionSummary : TransactionSummaryType
+insertResults[0..*] : InsertResultType
InsertResultType
+abstractRecord[0..*] : BriefRecord
+handle[0..1] : URI
TransactionSummaryType
+totalDeleted[0..1] : Integer
+totalInserted[0..1] : Integer
+totalUpdated[0..1] : Integer

Figure 14 — “transaction” operation UML static model

Table 37 — UML attributes in "transaction" operation request

Name	Definition	Data type and value	Optionality and use
insert	The insert action is used to create new metadata records in a catalog. Each insert action may contain one or more new metadata record instances that are to be inserted into the catalog.	Any, a container for one or more metadata record instances The schema for metadata records is defined in the protocol binding and may be extended or redefined in an Applcation Profile	Zero or more (Optional) Include when client wishes to insert one or more new catalogue records
update	The update action is used to modify existing records in the catalog. The update action contains a single new metadata record instance and a predicate that defines the set of catalogue records that will be modified. The predicate may identify zero or more records that are to be modified by the update action. The encoding of the predicate is specified in the protocol binding and may be further qualified or extended in an Application Profile.	Any, contains one instance of a metadata record that will be used to update existing records in catalog The schema of the record is defined in the protocol binding and may be extended or redefined in an Applcation Profile	Zero or more (Optional) Include when client wishes to modify one or more existing catalogue records
delete	The delete action is used to remove one or more records from a catalog. The records to be removed are identified by specifying a predicate with the operation. The predicate may identify zero or more records that are to be removed from the catalogue by the delete action. The encoding of the predicate is specified in the protocol binding and may be further qualified or extended in an Application Profile.	The delete action requires a constraint predicate that identifies the records in the catalogue to be removed	Zero or one (Optional) Include when client wishes to delete one or more existing records from a catalog

Table 38 — UML attributes in "transaction" operation normal response

Name	Definition	Data type and value	Optionality
transaction Summary	Summary of transaction results that includes the numbers of records inserted, updated, and deleted by the actions specified in the transaction	TransactionSummaryType Total number of records inserted, updated, and deleted (Integer)	One (Mandatory)
insert Results	Brief representation of a record created by the transaction, which **must** include the record identifier May contain a handle that relates newly created record with the insert action that created it	InsertResultType Structure composed of brief record type (application profile or protocol binding dependent) and an optional handle	Zero or more (Optional) Include one for each record created

7.2.6.3 harvestResource operation

The harvestResource operation facilitates the retrieval of remote resources from a designated location and provides for optional transactions on the local catalogue. The harvestResource operation is described in Table 39.

Figure 15 provides a UML model of the “harvestResource” operation that shows the complete Manager class with the HarvestResourceRequest and HarvestResourceResponse classes. The operation request includes the attributes listed and defined in Table 40. The normal operation response includes the attributes listed and defined in Table 41.

Table 39 — harvestResource operation

Definition	Allows a user to request that a catalogue service attempt to retrieve a resource from a specified location, and to optionally create one or more entries for that resource. A harvest attempt may occur periodically if an interval is specified.
Receives	A request message containing the source of the resource to be harvested
Returns	An acknowledgement that a harvestRequest has been received and validated (if a responseHandler is specified) or a summary of the harvest results that identifies newly harvested records (if a responseHandler is not specified). Most contents of the result depend on the types of data defined by the specific protocol binding and Application Profile.
Exceptions	InvalidRequest, ResourceNotFound
Pre-conditions	The user is permitted to modify catalogue contents, unless the scope of the harvest does not include an insert or update transaction
Post-conditions	One or more records are harvested from a remote system and optionally new catalogue entries are created or existing entries are updated, and the integrity and consistency of the catalogue contents are preserved

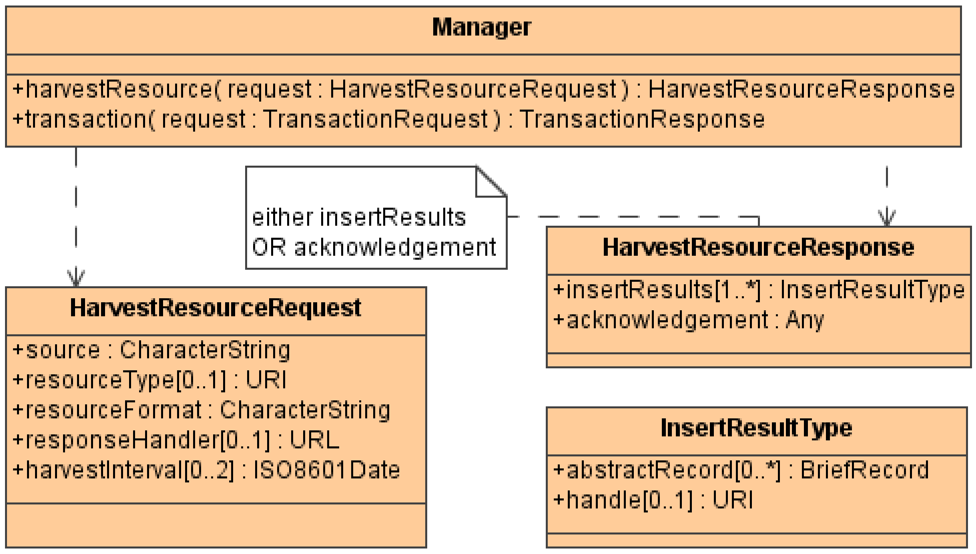
Manager
+harvestResource(request : HarvestResourceRequest) : HarvestResourceResponse
+transaction(request : TransactionRequest) : TransactionResponse
either insertResults
OR acknowledgement
HarvestResourceResponse
+insertResults[1..*] : InsertResultType
+acknowledgement : Any
HarvestResourceRequest
+source : CharacterString
+resourceType[0..1] : URI
+resourceFormat : CharacterString
+responseHandler[0..1] : URL
+harvestInterval[0..2] : ISO8601Date
InsertResultType
+abstractRecord[0..*] : BriefRecord
+handle[0..1] : URI

Figure 15 — harvestResource operation UML static model

Table 40 — UML attributes in harvestResource operation request

Name	Definition	Data type and value	Optionality and use
source	Location from which resource to be retrieved	URL	One (Mandatory)
resourceType	Identifier of type of resource to be harvested, if known	URI	Zero or one (Optional) If the parameter is not specified then the catalogue should determine the resourceType from the content of the message
resourceFormat	Identifier of media type indicating the format of resource to be harvested	CharacterString Value must be a media type supported by catalogue	One (Mandatory)
responseHandler	Network location to which the response will be forwarded when operation has been completed, for asynchronous requests	URL	Zero or one (Optional) If not included, process request synchronously
harvestInterval	Time interval between harvest attempts	Period Using ISO 8601 Period syntax (e.g., P6M indicates an interval of six months)	Zero or one (Optional) If the parameter is not specified then the catalogue should harvest the resource once in response to the request.

Table 41 — UML attributes in harvestResource operation normal response

Name	Definition	Data type and value	Optionality and use
acknowledgement	Summary of transaction results, with contents depending on the protocol binding and Application Profile (e.g. total records affected by each action)	Any	One (Mandatory)
insertResults	Brief representation of a record created by the transaction, which **must** include the record identifier May contain a handle that relates newly created record with the insert statement that created it	InsertResultType A structure composed of the brief record type (application profile or protocol binding dependant) and an optional handle	One or more (Mandatory) Include one for each new record created in catalog

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.7 Brokered Access class

7.2.7.1 Introduction

The Brokered Access class allows clients to place an order for an identified registered resource, for use when that resource is a data product that is not directly accessible to clients. This class has an optional association from the Catalogue Service class, in which case this interface is implemented by the Catalogue Service implementation.

Not all resources can be accessed directly. Brokered access provides for accessing resources that are controlled. Controlled resources might include those for which one or more of the following applies:

a) A fee is charged

b) Have security limitations

c) Require additional processing

d) Are not available electronically

NOTE This class is included partially for backwards compatibility. This class may be deprecated in the future to instead use a general framework for ordering more than catalogued data sets.

7.2.7.2 “order” operation

The single “order” operation is more completely specified in Table 42.

Figure 16 provides a UML model of the “order” operation that shows the complete BrokeredAccess class with the OrderRequest and OrderResponse classes. The operation request includes the attibutes listed and defined in Table 43. The normal operation response includes the attibutes listed and defined in Table 44.

Table 42 — Definition of “order” operation

Definition	Allows clients to order a specified product or resource, and negotiate order price and other factors with ordering service
Receives	Identifiers of desired product or resource of this order , user billing information and type of order
Returns	Order modifications, order estimates, order status
Exceptions	Missing Parameter Value, Invalid Parameter Value, ResourceNotFound, InvalidUserID
Pre-conditions	User registered with order service, resource identifiers known
Post-conditions	Response returned to requesting client

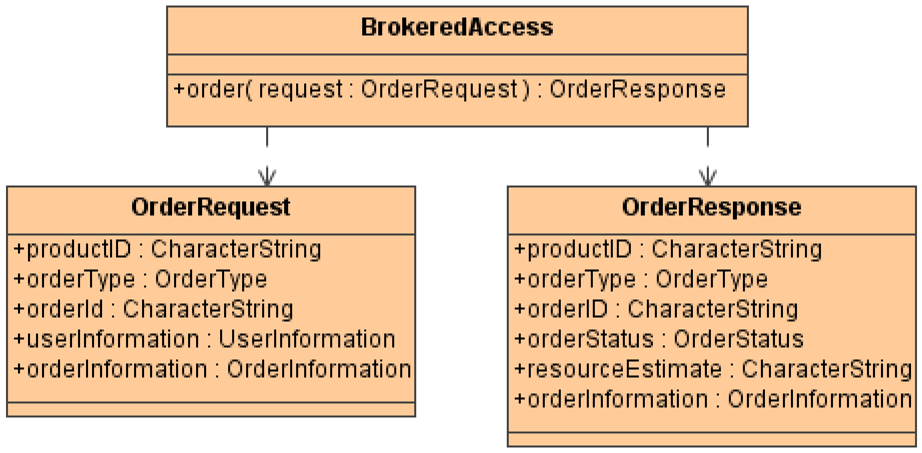
BrokeredAccess
+order(request : OrderRequest) : OrderResponse
OrderRequest
+productID : CharacterString
+orderType : OrderType
+orderId : CharacterString
+userInformation : UserInformation
+orderInformation : OrderInformation
OrderResponse
+productID : CharacterString
+orderType : OrderType
+orderID : CharacterString
+orderStatus : OrderStatus
+resourceEstimate : CharacterString
+orderInformation : OrderInformation

Figure 16 — "order" operation UML static model

Table 43 — UML attributes in "order" operation request

Name	Definition	Data type and value	Optionality
productID	Unique identifier for specific resource being ordered, taken from catalogue metadata	Character String type, not empty	One (Mandatory)
orderType	Type of order request	OrderType (CodeList), with allowed values of: orderEstimate, orderQuoteAndSubmit, orderMonitor, and orderCancel	One (Mandatory)
orderID	Unique identifier for an order	Character String type, not empty	One (Mandatory)
orderInformation	Specification of current order as provided by the client	OrderSpecification data type, see Annex C	One (Mandatory)
userInformation	Needed requester identification, for billing and delivery	UserInformation data type, see Annex C	One (Mandatory)

Table 44 — UML attributes in "order" operation normal response

Name	Definition	Data type and value	Optionality
productID	Unique identifier for specific resource being ordered	Character String type, not empty Values from catalogue metadata	One (Mandatory)
orderType	Type of order request	OrderType (CodeList), with allowed values of: orderEstimate, orderQuoteAndSubmit, orderMonitor, and orderCancel	One (Mandatory)
orderID	Unique identifier for this order	Character String type, not empty Value assigned by client	One (Mandatory)
orderStatus	Current status of the order	CodeList data type [a]	One (Mandatory)
resourceEstimate	Estimate of the resources needed to process and/or deliver the requested resource. Examples of these resources are time until delivery and cost.	Character String type, not empty Values TBD	One (Mandatory)
orderInformation	Specification of current order, as provided by client or modified by server during resource estimation	OrderSpecification data type, see Annex C	One (Mandatory)

a Possible orderStatus values are orderBeingEstimated, orderEstimated, orderBeingQuoted, orderBeingProcessed, orderCompleted, orderNotValid, and orderCancelled.

7.3 Protocol, interface and operation specializations

The catalogue service specification includes recognized Protocol Bindings, formerly known as Implementation Profiles. Clause 11 defines the rules by which an Application Profile as a dependent specification can be written. Protocol Bindings must implement the features present in the General Model, described in Clause 7 following the optionality expressed there. Protocol Bindings interpret the general model in the referenced implementation environment. These artifacts are discussed in detail in Subclause 7.2 and the Protocol Binding clauses of this document.

7.4 Dynamic model

7.4.1 Introduction

The Catalogue Interface defines a stateful session (a stateless interface will be added in future versions of this Implementation Specification). This subclause defines the states of the session and the allowed transitions between the states. All other state transitions are disallowed and are consider errors if exhibited by a server.

A physical server may support more than one session. Each of the sessions are independent when viewed from the interface defined by this specification.

In the state models below, a transition is typically triggered by a request. Following the messaging model introduced earlier, a Request is paired with a Response. Generally, a transaction in this model is bounded by a request-response pair. Note that a transaction can be statused or cancelled while it is active, i.e., before a response is issued. Once the server has sent a Response, the server treats the receipt of a StatusRequest (or CancelRequest) as an error, to which it responds gracefully. Gracefully means that the server should respond with a StatusResponse (or a CancelResponse) with a diagnostic indicating that the RequestIDtoStatus (or the RequestIDtoCancel) is not recognised. The server shall not change state in response to a StatusRequest (or CancelRequest) when the transaction is complete, i.e., a Response has been sent.

7.4.2 UML state diagram notation

The state diagrams in the following subclauses use the UMLnotation. Figure 17 provides a summary of the UML notation used in the following subclauses. Transitions are the paths between states. A transition will occur if the event occurs and the guard condition is true. If a transition occurs, the Action is completed prior to entering the next state.

Composite states contain multiple sub-states. Both the Sequential Composite State and the Concurrent Composite State types are used in model for the Catalogue Interface. In a Sequential Composite State only one sub-state is active at any given time. UML defines that when a transition enters Concurrent Composite State all of the sub-states are active, although some of the sub-states may remain in the Initial State. When exiting a composite state, all sub-states are exited as well.

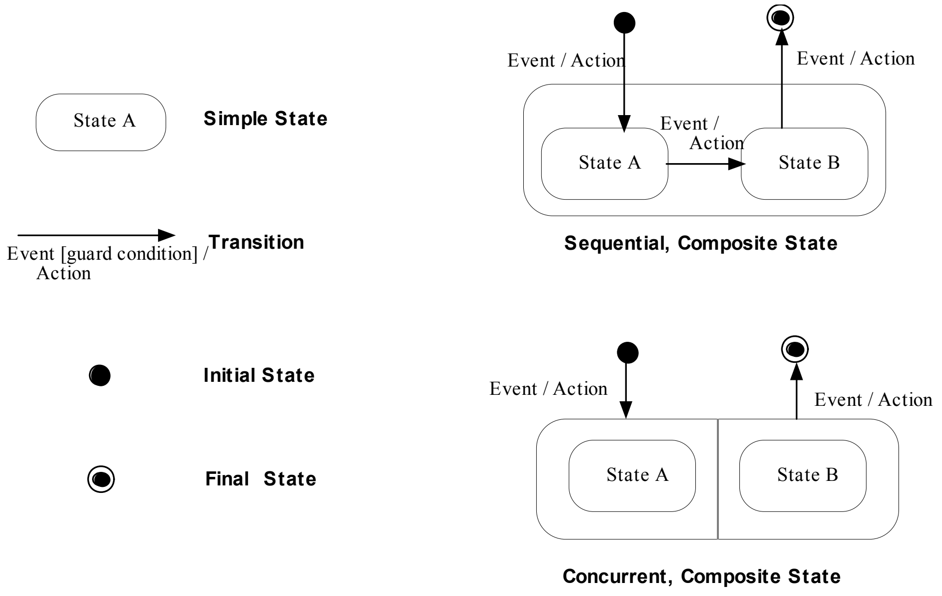

Figure 17 — UML state diagram notation

7.4.3 Catalogue server state machine

The top-level state diagram for the Catalogue Interface is shown in Figure 18. After a successful initialization, the session will be in the Main state. The Main state is a concurrent, composite state, consisting of four substates: Discovery, Access, Management, and Explain. While in the Main state, Requests (other than TerminateRequest) may cause transitions internal to the substates. To determine what transition occurs for the various Requests, the internals of the substates must be examined. (If a server does not support interfaces associated with a substate, the substate is not present for sessions with that server. For example, if the server does not support access, then the Access Substate is not present.)

When a TerminateRequest is received, the session will transition from the Main state to the end state, ending all processing associated with the substates of Main. The Catalogue Session state diagram allows the server to end a session after a designated, configurable duration, i.e., timeout. When a session times-out, the server closes the session without notification to the client. The server must be prepared to respond to client requests for a session that has timed out by returning the paired response containing a diagnostic indicating that the session does not exist.

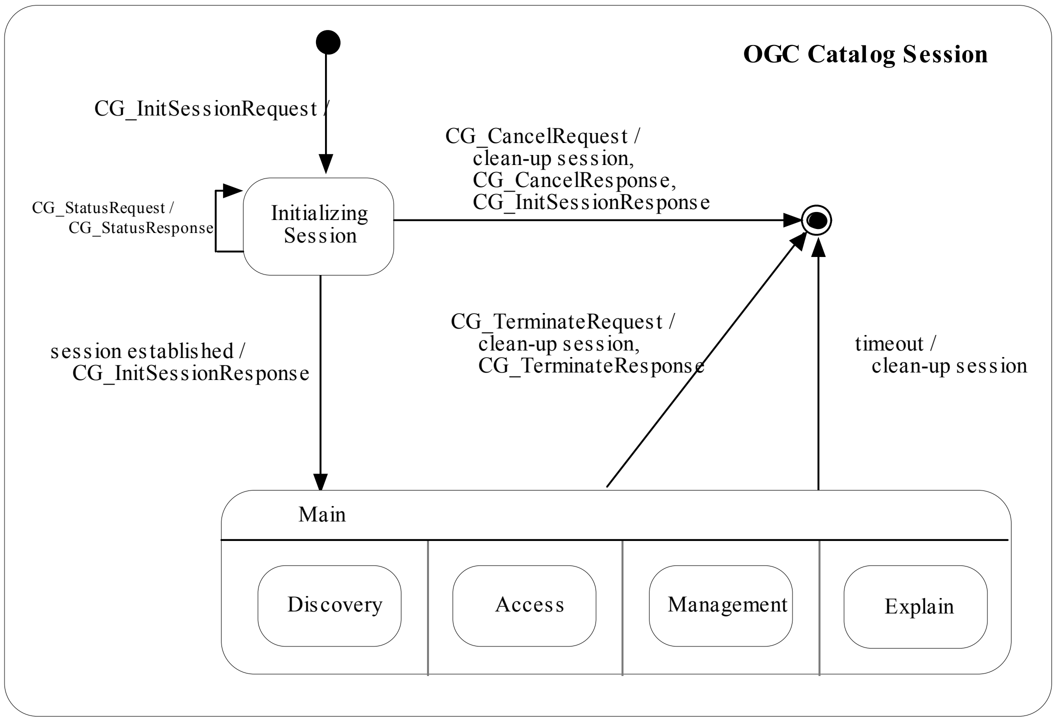
OGC Catalog Session
CG_InitSessionRequest /
CG_StatusRequest / CG_StatusResponse
Initializing Session
CG_CancelRequest / clean-up session, CG_CancelResponse, CG_InitSessionResponse
session established / CG_InitSessionResponse
CG_TerminateRequest / clean-up session, CG_TerminateResponse
timeout / clean-up session
Main
Discovery
Access
Management
Explain

Figure 18 — Catalogue session state diagram

7.4.4 Discovery state

Two views of the Discovery State diagram are provided: Figure 19 shows an abbreviated state diagram, Figure 20 shows the complete Discovery state diagram. The abbreviated version is only provided to assist the reader in understanding the complete diagram.

A session can be in the Discovery substate, once a successful initialization has occurred at which time the Discovery substate will be in the initial state. Upon receiving any QueryRequest, the Discovery state will transition to the Processing Query state. Transitions leaving the Processing Query state are dependent upon the resultType that was requested in the QueryRequest that caused entry into the Processing Query state. The four potential values for resultType are Validate, Result Set ID, Hits, Results. If a PresentRequest is sent by the client prior to the query completing, the session will transition to the Processing Query and Formatting Results state. The formatting of records and a PresentResponse must occur causing a transition to the Processing Query state, prior to completing the query and sending a QueryResponse, if necessary.

When the query completes and the resultType was not Results, the state will transition to the Idle state, sending a QueryResponse unless the resultType was Validate, in which case a response has already been sent. When the resultType was Results, the state will pass to the Formatting Records for Query state, until the results are ready and a QueryResponse containing the records can be sent. While in the Idle state, a PresentRequest may be sent by the client, in which case, if a result set is present, the state will transition to the Formatting Records state, until the results are ready and a QueryResponse containing the records can be sent. As will be seen in the next diagram, there need not be a result set when the Discovery substate is Idle. If no result set is present while in the Idle state and a PresentRequest is received, the state will not transition and a PresentResponse will be returned with a diagnostic.

If a QueryRequest is received while in the Idle state, the result set for the session, if present, will be reset, and the state will transition to the Processing Query state, creating a new result set. A catalogue session can only have a single result set. (Future enhancements of the Catalogue Interface may allow multiple result sets to exist in a session.) The result set is also deleted when a TerminateRequest is received and the Catalogue Interface state, which includes the Discovery substate, transitions from Main to the end state.

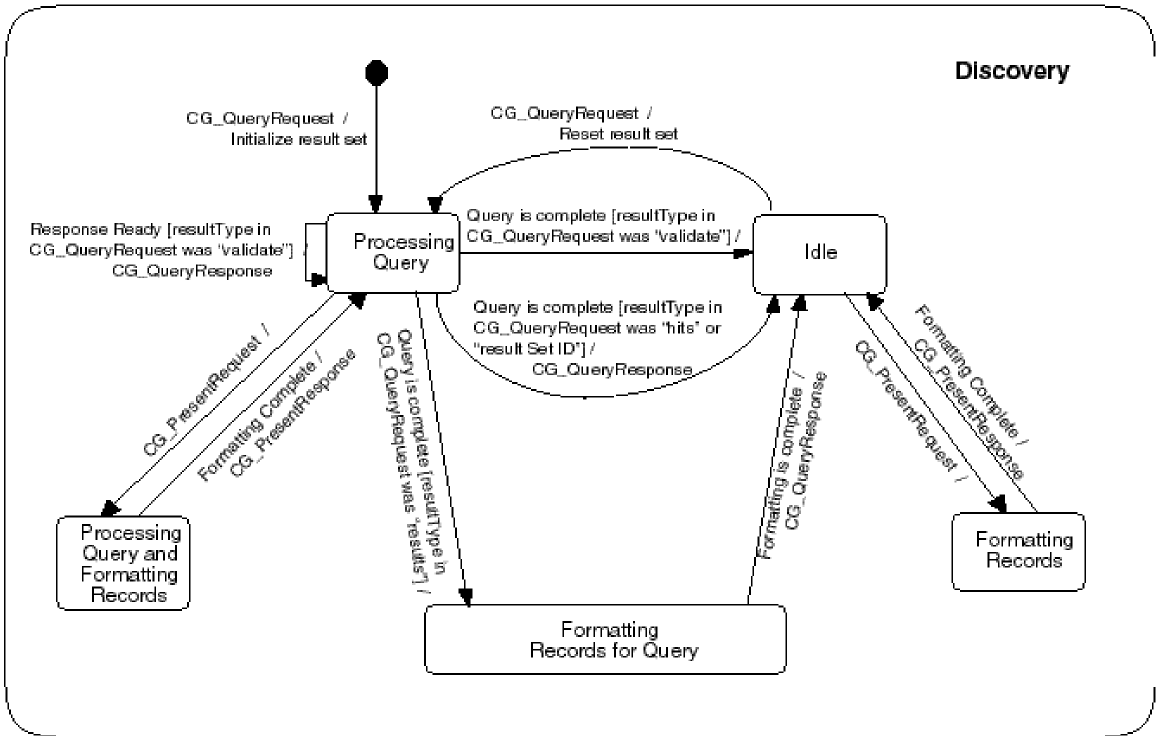
Discovery
CG_QueryRequest / Initialize result set
CG_QueryRequest / Reset result set
Response Ready [resultType in CG_QueryRequest was 'validate"] / CG_QueryResponse
Processing Query
Query is complete [resultType in CG_QueryRequest was 'validate"] /
Idle
Query is complete [resultType in CG_QueryRequest was "hits" or "result Set ID"] / CG_QueryResponse
CG_PresentRequest /
Formatting Complete / CG_PresentResponse
Query is complete [resultType in CG_QueryRequest was "results"] /
Formatting is complete / CG_QueryResponse
CG_PresentRequest /
Formatting Complete / CG_PresentResponse
Processing Query and Formatting Records
Formatting Records for Query
Formatting Records

Figure 19 — Discovery state diagram (without Status and Cancel)

The complete Discovery state diagram adds StatusRequest and CancelRequest. The substates of Discovery remain the same, but additional transitions are present. If a CancelRequest is received while in the Processing Query state, the session will transition to the Idle state. Depending upon the value of the freeResources parameter in the CancelRequest, a result set may or may not exist once in the Idle State. Note that because the client sets the request ID in a request, the client knows the ID that is used in a status or cancel request.

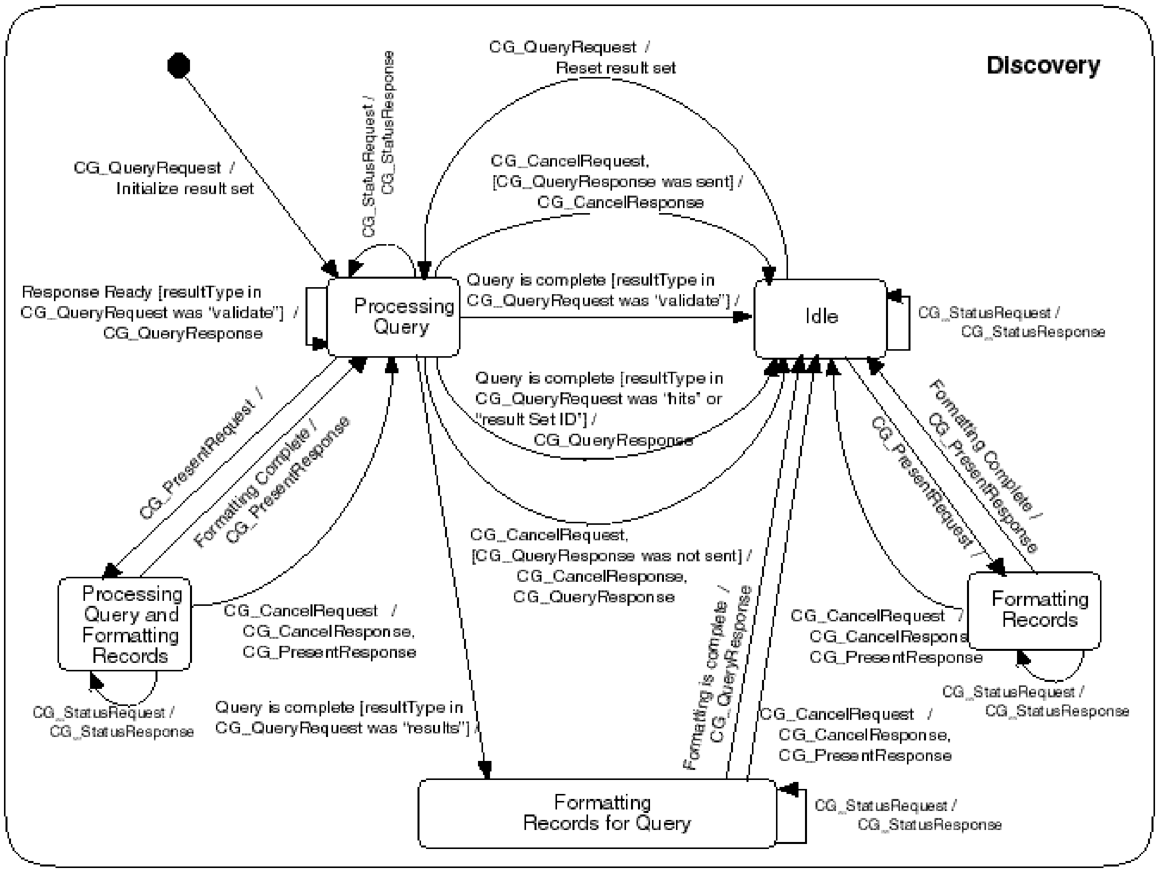
Discovery
CG_QueryRequest / Initialize result set
CG_StatusRequest / CG_StatusResponse
CG_QueryRequest / Reset result set
CG_CancelRequest, [CG_QueryResponse was sent] / CG_CancelResponse
Response Ready [resultType in CG_QueryRequest was 'validate"] / CG_QueryResponse
Processing Query
Query is complete [resultType in CG_QueryRequest was 'validate"] /
Idle
CG_StatusRequest / CG_StatusResponse
Query is complete [resultType in CG_QueryRequest was 'hits' or "result Set ID"] / CG_QueryResponse
CG_PresentRequest /
Formatting Complete / CG_PresentResponse
Formatting Complete / CG_PresentResponse
CG_PresentRequest /
CG_CancelRequest, [CG_QueryResponse was not sent] / CG_CancelResponse, CG_QueryResponse
Processing Query and Formatting Records
CG_CancelRequest / CG_CancelResponse, CG_PresentResponse
Formatting Records
CG_CancelRequest / CG_CancelResponse, CG_PresentResponse
CG_StatusRequest / CG_StatusResponse
CG_StatusRequest / CG_StatusResponse
Query is complete [resultType in CG_QueryRequest was "results"] /
Formatting is complete / CG_QueryResponse
CG_CancelRequest / CG_CancelResponse, CG_PresentResponse
Formatting Records for Query
CG_StatusRequest / CG_StatusResponse

Figure 20 — Discovery state diagram (complete)

7.4.5 Access state diagram

The Access State Diagram is shown in Figure 21. A session can be in the Access substate, once a successful initialization has occurred at which time the Access substate will be in the initial state. Upon receiving a BrokeredAccessRequest, the Access state will transition to the Processing Request State. During the Processing Request State, the state of an Order may be modified based on the contents of the BrokeredAccessRequest. The state of the Order is a separate state machine; see Figure 22 and Figure 23 where order estimation and order submission are diagrammed. Transitions in the Order state may occur independent of OGC Catalogue Interface requests, e.g., order fulfilled is a transition that occurs without a BrokeredAccessRequest. The server may delete orders. The server must be prepared to respond to client request for an order that has been deleted by returning the paired response containing a diagnostic indicating that the order does not exist.

Once the processing of a BrokeredAccessRequest has completed a response is sent and the state transitions to Idle. Transition out of the Idle state occurs upon the client sending a BrokeredAccessRequest in which case the state transitions to Processing Request. When a TerminateRequest is received, the Catalogue Interface state, which includes the Access substate, transitions from Main to the end state also closing the Access state.

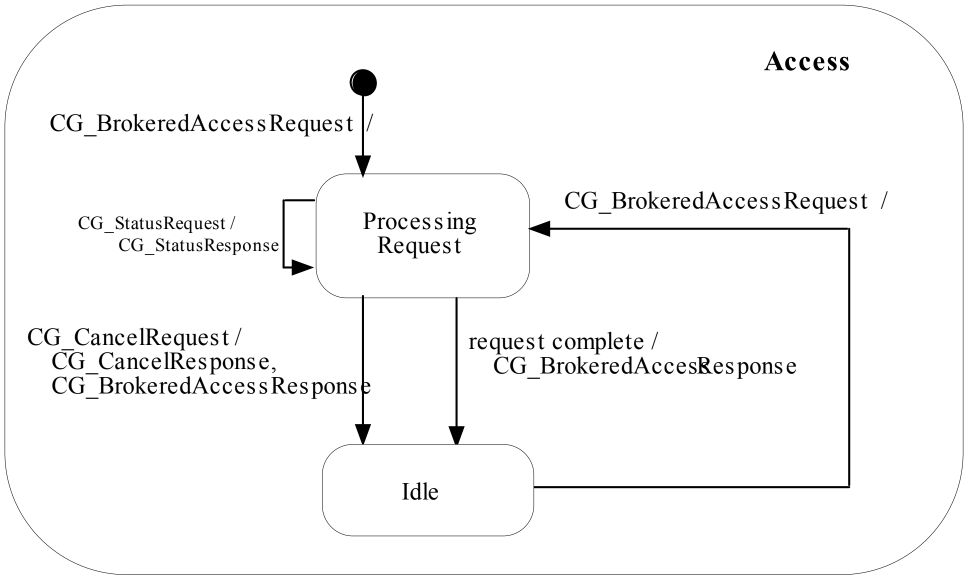

Figure 21 — Access state diagram

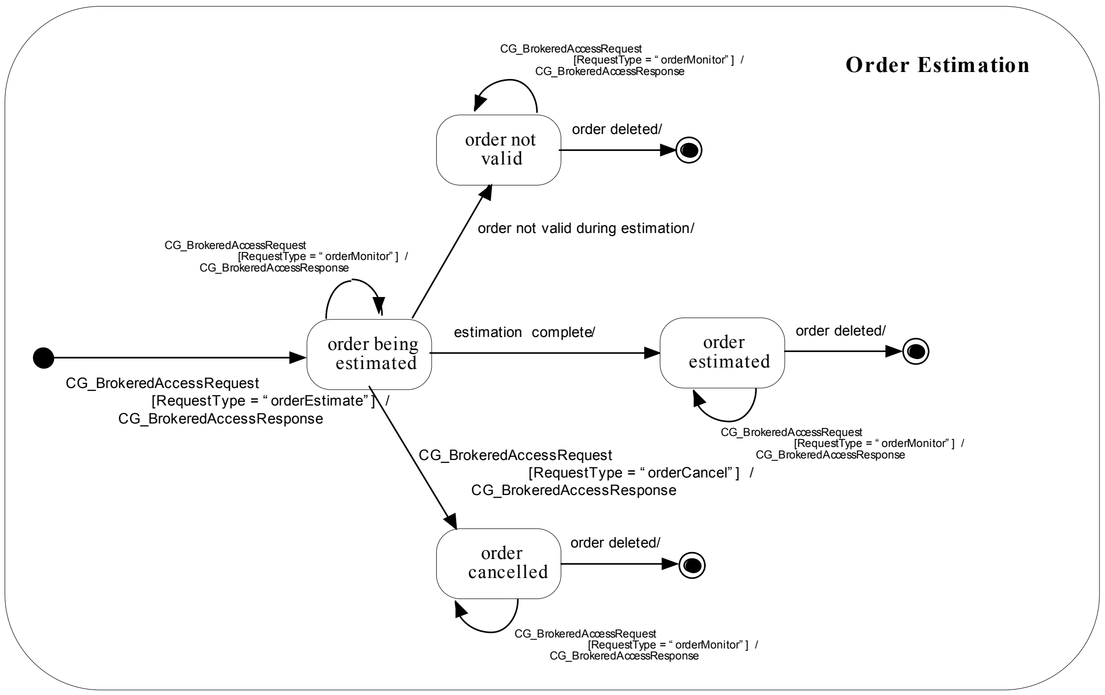
Order Estimation
CG_BrokeredAccessRequest
[RequestType = " orderMonitor"] /
CG_BrokeredAccessResponse
order not
valid
order deleted/
order not valid during estimation/
CG_BrokeredAccessRequest
[RequestType = " orderMonitor"] /
CG_BrokeredAccessResponse
order being
estimated
estimation complete/
order
estimated
order deleted/
CG_BrokeredAccessRequest
[RequestType = " orderEstimate"] /
CG_BrokeredAccessResponse
CG_BrokeredAccessRequest
[RequestType = " orderMonitor"] /
CG_BrokeredAccessResponse
CG_BrokeredAccessRequest
[RequestType = " orderCancel"] /
CG_BrokeredAccessResponse
order
cancelled
order deleted/
CG_BrokeredAccessRequest
[RequestType = " orderMonitor"] /
CG_BrokeredAccessResponse

Figure 22 — Order estimation state diagram

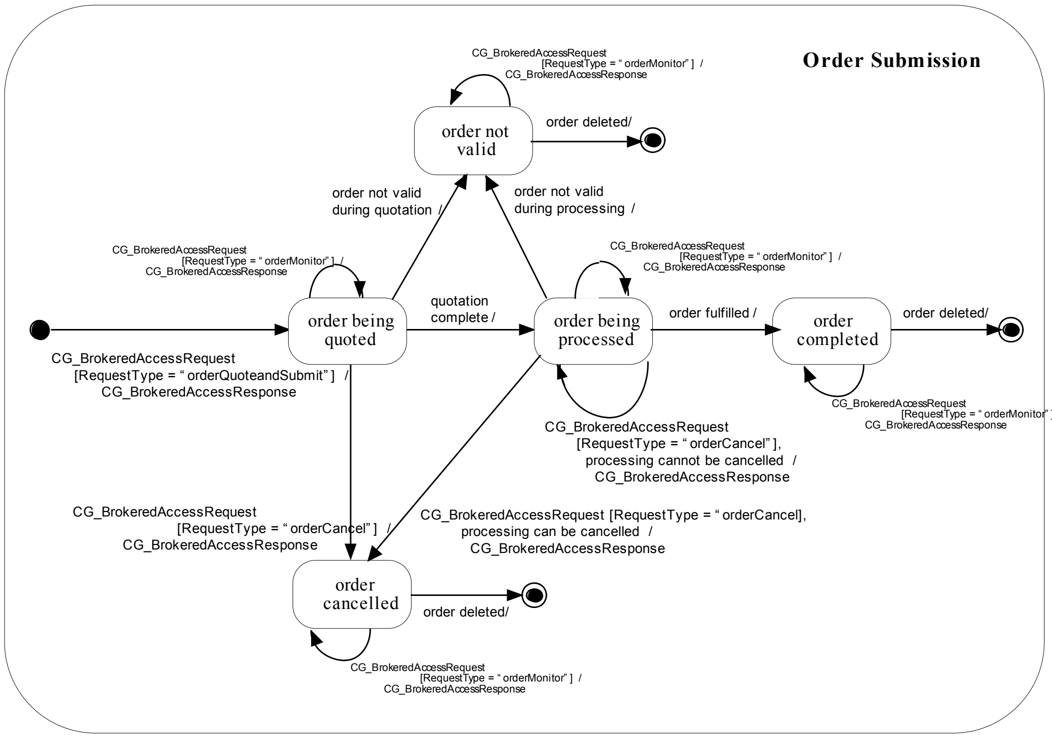
Order Submission
order not valid
order deleted/
order not valid during quotation /
order not valid during processing /
order being quoted
quotation complete /
order being processed
order fulfilled /
order completed
order deleted/
order cancelled
order deleted/
CG_BrokeredAccessRequest [RequestType = " orderQuoteandSubmit"] / CG_BrokeredAccessResponse
CG_BrokeredAccessRequest [RequestType = " orderMonitor"] / CG_BrokeredAccessResponse
CG_BrokeredAccessRequest [RequestType = " orderCancel"] / CG_BrokeredAccessResponse
CG_BrokeredAccessRequest [RequestType = " orderCancel"], processing cannot be cancelled / CG_BrokeredAccessResponse
CG_BrokeredAccessRequest [RequestType = " orderCancel], processing can be cancelled / CG_BrokeredAccessResponse

Figure 23 — Order submit state diagram

7.4.6 Management state

The Management State Diagram is shown in Figure 24. A session can be in the Management substate, once a successful initialization has occurred at which time the Management substate will be in the initial state. The requests are independent and paired, i.e., the response upon leaving the Processing Request state is determined by the request that caused the transition into the Processing Request State.

Once the processing of a request has completed a response is sent and the state transitions to Idle. Transition out of the Idle state occurs upon the client sending a subsequent management request in which case the state transitions to Processing Request. When a TerminateRequest is received, the Catalogue Interface state, which includes the Management substate, transitions from Main to the end state also closing the Management state.

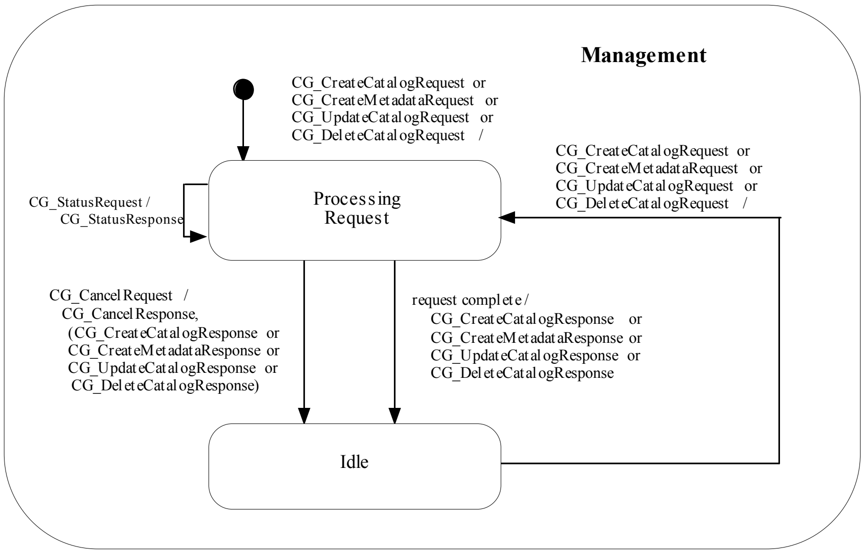
Management
CG_CreateCatalogRequest or
CG_CreateMetadataRequest or
CG_UpdateCatalogRequest or
CG_DeleteCatalogRequest /
CG_StatusRequest /
CG_StatusResponse
Processing Request
CG_CreateCatalogRequest or
CG_CreateMetadataRequest or
CG_UpdateCatalogRequest or
CG_DeleteCatalogRequest /
CG_CancelRequest /
CG_CancelResponse,
(CG_CreateCatalogResponse or
CG_CreateMetadataResponse or
CG_UpdateCatalogResponse or
CG_DeleteCatalogResponse)
request complete /
CG_CreateCatalogResponse or
CG_CreateMetadataResponse or
CG_UpdateCatalogResponse or
CG_DeleteCatalogResponse
Idle

Figure 24 — Management state diagram

7.4.7 Explain state diagram

The Explain State Diagram is shown in Figure 25. A session can be in the Explain substate, once a successful initialization has occurred at which time the Explain substate will be in the initial state. The requests are independent and paired, i.e., the response upon leaving the Processing Request state is determined by the request that caused the transition into the Processing Request State.

Once the processing of a request has completed a response is sent and the state transitions to Idle. Transition out of the Idle state occurs upon the client sending a subsequent explain request in which case the state transitions to Processing Request. When a TerminateRequest is received, the Catalogue Interface state, which includes the Explain substate, transitions from Main to the end state also closing the Explain state.

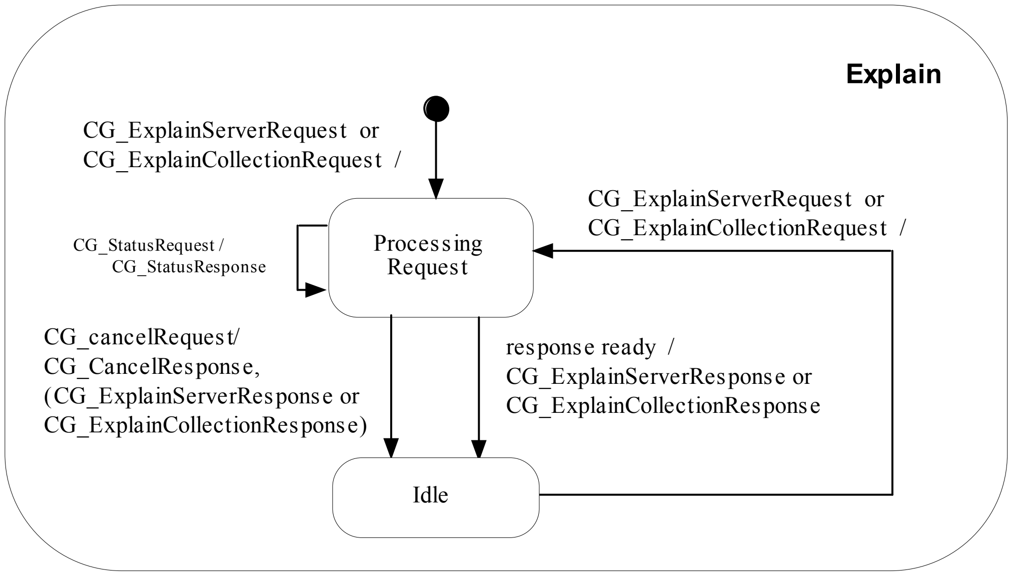
Explain
CG_ExplainServerRequest or
CG_ExplainCollectionRequest /
CG_StatusRequest /
CG_StatusResponse
Processing Request
CG_ExplainServerRequest or
CG_ExplainCollectionRequest /
CG_cancelRequest/
CG_CancelResponse,
(CG_ExplainServerResponse or
CG_ExplainCollectionResponse)
response ready /
CG_ExplainServerResponse or
CG_ExplainCollectionResponse
Idle

Figure 25 — Explain state diagram

8 Z39.50 protocol binding

8.1 Architecture

8.1.1 Introduction

The Z39.50 Protocol binding uses a message-based client server architecture implemented using the ANSI/NISO Z39.50 Application Service Definition and Protocol Specification [*ISO 23950*]. This protocol binding maps each of the general model operations to a corresponding service specified in the ANSI/NISO/ISO standard[*http://lcweb.loc.gov/z3950/agency/document.html*]. For compliance, clients and servers must support Z39.50 Version 3.

At a minimum, Catalogue Services implemented using the Z39.50 protocol binding must support the Discovery and Session operation groupings.

The Z39.50 Protocol binding offers the choice of the following transport mechanisms:

a) Directly over TCP where services are encoded using the Basic Encoding Rules (BER) [*ISO 8825*].

b) Z39.50 – Next Generation Search/Retrieve – Web Service (SRW) or its URL Access Mechanism (SRU) Version 1.0 [http://lcweb.loc.gov/z3950/agency/zing/srw/index.html].

BER over TCP/IP is the historical implementation of Z39.50 transport, whereas SRW/SRU provides a transition strategy from legacy Z39.50/BER to a Web Services interface.

8.1.2 Supported services

Each operation specified in this protocol binding corresponds to a Z39.50 Service, and each consists of paired client requests and server responses in a session-based environment. It is possible to implement "piggy-back present" wherein session facilities are not required. The Z39.50 Services used in this protocol binding include the Init, Search, Present, Resource Control, Trigger Resource Control, Sort, Extended Services and Close.

8.1.3 Core queryable elements

The OGC Catalogue Services specification requires abstract query of a small number of metadata elements for cross-collection, cross-discipline search. These elements are described using the following Use Attribute, Structure, and Relation mappings. Namespace prefixes denote Z39.50 profiles [*http://lcweb.loc.gov/z3950/agency/profiles/profiles.html*].

Table 45 — Correspondence of Z39.50 Attributes to general model equivalents

General model queryable name	Use attribute	Data type	Valid relations
LatLonBoundingBox	BoundingCoordinates (geo:2060)	Coordinate String (geo:201)	Overlaps (geo:7)
CRS3	Any (bib1:1016)	Word (bib-1:2), Word List (bib-1:6)	Equal (bib-1:3)
Keyword	ThemeKeyword (geo:2002), Place Keyword (geo:2042)	Word List (bib-1:6)	Equal (bib-1:3)
Title	Title (bib-1:4)	Word List (bib-1:6)	Equal (bib-1:3)
Abstract	Abstract (bib-1:62)	Word List (bib-1:6)	Equal (bib-1:3)
Format	Geospatial Data Presentation Form (geo:3805)	Word List (bib-1:6)	Equal (bib-1:3)
DCPType	Any (bib1:1016)	Word (bib-1:2), Word List (bib-1:6)	Equal (bib-1:3)

8.2 General model to Z39.50 protocol binding operations mapping

Table 46 provides a mapping between general model operations and the Z39.50 Protocol binding services. The messages listed under the Z39.50 Protocol binding Service column are representative operations from the ISO 23950 standard that provide appropriate functionality. Further interpretation is provided through details in the footnotes. This table is provided to orient the programmer in correspondence with the general model but does not provide parameter-level mapping. This table also only depicts the mandatory (Discovery) catalogue services operations and does not declare equivalence for the optional management and access operations in this version.

[3] No specific Use Attribute exists in Z39.50 as a surrogate for Coordinate Reference System. To support this search, the full-text search (Any) should be applied to include parts of the metdata document that would include authority or namespace plus the value of the CRS, e.g. EPSG:4326). Servers should therefore support search for word/string matches including colon as a recognised character.

Table 46 — General Model to Z39.50 protocol binding operations mapping and obligation

General model operation	Z39.50 BER equivalent	SRW equivalent	Obligation
Session.initialise	initRequest[1]		Conditional, if session used
Session.close	close[2]		Conditional, if session used
OGC_Service.getCapabilities	searchRequest[3, 4]	explainRequest	Conditional, if service supports Explain
Session.status	triggerResourceControlRequest		Optional
Session.cancel	triggerResourceControlRequest		Optional
Discovery.query	searchRequest[3] and sortRequest	SearchRetrieveRequest	Mandatory
Discovery.present	presentRequest	SearchRetrieveRequest	Mandatory
Discovery.describeRecordType	SearchRequest[6]	Inferred by XML namespace reference	Conditional, if service supports Explain
BrokeredAccess.order	ExtendedServicesRequest[7]		Optional

[1] The following init Options are used in this protocol binding: search, present, sort, extended-services, trigger-resource-control, named result sets, and resource-control.

[2] Although Z39.50 permits both the client and server to initiate a Close request, for conformance with the general model, only the client is permitted to initiate a Close request. In practice, a server may terminate a session after a reasonable amount of idle client activity.

[3] Note that the ResultType values of results and hits are supported in this protocol binding. The ResultType values of result set ID and validate are unsupported.

[4] The OGC_Service.getCapabilities is implemented using a searchRequest on the Explain Database with ExplainCategory = TargetInfo and DatabaseInfo.

[5] The CatalogEntryType and QueryScope parameters in the QueryRequest are implemented in the Z39.50 Protocol binding as external elements of the SearchRequest.

[6] The Discovery.describeRecordType is implemented using a searchRequest on the Explain Database with ExplainCategory = TargetInfo and RetrievalRecordDetails.

[7] Brokered Access is implemented in the Z39.50 Protocol binding using the Order Extended Service defined in Subclause 8.3.3. The Order Extended Service uses the Z39.50 Extended Service mechanism.

8.3 Z39.50 BER implementation notes

Z39.50 using Basic Encoding Rules over TCP is implemented using registered profiles listed by the maintenance agency: [http://lcweb.loc.gov/z3950/agency/profiles/profiles.html]. These profiles indicate required operations, sets of registered Use Attributes (search fields), Relations (operators), Z39.50 datatypes, Element Sets (named sets of returned fields/elements), Preferred Syntaxes (encoding format), and information sets (metadata standards and/or schemas returned). The most relevant Community Profiles for the discovery of geospatial data resources are the Geospatial Metadata Profile, GEO, Version 2.2, and the Catalogue Interoperability Profile, CIP, Version 2.4.

In the context of a managed session, the client transmits request messages to the server and the server returns response messages to the client directly over TCP as specified in *IETF RFC 1729: Using the Z39.50 Information Retrieval Protocol in the Internet Environment* [*ftp://ftp.ietf.org/rfc/rfc1729.txt*], where all request and response messages are encoded using BER.

Figure 26 illustrates a typical set of transactions that may occur between a client and server, and between the server and its interface to an external catalog. The client sends an initRequest message to the server, the external system processes the initRequest message by initializing a session with the client and the server returns an initResponse message to the client. This interaction establishes a session in which all subsequent interactions occur.

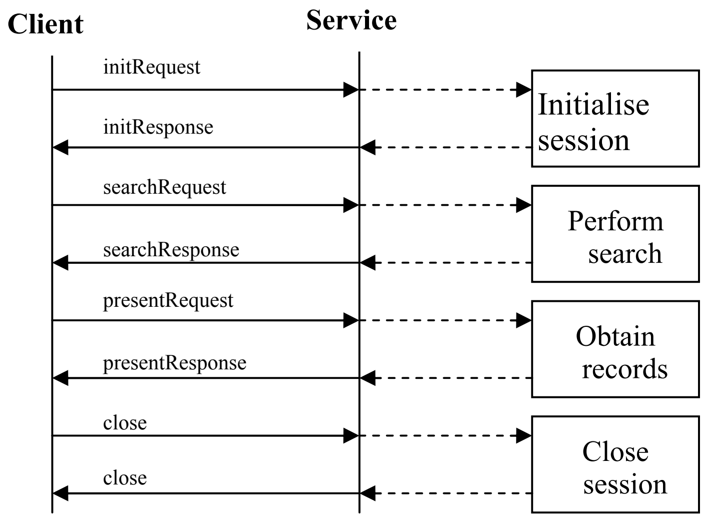

Figure 26 — Z39.50 Protocol binding Sequence Diagram

Next the client constructs a query and sends the query in the searchRequest message to the server. The server runs the search on the external catalogue system, and returns the requested results in the searchResponse message. If the search was successful, a virtual result set is created and the client may request records from the result set using the presentRequest message. In the presentRequest, the client may request any contiguous set of records from the result set (e.g., records 10 through 20). The server returns the records to the client in the presentResponse message. The client may continue to perform additional searches and record retrievals, or may close the session with the server by sending a close message. Optionally, the server may respond with a close message.

8.3.1 Message encoding

For Z39.50 over TCP, messages are encoded using the Basic Encoding Rules (BER) from the ASN.1 specification of Z39.50 available from http://lcweb.loc.gov/z39.50/agency/document.html .

8.3.2 Additional search info

This subclause contains the parameters used in the "otherInfo" part of a Z39.50 searchRequest in order to implement the CatalogEntryType and QueryScope parameters in the QueryRequest of the General Model.

"otherInfo" in a SearchRequest may be used by the origin to specify the scope of a search, i.e., whether the search domain is local to a server or distributed to many servers. This is achieved using the SearchControl EXTERNAL in otherInfo. SearchControl is defined below using ASN.1 notation. If otherInfo is not provided, the type of item descriptors to be searched shall be derived from the query definition and/ or the content of the collection and the default scope of a local search shall be assumed.

The Search Control structure contains two items: itemDescriptorType which maps to CatalogEntryType and searchScope which maps to QueryScope. The CIP-Release-B-APDU {Z39.50-CIP-B-APDU 1} defines the following items:

```
SearchControl ::= SEQUENCE
     {
          itemDescriptorType [1] IMPLICIT INTEGER
               {
                    collectionDescriptorSearch (1),
                    productDescriptorSearch (3),
                    serviceDescriptorSearch (4),
                    catalogDescriptorSearch (5)
               }
          searchScope [2] IMPLICIT INTEGER
               {
                    localSearch (1),
                    wideSearch (2)
               }
     }
```

For further information, see Subclause 3.5.2.5 and Appendix E. 6.1. of Catalogue Interoperability Protocol (CIP) Specification - Release B, CEOS/WGISS/PTT/CIP-B, June 1998, Issue 2.4, Committee on Earth Observation Satellites (CEOS) (ftp://harp.gsfc.nasa.gov/incoming/fed/cip_spec24.pdf).

8.3.3 Order extended service

The Order Extended Service, which is a custom Z39.50 Extended Service, allows an origin to order products previously queried. The Order ES is presented in Table 47.

Further information describing the Order Extended Service can be found in Catalogue Interoperability Protocol (CIP) Specification - Release B, CEOS/WGISS/PTT/CIP-B, June 1998, Issue 2.4, Committee on Earth Observation Satellites (CEOS) (ftp://harp.gsfc.nasa.gov/incoming/fed/cip_spec24.pdf).

Table 47 — Order extended service

ASN.1 definition	Meaning

{Z39.50-CIP-Order-ES} DEFINITIONS ::=
BEGIN
IMPORTS OtherInformation, InternationalString, IntUnit
    FROM Z39.50-APDU-1995;
 	
CIPOrder ::= CHOICE
  {
  esRequest [1] IMPLICIT SEQUENCE{
    toKeep [1] OriginPartToKeep,
    notToKeep [2] OriginPartNotToKeep},
  taskPackage [2] IMPLICIT SEQUENCE{
    originPart [1] OriginPartToKeep,
    targetPart [2] TargetPart}
  }
	The Order Extended Serivce uses the Z39.50 Extended Servi ce Facility.

OriginPartToKeep ::= SEQUENCE
  {
  action [1] IMPLICIT INTEGER {
    orderEstimate (1),
    orderQuoteAndSubmit (2),
    orderMonitor (3),
    orderCancel (4)},
  orderId [2] InternationalString OPTIONAL,
  orderSpecification [3] OrderSpecification OPTIONAL,
  statusUpdateOption [4] StatusUpdateOption OPTIONAL,
  userInformation [5] UserInformation OPTIONAL,
  otherInfo [6] OtherInformation OPTIONAL
  }
	The **OriginPartToKeep** contains the following:
	action, which indicates the type of operation that is requested to be performed for the order request. The supported operations are the following:
	orderEstimate, which is used to validate and obtain the estimate of an order specification.
	orderQuoteAndSubmit, which is used to quote[4] and submit an order specification.
	orderMonitor, which is used to monitor the progress of the processing of an order request.
	orderCancel, which is used to cancel an order request.

[4] The estimate for an order is approximate and non-binding, whereas the quote for an order is precise and binding.

ASN.1 definition	Meaning
	orderId, which is the identifier of the order request as provided as input by the origin. **orderSpecification**, which is the specification of the order request as provided as input by the origin. Note that, in principle, the order request specified by the origin is unstructured, i.e. it contains a list of item descriptor identifiers and the order options related to them, but does not attempt to group them into packages and delivery units. **statusUpdateOption**, which indicates how the origin wishes to be kept up to date as to the status of the order processing. **userInformation**, which contains the personal user information as provided as input by the origin. **otherInformation**, which contains additional information not specified by the CIP.

```
OriginPartNotToKeep ::= SEQUENCE
  {
  orderId            [1] InternationalString  OPTIONAL,
  orderSpecification [2] OrderSpecification   OPTIONAL,
  userInformation    [3] UserInformation      OPTIONAL,
  otherInfo          [4] OtherInformation     OPTIONAL
  }
```
	The **OriginPartNotToKeep5** contains the following:
	orderId, which is the identifier of the order request.
	orderSpecification, which is the specification of the order request.
	userInformation, which contains the personal user information.
	otherInformation, which contains additional information not specified by the CIP.

[5] The definitions used in OriginPartNotToKeep are strictly identical to the ones provided in OriginPartToKeep. The former is used as input by the target (which may overwrite some values as appropriate) for the definition of TargetPart, whereas the latter remains unmodified and is stored in the task package. This duplication therefore allows the comparison of the order as specified by the origin (OriginPartToKeep) with the order as returned by the target (TargetPart).

ASN.1 definition	Meaning

```
TargetPart          ::=        SEQUENCE
  {
  orderId                  [1]   InternationalString,
  orderSpecification       [2]   OrderSpecification          OPTIONAL,
  orderStatusInfo          [3]   OrderStatusInfo        OPTIONAL,
  userInformation          [4]   UserInformation        OPTIONAL,
  otherInfo                [5]   OtherInformation       OPTIONAL
  }
```
	The **TargetPart** contains the following:
	orderId, which is the identifier of the order request as provided as output by the target.
	orderSpecification, which is the specification of the order request as provided as output by the target. This order specification provided by the target overrides the specification provided as input by the origin in **originPartNotToKeep.** It contains the item descriptors and order options supplied as input, with any necessary modifications or additions, in a structured manner, i.e. the item descriptors are grouped into packages and delivery units.
	orderStatusInfo, which indicates the status of the order request being performed6.
	userInformation, which contains the personal user information.
	otherInfo, which contains additional information not specified by the CIP

```
StatusUpdateOption       ::=          CHOICE
  {
  manual                   [1]   NULL,
  automatic                [2]   IMPLICIT INTEGER {
                                 eMail                  (1)}
  }
```
	The **StatusUpdateOption** provides options for how the user will receive updates on the status of an **extended service** request. The parameters are:
	manual the user performs the status request.
	automatic where the OHS filing the order provides status updates for the user via **email7**.

[6] Note the difference between the operationStatus, which is provided in the ES Response, and the orderStatusInfo, which is included in the task package. operationStatus provides status information for the ES operation as a whole and indicates whether the ES operation has been performed successfully or not by the target. orderStatusInfo provides status information for the order specified in the task package and indicates the state of the order or the process being performed for an order at the LOHS.

[7] This could be expanded in the future to include, for example, automatic update via the origin.

ASN.1 definition	Meaning

```
UserInformation ::= SEQUENCE
  {
  userId          [1]  InternationalString,
  userName        [2]  InternationalString   OPTIONAL,
  userAddress     [3]  PostalAddress         OPTIONAL,
  telNumber       [4]  InternationalString   OPTIONAL,
  faxNumber       [5]  InternationalString   OPTIONAL,
  emailAddress    [6]  InternationalString   OPTIONAL,
  networkAddress  [7]  InternationalString
        OPTIONAL,
  billing         [8]  Billing               OPTIONAL
  }
```
	The **Userinformation** structure is presented by the origin part of a request to a target. The information provided contains mandatory fields (the user identifier) and optional fields. The target will allow the **Userinformation** structure contents to be used as an input to the delivery specification for elements which can be altered by the user. The target will refer to the local database contents for the user and will use the contents of the database, or the **Userinformation** structure depending on the privilege of the user to offer alternative information. The **UserInformation** structure consists of the following attributes:
	userId the user identifier, the identifier which the user provides as part of an **InitialiseRequest.**
	userName the full name of the user.
	userAddress a structure to hold the users address.
	telNumber the users telephone number.
	faxNumber the fax number for the user.
	emailAddress the electronic mail address for the user.
	networkAddress the network address to send files to electronically. For Internet addresses, the address is written in URL format to allow directories as well as domain's to be specified.
	billing the method of payment (and hence of billing) available for the user.

```
OrderSpecification ::= SEQUENCE
  {
  orderingCentreId        [1]  InternationalString,
  orderPrice              [2]  PriceInfo            OPTIONAL,
  orderDeliveryDate       [3]  InternationalString
        OPTIONAL,
  orderCancellationDate   [4]  InternationalString  OPTIONAL,
  deliveryUnits           [5]  SEQUENCE OF DeliveryUnitSpec,
  otherInfo               [6]  OtherInformation     OPTIONAL
  }
```
	The **OrderSpecification** is the specification of the order request and contains the following:
	orderingCentreId, which identifies the ordering centre at which the order will be performed.
	orderPrice, which is the price for the whole order.
	orderDeliveryDate, which is the latest date at which the order can be expected to be delivered to the user.
	orderCancellationDate, which is the latest date at which the user can cancel the order.
	deliveryUnits, which contains the definition of the delivery units which compose the order.
	otherInfo, which may be used to provide additional information not specified by the CIP.

ASN.1 definition	Meaning

DeliveryUnitSpec ::= SEQUENCE
{
deliveryUnitId [1] InternationalString OPTIONAL,
deliveryUnitPrice [2] PriceInfo OPTIONAL,
deliveryMethod [3] DeliveryMethod OPTIONAL,
billing [4] Billing OPTIONAL,
packages [5] SEQUENCE OF PackageSpec,
otherInfo [6] OtherInformation OPTIONAL
}
	The **DeliveryUnitSpec** contains the specification of a single delivery unit (i.e. part of an order that is delivered as a unit):
	deliveryUnitId, which is the identifier of the delivery unit.
	deliveryUnitPrice, which is the price of the delivery unit.
	deliveryMethod, which is the method with which the delivery unit is delivered to the user.
	billing, which is the method with which the user is going to be billed.
	packages, which contains the definition of the packages which compose the delivery unit.
	otherInfo, which may be used to provide additional information not specified by the CIP.

DeliveryMethod ::= CHOICE
{
eMail [1] InternationalString,
ftp [2] FTPDelivery,
mail [3] PostalAddress,
otherInfo [4] OtherInformation
}
	The **DeliveryMethod** defines the method with which a delivery unit is delivered to the user and is one of the following:
	eMail, which specifies the email address that the order will be delivered to
	ftp, which specifies that the order will be delivered via ftp, the type of transfer and the ftp address
	mail, which specifies that the order will be delivered via mail and provides the postal address
	otherInfo, which may be used to provide additional information (such as an alternative delivery method) not specified by the CIP.

ASN.1 definition	Meaning

FTPDelivery ::= SEQUENCE
{
transferDirection [1] IMPLICIT INTEGER
{
push (0),
pull (1)
},
ftpAddress [2] InternationalString
}
	The **FTPMethod** defines the method with which a delivery unit is delivered to the user and is one of the following:
	transferDirection, which specifies that the order will be delivered via e-mail.
	ftpAddress, which specifies that the order will be delivered via ftp.

Billing ::= SEQUENCE
{
paymentMethod [1] PaymentMethod,
customerReference [2] IMPLICIT CustomerReference,
customerPONumber [3] IMPLICIT InternationalString OPTIONAL
}
	The **Billing** structure8 contains attributes which describe the method by which a user will pay for a service, together with supporting information regarding the payment. The attributes are:
	paymentMethod indicates the method of payment used.
	customerReference is the customer provided reference for the order.
	customerPONumber is the purchase order provided by the customer for the order.

PaymentMethod ::= CHOICE
{
billInvoice [0] IMPLICIT NULL,
prepay [1] IMPLICIT NULL,
depositAccount [2] IMPLICIT NULL,
privateKnown [3] IMPLICIT NULL,
privateNotKnown [4] IMPLICIT EXTERNAL},
}
	The **PaymentMethod** structure contains attributes which describe the method by which a user will pay for a service. The attributes are:
	billInvoice indicates that an invoice is to be sent to the user (or payee).
	prepay indicates that payment has already been agreed/performed.
	depositAccount indicates that there is a deposit account for the payment.
	privateKnown indicates that the payment method is private and known.
	privateNotKnown contain private unknown payment method information.

[8] The Billing structure used by the Order Extended Service is derived from the addlBilling structure defined in the Item Order ES.

ASN.1 definition	Meaning

CustomerReference ::= SEQUENCE
{
customerId [1] InternationalString,
accounts [2] SEQUENCE OF InternationalString
}
	The **CustomerReference** structure contains attributes which provide a customer reference for the order. The attributes are:
	customerId indicates the customer identifier at the LOHS.
	accounts is the name of the account(s) available to apply charges to on behalf of the user.

PostalAddress ::= SEQUENCE
{
streetAddress [1] InternationalString,
city [2] InternationalString,
state [3] InternationalString,
postalCode [4] InternationalString,
country [5] InternationalString
}
	PostalAddress contains the postal address for a user and consists of:
	streetAddress, which is the street name and number.
	city, which is the name of the city (or nearest city).
	state, which is the name of the state or county.
	postalCode, which is the country specific postal code.
	country, which is the name of the country.

PackageSpec ::= SEQUENCE
{
packageId [1] InternationalString OPTIONAL,
packagePrice [2] PriceInfo OPTIONAL,
package [3] CHOICE
{
predefinedPackage [1] PredefinedPackage,
adHocPackage [2] AdHocPackage
},
packageMedium [4] InternationalString,
packageKByteSise [5] INTEGER,
otherInfo [6] OtherInformation OPTIONAL
}
	The **PackageSpec** contains the specification of a single package (i.e. part of an order that is delivered on a single medium):
	packageId, which is the identifier of the package.
	packagePrice, which is the price of the package.
	package, which contains the specification of the package. The package is one of the following:
	predefinedPackage, which is a package pre-defined by the data provider.
	adHocPackage, which is a package constructed ad-hoc by the data provider to fulfil the order request.
	packageMedium, which is the medium on which the package will be delivered to the user.
	packageKByteSise, which contains the sise of the **package** in kilobytes.
	otherInfo, which may be used to provide additional information not specified by the CIP.

ASN.1 definition	Meaning
`PredefinedPackage ::= SEQUENCE { collectionId [1] InternationalString, orderItems [2] SEQUENCE OF OrderItem, otherInfo [3] OtherInformation OPTIONAL }`	A **PredefinedPackage** contains the definition of a package that is pre-defined by the data provider. A PredefinedPackage is a collection that is stored in advance (i.e. not to fulfil a specific order) on a medium and is defined as follows: **collectionId**, which is the identifier of the pre-packaged collection. Must be formatted according to the naming convention for collection identifiers specified in Appendix E. **orderItems**, which contains the list of the order items contained in the package. **otherInfo**, which may be used to provide additional information not specified by the CIP.
`AdHocPackage ::= SEQUENCE OF OrderItem`	An **AdHocPackage** is a package that is defined ad-hoc by a data provider to fulfil a specific order. An **AdHocPackage** contains the list of the order items contained in the package.
`OrderItem ::= SEQUENCE { productId [1] InternationalString, productPrice [2] PriceInfo OPTIONAL, productDeliveryOptions [3] ProductDeliveryOptions OPTIONAL, processingOptions [5] ProcessingOptions OPTIONAL, sceneSelectionOptions [6] SceneSelectionOptions OPTIONAL, orderStatusInfo [7] OrderStatusInfo OPTIONAL, otherInfo [8] OtherInformation OPTIONAL }`	The **OrderItem** contains the specification of a single order item (i.e. the product that is ordered and that is to be delivered): **productId**, which is the identifier of the ordered product. **productPrice**, which is the price of the product. **productDeliveryOptions**, which contains delivery options for the product. **processingOptions**, which specifies the processing options that are to be applied on the product before delivery. **sceneSelectionOptions**, which specifies the selection of the scene from the whole product that is to be delivered. **orderStatusInfo**, which indicates the status of the order item9. **otherInfo**, which may be used to provide additional information not specified by the CIP.

[9] Note the difference between the orderStatusInfo in TargetPart, which indicates the state, or the process being performed for, an order as a whole at the LOHS, and the orderStatusInfo in OrderItem, which indicates the state, or the process being performed for, a specific order item within an order at the LOHS.

ASN.1 definition	Meaning

ProductDeliveryOptions ::= SEQUENCE
{
productByteSise [1] INTEGER OPTIONAL,
productFormat [2] InternationalString OPTIONAL,
productCompression [3] InternationalString OPTIONAL,
otherInfo [4] OtherInformation OPTIONAL
}
	The **ProductDeliveryOptions** contains the specification of the options regarding the delivery of a product:
	productByteSise, which contains the sise of the product in bytes.
	productFormat, which specifies the format of the product.
	productCompression, which specifies the compression mechanism applied to the product.
	otherInfo, which may be used to provide additional information not specified by the CIP.

ProcessingOptions ::= CHOICE
{
formattedProcessingOptions [1] EXTERNAL,
unformattedProcessingOptions [2] InternationalString
}
	The **ProcessingOptions** specifies the processing options that are to be applied on the product before delivery and is one of the following:
	formattedProcessingOptions, which specifies the processing options according to the format specified in [ORD].
	unformattedProcessingOptions, which specifies the processing options in a free-text form.

SceneSelectionOptions ::= CHOICE
{
formattedSceneSelectionOptions [1] EXTERNAL,
unformattedSceneSelectionOptions [2] InternationalString
}
	The **SceneSelectionOptions** specifies the selection of the scene from the whole product that is to be delivered and is one of the following:
	formattedSceneSelectionOptions, which specifies the scene selection options according to the format specified in [ORD].
	unformattedSceneSelectionOptions, which specifies the scene selection options in a free-text form.

PriceInfo ::= SEQUENCE
{
price [1] IntUnit,
priceExpirationDate [2] InternationalString,
additionalPriceInfo [3] InternationalString OPTIONAL
}
	The **PriceInfo** contains the information related to the price of an item:
	price, which contains the price of the item.
	priceExpirationDate, which specifies the latest date at which the price provided is valid (i.e. until the expiration date the origin is guaranteed that the price will not vary. However, after the expiration date the price may change).
	additionalPriceInfo, which may be used to provide a textual explanation when the price of a item differs from the sum of the elements which compose this item (e.g. it can be used to explain why the price of a delivery unit differs from the sum of the prices of the packages which compose the delivery unit).

ASN.1 definition	Meaning

OrderStatusInfo ::= SEQUENCE
{
orderState [1] CHOICE
{
staticState [1] StaticState,
dynamicState [2] DynamicState
},
additionalStatusInfo [2] InternationalString OPTIONAL
}
	OrderStatusInfo describes the status of an extended service order request. The different status values are:
	orderState indicates the state of the order request or the processing being performed for the order:
	staticState indicates the state of the order when no order request is being performed.
	dynamicState indicates the processing that is currently performed for an order request.
	additionalStatusInfo contains additional status information provided by the LOHS (e.g. to clarify the meaning of the **orderState**).

StaticState ::= [1] IMPLICIT INTEGER
{
orderNotValid (1),
orderEstimated (2),
orderCompleted (3)
}
	StaticState describes the state of an order when no order request is active. The possible states are:
	orderNotValid indicates that the order has not been successfully validated.
	orderEstimated indicates that the order has been successfully validated and that an estimate is provided.
	orderCompleted indicates that the order has been completed.

DynamicState ::= [2] IMPLICIT INTEGER
{
orderBeingEstimated (4),
orderBeingQuoted (5),
orderBeingProcessed (6),
orderBeingCancelled (7),
orderBeingDeleted (8)
}
 	
END
	DynamicState describes the state of an order when an order request is active and thus being process. The possible states are:
	orderBeingEstimated the order is currently being estimated by the target order handling system.
	orderBeingQuoted the order is currently being quoted by the target order handling system.
	orderBeingProcessed the order is currently being processed by the target order handling system.
	orderBeingCancelled the order request which was previously sent to the target is being cancelled.
	orderBeingDeleted the order is being deleted.

8.4 Search/Retrieve Web Service (SRW/SRU) implementation notes

SRW is the “Search/Retrieve Web Service” variant of Z39.50 that implements a simplified, stateless approach to catalogue services that preserves core functions of the Information retrieval protocol but offers them over HTTP. SRW provides Simple Object Access Protocol (SOAP) access to post and receive messages as formatted XML; SRU is a URL-based access method that employs keyword-value pairs (KVP) using GET interfaces. SRU supports a simplified predicate language that can be mapped to the OGC Common Query Language. SRW permits the use of the KVP query or Xpath expressions for search over SOAP.

SRW defines a web service combining several Z39.50 features, most notably, the Search, Present, and Sort Services. Additional features/services may be added later or defined later as new web services. The Z39.50 concepts retained in SRW include result sets, abstract access points, abstract record schemas, explain, and diagnostics. SRW features which differ from Z39.50 include the result set is named by the server rather than the client, lack of connections or sessions, and the fact that a service/server is synonymous with a database or target (services infer a single database). All SRW records are retrieved according to a single record syntax (XML) and therefore the Z39.50 concept of record syntax is not meaningful in SRW. The Z39.50 concepts of element set/specification and schema are represented by XML schemas. Explain information identifies supported access points and record schemas. Finally, XML is used in place of ASN.1 and BER.

Table 48 represents the request arguments for simplified search using SRW.

Table 48 — SearchRetrieve request parameters

Name	Type	Obligation	Description
query	xsi:string	Only one of xQuery or query must be present	Contains a query expressed in CQL to be processed by the server. This parameter may only be present if 'xQuery' is not present.
xQuery	srw:xcqlType	Only one of xQuery or query must be present	Contains a query expressed in XCQL to be processed by the server. This parameter may only be present if 'query' is not present. This parameter is not valid for SRU.
sortKeys	xsi:string	Optional	Contains a sequence of sort keys to be applied to the results, if any. The keys are expressed in the simple string format for sort in SRW.
xSortKeys	srw:xsortType	Optional	Contains a sequence of sort keys to be applied to the results, if any. The keys are expressed in the XML format for sort in SRW. This parameter is not valid for SRU.
startRecord	xsi:integer	Optional	The position within the sequence of matched records of the first record to be returned. The first position in the sequence is 1. The value supplied must be greater than 0. Default value if not supplied is 1.
maximumRecords	xsi:integer	Optional	The number of records requested to be returned. The value must be 0 or greater. Default value if not supplied is determined by the server.
recordSchema	xsi:string	Optional	The schema in which any records should be returned. The value is the URI identifer for the schema. The default value if not supplied is determined by the server.

Table 49 — SearchRetrieve response parameters

Name	Type	Obligation	Description
numberOfRecords	*xsi:integer*	Mandatory	The number of records matched by the query. If the query fails this will be 0.
resultSetId	*xsi:string*	Optional	The identifier for a result set that was created through the execution of the query.
resultSetIdleTime	*xsi:integer*	Optional	The number of seconds in which the created result set will be destroyed. The result set may be destroyed before this by the server.
records	*array of records*	Optional	A sequence of records matched by the query, or surrogate diagnostics.
diagnostics	*array of diagnostics*	Optional	A sequence of non surrogate diagnostics generated during execution.
nextRecordPosition	*xsi:integer*	Optional	The next position within the result set after the final record returned. If there are no remaining records, this value will be 0.
echoedRequest	*xsi:string*	Optional	The request parameters echoed back to the client in a simple XML form.

Explain functionality is further described in the SRW document.

9 CORBA/IIOP protocol binding

9.1 Architecture

This clause describes the CORBA protocol binding. The intention of the CORBA protocol binding is to follow the General Model closely. This enables the building of lightweight bridges between the CORBA protocol binding and the Z39.50 Protocol binding or the HTTP protocol binding. The CORBA protocol binding is described in IDL (interface definition language) of OMG (the Object Management Group). The interfaces follow the General Model as closely as possible. Table 50 provides a mapping between general model operations and the CORBA Protocol binding services.

9.1.1 Supported services

The core of the CORBA protocol binding consists of only one interface: CatalogServices. The separate services of the General Model (Discovery, BrokeredAccess, Manager and Session) are defined in separate interfaces to reflect the General Model. They are all inherited by the central interface CatalogServices. At a minimum, Catalogue Services implemented using the CORBA protocol binding must support the Discovery and Session interfaces as described in Table 50. The Manager- and BrokeredAccess-interfaces are optional.

The operations of CatalogServices, without exception, take a request message as an input parameter and return a response parameter. All messages are filled with standard or compound CORBA structures. Name value pairs, an optional way to transfer meta information, are borrowed from the OMG CORBA 2.3 *Dynamic Any* specification.

9.1.2 Core queryable elements

The OGC Catalogue Services specification requires an abstract query of a small number of metadata elements for cross-collection, cross-discipline search (see Core queryable properties).

9.2 Content types (Catalogue entry types)

The content types define the type of resources a catalogue can contain (parameter "ContentType" in the minimal OGC model). In the former Catalogue Service version the allowed values for this parameter were restricted to: 'product, collection, catalog, service'. Now we have 'product' (maps to 'dataset' in the general model), 'collection' (maps to 'datasetcollection' in the general model), 'service' (as in the general model)

9.3 Supported query languages

The CORBA protocol binding supports the following query languages:

c) CQL (Common Query Language)[10] (mandatory)

d) OGC Filter Encoding[11] (optional)

e) Z39.50 Type-1 (optional)

f) SQL3_SimpleFeature (optional)

9.4 Result set encodings

9.4.1 XML

The default encoding for returning results is XML. The General Model clause describes the common Summary set fields and XML syntax to be supported by all Protocol Bindings (see subclause 6.3).

9.4.2 Name-Value pairs

Additional, the CORBA profile adds a Name-Value (**NV)** entry to the message format enumeration (see 9.6). Specifying NV lets the server return results as name-value pairs. Name-value pairs are specified in the OMG CORBA 2.3 DynamicAny specification, but to be complete, the definition is repeated below and in the IDL. Usage of NameValuePair specification from OMG CORBA 2.3 DynamicAny aligns Catalogue Services CORBA Profile with revision 1.1 (draft 3) of Simple Feature Access for CORBA.

```
enum MessageFormat {XML, HTML, TXT, NV};
module DynamicAny
{
   struct NameValuePair
   {
      string name;
      any value;
```

[10] **Error! Reference source not found.**

[11] Version 1.0.0

```
   };

   typedef sequence<NameValuePair> NameValuePairSeq;
};
```

So if the server gives the results back as XML in the next example:

```
<?xml version="1.0"?>
<!DOCTYPE Metadata SYSTEM "min.dtd" >
<Metadata>
   <Title>Countries of Europe</Title>
   <Abstract>This dataset contains the countries of Europe</Abstract>
   <GeographicBoundingBox>
      <westBoundLongitude>-24.17</westBoundLongitude>
      <eastBoundLongitude>40.71</eastBoundLongitude>
      <northBoundLatitude>71.26</northBoundLatitude>
      <southBoundLatitude>27.63</southBoundLatitude>
   </GeographicBoundingBox>
</Metadata>
```

Name-value pair results are as follows:

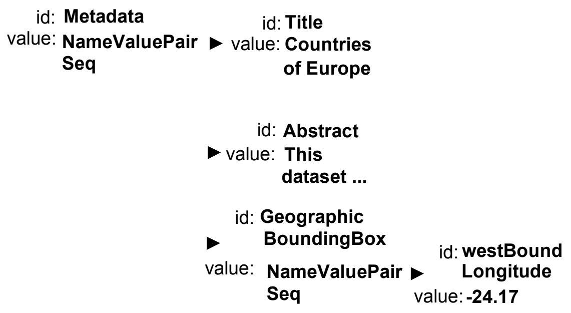

Figure 27 — Name-value pair results

The advantage is that pure CORBA environments do not have to parse the XML to get the results. They receive them in a suitable general structure. If the CORBA server is combined with another type of client, e.g. a Web client, then probably XML (the default) will be preferred.

The **any value** member can contain any type: standard types as long, double, string, types as NameValuePair or NameValuePairSeq (this gives the possibility to create recursive structures) or user-defined types.

9.5 General model to CORBA protocol binding operations mapping

Table 50 provides a mapping between general model operations and the CORBA Protocol binding services. The CORBA Protocol binding messages are defined in Subclause 9.6. The messages listed under the CORBA Protocol binding Equivalent column are the operations that provide appropriate functionality used in the CORBA interfaces. Further interpretation is provided through details in the footnotes. This table is provided to orient the programmer in correspondence with the general model but does not provide parameter-level mapping.

Table 50 — General Model to CORBA protocol binding operations mapping and obligation

General model operation	CORBA protocol binding equivalent	Obligation
Session.initialize	OGC_StatefulService. initSession	Conditional, if session used
Session.close	OGC_StatefulService. terminateSession[2]	Conditional, if session used
OGC_Service.getCapabilities	OGC_Service. explainServer	Mandatory
Session.status	OGC_StatefulService.status	Optional
Session.cancel	OGC_StatefulService.cancel	Optional
Discovery.query	CG_Discovery. query [3]	Mandatory
Discovery.present	CG_Discovery .present	Mandatory
Discovery.describeRecordType	CG_Discovery . explainCollection	Conditional, if service supports Explain
Discovery.getDomain	CG_Discovery.getDomain	Conditional, if service supports getDomain
Manager.transaction	CG_Manager. transaction	Optional
Manager.harvestRecords	--	--
BrokeredAccess.order	CG_Access. brokeredAccess	Optional

[2] Although the CORBA protocol binding permits both the client and server to initiate a terminateSession request, for conformance with the general model, only the client is permitted to initiate a terminateSession request. In practice, a server may terminate a session after a reasonable amount of idle client activity.

[3] Note that the ResultType values “results”, “hits”, “resultSetID” and “validate” are supported in this protocol binding.

9.6 Interface definition - IDL

9.6.1 Introduction

This subclause describes the CORBA IDL. It first describes enumerations and then structures, unions, and messages, respectively. It concludes with a description of the CatalogServices interface, the core of the profile, and other interfaces.

All enumerations, structures, unions, messages and interfaces are part of the OGC_CatalogService module. Module names have to be harmonized across all OGC CORBA specifications and have to be prefixed by opengis.org.

```
#pragma prefix "opengis.org"
module OGC_CatalogService
{
...
};
```

Throughout the module OGC_CatalogService the IDL types wstring and wchar are used instead of string and char to allow usage of different character codesets (other than Unicode) for internationalization (i18n).

In CORBA IDL type definitions for sequences containing different element datatypes are used to avoid anonymous sequences in IDL mappings for some programming languages.

9.6.2 Enumerations

Enumerations can be modeled by a direct translation of all code-lists of the General Model. The following enumerations are borrowed literally:

```
enum AttributeCategory {queriable, presentable, both};
enum CatalogEntryType {product, collection, service};
enum CharacterSet {ASCII, UniCode, ShiftJIS};
enum PredefinedPresentationType {full, summary, brief};
enum QueryLanguage {OGC_Common, OGC_Filter, Z3950_TypeOne
SQL3_SimpleFeature };
enum QueryScope {distributed, locale};
enum ResultType {validate, resultSetID, hits, results};
enum SortOrder {ascending, descending};
enum Status {success, successResultsAvailable, processingNormal,
processingQueued, processingPausedOrSuspended, failure,
failureAccessDenied};
```

Additional, the CORBA profile adds an **NV** entry to the message format enumeration (see 9.4.2):

enum MessageFormat {XML, HTML, TXT, NV};

9.6.3 Structures and unions

Most of the structures and unions from the General Model can be translated directly into CORBA structs and unions. Here the collectionID of the general model is translated as follows:

```
union CollectionName
   switch(long)
   {
      case 1 : wstring collectionID;
      case 2 : wstring collectionName;
   };
```

A special capability is present in QueryExpression in the CORBA Profile that allows passing of parameters that can't be converted to strings but must be bound to variables in string *theQuery* (e.g. "?" in JDBC). For example, references or handles for metadata retrieved from related collections in previous queries. *queryParameters* might contain a NameValuePairSeq or non ASCII XML Data. The additional member aligns Catalogue Services query facilities with respective Simple Feature Access for CORBA query facilities.

```
struct QueryExpression
{
   wstring theQuery;
   wstring theNamespace;
   QueryLanguage theLanguage;
  any queryParameters;
};
```

To allow for globally unique *sessionID* a long long (Long) is used as datatype instead of long (Integer).

```
struct RequestID
{
   long long sessionID;
   long counter;
};
struct SortField
{
   wstring attributeName;
   SortOrder sortOrder;
};
```

The CORBA protocol binding specifies an **any** structure member for the retrievedData in a way that strings (e.g. xml) or name-value pairs or sequences can be stored.

```
struct ReturnData
{
   MessageFormat encoding;
   any payload;
};
```

The responseElements in the general model specify a set name or a list of metadata elements to be returned in the context of a specific metadata structure. In CORBA IDL this is specified by a union which can represent a sequence of attribute names or alternative a PredefinedPresentationType.

```
typedef sequence<wstring> StringSeq;
union PresentationDescription
   switch(long)
   {
      case 1 : StringSeq attributes; // TupleType in GM
      case 2 : PredefinedPresentationType presentationType;
   };
```

The SchemeID structure uses a structure member Schema. This is in the CORBA profile defined as a sequence of name-value pairs from the OMG CORBA 2.3 DynamicAny module. All names, types, and used sequences can be specified in name-value pairs. A schema, tuple-type or a dictionary is not needed here.

```
typedef DynamicAny::NameValuePairSeq Schema;
struct SchemaID
{
   wstring schemeName;
   Schema schema;
};
```

The getDomain operation in the general model specifies descriptions of domains of one or more requested metadata properties or requested parameters in their responseElements which are implemented as follows:

```
   // Enumeration for the type of the metadata property or request
parameter
```

```
enum DomainType
   { domainTypeString, domainTypeDate, domainTypeInteger, domainTypeReal,
            domainTypeSpatial };

   // Enumeration for the type how valid values can be described
enum DomainValuesType
      { valueRef, valueRange };

// valid range, composed of lower and upper boundary value
struct ValueRange
{
   wstring lowerValue;
   wstring upperValue;
};
typedef sequence<ValueRange> ValueRangeSeq;

// Definition of the real type by totalDigits and fractionDigits
   struct RealTypeDef
   {
      long totalDigits;
      long fractionDigits;
   };

// type definition of the metadata property or request parameter
union DomainDef
   switch(DomainType)
   {
      case domainTypeString : long length;
      case domainTypeDate : wstring dateFormat;
      case domainTypeInteger : long totalDigits;
      case domainTypeReal : CG_RealTypeDef realTypeDef;
      case domainTypeSpatial : long dimensions;
   };

// definition of valid values (range, value list) of the type
union DomainValueDef
   switch(DomainValuesType)
   {
      case valueRef : StringSeq valueList;
      case valueRange : ValueRangeSeq rangeList;
   };

// definition of domain values (typical value, valid values, value name,
value description) of the type
struct DomainValue
{
   wstring value;
   DomainValueDef valuesDef;
   wstring title;
   wstring description;
   wstring metadataURL;
};
typedef sequence<DomainValue> DomainValueSeq;

// Descriptions of domain of one requested metadata properties or request
parameter
struct Domain
```

```
{
      wstring attributeName;
      DomainDef domainDef;
      DomainValueSeq domainValue;
   };
   typedef sequence<Domain> DomainSeq;
```

9.6.4 Definitions for brokered access

The General Model defines some code-lists and structures for brokered access. These definitions are directly translated into their CORBA counterparts:

```
enum BrokeredAccessRequestType {orderEstimate, orderQuoteAndSubmit,
         orderMonitor, orderCancel};
struct OrderItem
{
   // Note: datatypes not provided by GM
   any productID;
   any productPrice;
   any productDeliveryOptions;
   any processingOptions;
   any sceneSelectionOptions;
};
struct OrderSpecification
{
   // Note: datatypes not provided by GM
   any orderCentreID;
   any orderPrice;
   any orderDeliveryDate;
   any orderCancellationDate;
   any deliveryMethod;
   any package;
};
enum OrderStatus {orderBeingEstimated, orderEstimated,
   orderBeingQuoted, orderBeingProcessed,
   orderCompleted, orderNotValid, orderCancelled};
enum PackagingType {predefinedPackage, adhocPackage};
struct PackageSpecification
{
   // Note: datatypes not provided by GM
   any packageId;
   any packagePrice;
   PackagingType package;
   any packageMedium;
   long packageSize;
};
enum PaymentMethod {credit, cash, purchaseOrder};
enum StatusUpdateType {manual, automatic};
struct UserInformation
{
   wstring userName;
   wstring userAddress;
   wstring phoneNumber;
   wstring faxNumber;
   wstring emailAddress;
   wstring netAddress;
   PaymentMethod paymentMethod;
};
```

9.6.5 Capabilities

The capabilities in the General Model are designed with inheritance. In CORBA designing capabilities as interfaces can reflect this, but this is not useful. Capabilities like messages (see below) have to be transferred over the network. Therefore, they are defined as either type definitions or structures.

```
typedef boolean AllSupportedRequest;
typedef boolean Defaults;
struct DefaultTimeOut
{
   unsigned long long timeOut;
   // used to be OGC_Basic::UomTime, but OGC_Basic is no longer maintained
as normative part of the Catalogue Services Specification
};
typedef boolean Explain;
struct Messaging
{
   CharacterSet characterSet;
   MessageFormat messageFormat;
};
struct Query
{
   wstring version;
   CharacterSet characterSet;
   QueryLanguage queryLanguage;
};
struct Session
{
   wstring language;
   wstring catalogSpecificationVersion;
   CharacterSet characterSet;
};
struct SoftwareInformation
{
   wstring vendor;
   wstring SVversionNumber;
   wstring IFversionNumber;
};
typedef sequence<CollectionName> SupportedCollections;
```

To be able to make a sequence of different capabilities, a union Capability is created, encompassing all derived capabilities.

A union normally has a discriminator. This can be a **long** value, but this is generally not preferred because you have to remember the value indicating the intended capability. Therefore, an enumeration of capabilities is included in the CORBA profile.

```
enum CapabilityType
   { ctAllSupportedRequest, ctDefaults, ctDefaultTimeOut, ctExplain,
ctMessaging, ctQuery, ctSession, ctSoftwareInformation,
ctSupportedCollections
};
```

```
union Capability
   switch(CapabilityType)
   {
      case ctAllSupportedRequest : AllSupportedRequest
allSupportedRequest;
      case ctDefaults : Defaults defaults;
      case ctDefaultTimeOut : DefaultTimeOut timeOut;
      case ctExplain : Explain explain;
      case ctMessaging : Messaging messaging;
      case ctQuery : Query query;
      case ctSession : Session session;
      case ctSoftwareInformation : SoftwareInformation
softwareInformation;
      case ctSupportedCollections : SupportedCollections
supportedCollections;
   };
```

9.6.6 General messages

The General Model is a message-based model, where messages are designed in the form of a class hierarchy. In CORBA IDL, the messages are translated as structs. Writing them in the form of interfaces is not useful. In CORBA, the objects (instances of interfaces) stay on a remote server machine and are referred to by a client machine. They are not transferred over the network. This is definitely not the intention for messages.

All messages have the same form as the messages described in the General Model. However, messages in the form of structs cannot inherit from each other in CORBA. Therefore the Message class is also included in the CORBA profile and a member of all other messages, called 'base'.

```
struct Message
{
   long long sessionID;
   wstring destinationID;
   RequestID requestID;
   wstring additionalInfo;
};
```

All other messages, which in the General Model inherit from Message, have in the CORBA profile the Message as a structure member. The next messages do not add extra structure members. Alternatively, they might have been modeled by a typedef. But to be consistent with the rest of the messages these message have **base** as a structure member.

Note that the response in the General Model also contains a string structure member **diagnostic**. This parameter is not specified in the CORBA profile. Error handling will be handled by exceptions, the standard CORBA facility. Exceptions are described below. WWW/CORBA bridges can catch these exceptions and convert them into diagnostic info if necessary.

```
struct InitSessionRequest
{
   Message base;
};
struct InitSessionResponse
{
   Message base;
};
```

```
struct TerminateRequest
{
   Message base;
};
struct TerminateResponse
{
   Message base;
   Status status;
};
```

The status and cancel messages add a few structure members in addition to the **base** structure member.

```
struct StatusRequest
{
   Message base;
   RequestID requestIDtoStatus;
};
struct StatusResponse
{
   Message base;
   RequestID requestIDtoStatus;
   Status status;
};
struct CancelRequest
{
   Message base;
   RequestID requestIDtoCancel;
   boolean freeResources;
};
struct CancelResponse
{
   Message base;
   Status status;
   RequestID canceledRequest;
};
```

The explain server messages add sequence of capabilities to the base message. The capability-type sequence can be filled with capability-types to specify which capabilities are requested from the server. The server responds with reporting each capability in a sequence of capabilities.

```
   typedef sequence<CapabilityType> CapabilityTypeSeq;
   struct ExplainServerRequest
   {
      Message base;
      CapabilityTypeSeq capabilities;
   };
   struct ExplainServerResponse
   {
      Message base;
      CapabilityTypeSeq capabilities;
   };
```

9.6.7 Discovery messages

There are four request/response message pairs in the discovery service. To enhance distributed searching, an additional structure member for the query message is provided. This member is not included in the General Model. This structure member **asynchronous** can be set to **true** to force asynchronous searching. The query method will return immediately, setting structure member **hits** in the response to zero. Query results can be retrieved later on, when the query is ready. The progress of the query can be examined with the status messages. The query can be cancelled with the cancel messages.

NOTE 1 This asynchronous behaviour is only specified for the query request message. All other operations (e.g. init, terminate, status, cancel, explain, present) are not considered as time-consuming and return immediately after processing.

Another structure member, **maxLevel**, is added to have more control in the range of the distribution. If one catalogue contains another one, that other one contains a third one, and so on, you will possibly specify that only two levels of sub-catalogs will be searched. Setting the **maxLevel** member to two will force this. Setting **maxLevel** to -1 forces searching all sub-catalogs.

NOTE 2 If the **queryScope** is **Local** there is no distributed search at all.

```
typedef sequence<SortField> SortFieldSeq;
struct QueryRequest
{
   Message base;
   QueryExpression queryExpression;
   ResultType resultType;
   long iteratorSize;
   long cursor;
   MessageFormat returnFormat;
   PresentationDescription presentation;
   SortFieldSeq sortField;
   QueryScope queryScope;
   CollectionName collectionID;
   CatalogEntryType resourceType;
   boolean asynchronous;
   long maxLevel;
};
struct QueryResponse
{
   Message base;
   ReturnData retrievedData;
   CollectionName resultSetID;
   Status status;
   long hits;
   long cursor;
};
struct PresentRequest
{
   Message base;
   CollectionName resultSetID;
   PresentationDescription presentation;
   SortFieldSeq sortField;
   MessageFormat returnFormat;
   long iteratorSize;
   long cursor;
};
```

```
struct PresentResponse
{
   Message base;
   ReturnData retrievedData;
   long cursor;
   long hits;
   Status status;
};
struct ExplainCollectionRequest
{
   Message base;
   AttributeCategory attributeCategory;
   CollectionName collectionID;
   MessageFormat returnFormat;
};
struct ExplainCollectionResponse
{
   Message base;
   CollectionName collectionID;
   SchemaID dataModel;
   Status status;
};
struct GetDomainRequest
{
   Message base;
   StringSeq attributes;
};

struct GetDomainResponse
{
   Message base;
   DomainSeq attributeDomains;
};
```

9.6.8 Management messages

The General Model defines messages for managing catalogs. These messages are translated to the CORBA profile literally.

```
typedef sequence<DynamicAny::NameValuePairSeq> NameValuePairSeqSeq;

   // Messages for managing functions

   // Insert-Structure for inserting metadata (payload) to a collection
   // of a defined catalogType in a predefined format (encoding)
   struct InsertMetadata
   {
      CollectionName collectionID;
      CatalogEntryType catalogType;

      MessageFormat encoding;

      // the data to insert, e.g. a list of NV-Pairs or ISO19139-
      // XML
      any payload;
   };
```

```
// Update-Structure for updating a certain set (defined by the
// queryExpression) of metadata (payload) of a
// collection/catalogType in a predefined format (encoding)
   struct UpdateMetadata
   {
      CollectionName collectionID;
      CatalogEntryType catalogType;

      MessageFormat encoding;

      QueryExpression queryExpression;

      // the data to insert, e.g. a list of NV-Pairs or ISO19139-
      // XML
      any payload;
   };

   // Delete-Structure for deleting a certain set (defined by the
// queryExpression) of metadata of a
// collection/catalogType in a predefined format (encoding)
   struct DeleteMetadata
   {
      CollectionName collectionID;
      CatalogEntryType catalogType;

      QueryExpression queryExpression;
   };

   // Enumeration for the type of a single manipulation in a
   // transaction
   enum MetadataManipulationType
      { ctInsert, ctUpdate, ctDelete };

   // Structure for the content of a single manipulation in a
   // transaction
   union MetadataManipulation
      switch(MetadataManipulationType)
      {
         case ctInsert : InsertMetadata insertData;
         case ctUpdate : UpdateMetadata updateData;
         case ctDelete : DeleteMetadata deleteData;
      };

   typedef sequence<MetadataManipulation> MetadataManipulationSeq;

   // Struct of a transaction, as a sequence of single manipulations
   struct Transaction
   {
      MetadataManipulationSeq manipulations;
   };

   struct TransactionRequest
   {
      Message base;
      Transaction transactionData;
   };
   struct TransactionResponse
   {
      Message base;
      Status status;
      // value of -1 means not calculated
      long totalInserted;
      long totalUpdated;
      long totalDeleted;
      // List of newly generated catalogue entry identifiers
      // assigned to the new catalogue entry instances
      NameValuePairSeqSeq newKeyList;
   };
```

9.6.9 Access messages

The General Model specifies direct access and brokered access. Direct access is provided by interfaces such as the OGC Simple Features and Coverage interfaces for CORBA. If a catalogue entry denotes an OGC Feature, a Feature Collection or a Coverage, the meta-information of this entry can be populated with an **ior** (interoperable object reference). This meta-information entity is called **ior** and is filled with the standard representation of an **ior**, specified by the OMG (Object Management Group), the creators of CORBA. In XML this looks like the following (abbreviated) example:

```
   <ior>IOR:010631002800000049444c3a6f6d672e6f...</ior>
```

Brokered access is specified by a request and a response message, conform all operations of the General Model. The messages are listed below.

```
struct BrokeredAccessRequest
{
   Message base;
   wstring productHandle;
   OrderSpecification orderInformation;
   wstring orderID;
   BrokeredAccessRequestType requestType;
   UserInformation userInformation;
   StatusUpdateType statusOrderUpdateType;
   PackageSpecification packageSpecification;
};
typedef sequence<long> LongSeq;
struct BrokeredAccessResponse
{
   Message base;
   OrderStatus orderStatus;
   LongSeq resourceEstimate;
   CollectionName order;
   wstring orderID;
   Status status;
   BrokeredAccessRequestType requestType;
};
```

9.6.10 Exceptions

Exceptions are not specified in the General Model because they are profile specific. In CORBA exceptions are considered as an appropriate way to notify error situations to clients. The CORBA profile specifies exceptions. The **diagnostic** structure member of the response messages are not used in the CORBA profile, their role is taken over by the exceptions. Some exceptions specify the **diagnostic** (w) string as an exception parameter. By other exceptions this is not necessary, as the exceptions are self-explaining.

```
exception InvalidRequest{};
exception InvalidSession{};
exception InvalidCollection{ wstring diagnostic; };
```

The exception InvalidQuery is thrown if the client specifies an invalid query.

NOTE The exception is not thrown if the **resultType** field is set to **validate**.

```
exception InvalidQuery{ wstring diagnostic; };
```

The exception NotImplemented is defined in cases where the client asks for not-implemented behavior. This might occur by requesting the optional access or management services.

```
exception NotImplemented{ wstring diagnostic; };
```

The NotSupported exception is thrown if the client specifies something in a request parameter that is not implemented by the server. For example the client can specify its query in Z3950_TypeOne, but the server can only interpret OGC_Common queries.

```
exception NotSupported{ wstring diagnostic; };
```

The last exception, CatalogError, indicates an error when none of the above exceptions is appropriate.

```
exception CatalogError{ wstring diagnostic; };
```

9.6.11 Catalogue Service interfaces

The interface Discovery implements methods for discovery: **query**, **present, explainCollection and getDomain**. These methods take a request message as input parameter and return a response message as output parameter.

```
   interface Discovery
   {
      QueryResponse query(in QueryRequest request)
            raises(InvalidSession, InvalidQuery, InvalidCollection,
NotSupported, CatalogError);
      PresentResponse present(in PresentRequest request)
            raises(InvalidSession, InvalidCollection, NotSupported,
CatalogError);
      ExplainCollectionResponse explainCollection(in
ExplainCollectionRequest request)
            raises(CatalogError);
      GetDomainResponse getDomain(in GetDomainRequest request)
            raises(CatalogError);
   };
```

The next interface describes the Manager interface, which defines catalogue management functions. The operation transaction is taken literally from the General Model. This operation can create, update, or delete catalogue entries. The appropriate meta information will be provided in the request messages.

```
   interface Manager
   {
      // This operation is used to by a client that has the appropriate
      // user privileges to execute a whole set of insert, update and
      // delete operation of metadata to a catalog.
      TransactionResponse
            transaction(in TransactionRequest request)
               raises(NotImplemented, CatalogError);
   };
The interface Access is the interface for access messages. It describes
only one operation: the brokeredAccess function which has the request as
input and which returns the response. Direct access is provided by
interfaces as the Simple Feature interface and the Coverage interface.
These interfaces are not described here. The client can get a reference to
these interfaces by examining the ior field in the meta-information.
interface Access
{
   BrokeredAccessResponse
         brokeredAccess(in BrokeredAccessRequest request)
            raises(NotImplemented, CatalogError);
};
```

The OGC_StatefulService interface provides four operations for interactive sessions between a server and a client. All operations have a comparable form of the operations specified in the General Model.

```
   interface OGC_StatefulService : OGC_Service
   {
InitSessionResponse initSession(in InitSessionRequest
request)
            raises(CatalogError);
TerminateResponse terminateSession(in TerminateRequest
request)
            raises(InvalidSession, CatalogError);
      StatusResponse status(in StatusRequest request)
            raises(InvalidSession, InvalidRequest,
CatalogError);
      CancelResponse cancel(in CancelRequest request)
            raises(InvalidSession, InvalidRequest,
CatalogError);
   };
```

The CatalogServices interface is the core of the CORBA profile.

The CatalogServices inherits from the interfaces Discovery, Access and Manager. In this way these services are realized.

NOTE Access and manager services are optional. If a server does not implement these services it throws the exception **NotImplemented**.

The CatalogServices also inherits from OGC_StatefulService that is described below.

```
   interface CatalogServices : OGC_StatefulService, Discovery, Access,
Manager
   {
   };
```

9.6.12 Basic interfaces

Because of the asynchronous behavior of the query operation, a callback notifying the termination of the query might be useful. The Observer Design Pattern [GAMMA97] describes a standard mechanism for notifications to one or more clients. We envision that such a mechanism will be useful for many operations in the OpenGIS world. Therefore the OGC_Observer and the OGC_Subject interfaces are modeled separately. These interfaces might be moved to an OGC general module, in the same or a similar form. The next interfaces describe the mechanism.

NOTE They are not mentioned in the General Model, as this is a CORBA specific behaviour.

```
interface OGC_Observer;
interface OGC_Subject
{
   void attachObserver(in OGC_Observer Observer);
   void detatchObserver(in OGC_Observer Observer);
   void notifyObserver();
};
interface OGC_Observer
{
   void updateSubject(in OGC_Subject ChangedSubject);
};
```

The CatalogServices interface inherits from **OGC_Service**. This is envisioned as the basic interface for all OpenGIS services. As it does not exist yet, the content of this interface is not clear.

```
interface OGC_Service : OGC_Subject
{
};

   interface OGC_StatefulService : OGC_Service
   {
      InitSessionResponse initSession(in InitSessionRequest request)
            raises(CatalogError);
      TerminateResponse terminateSession(in TerminateRequest request)
            raises(InvalidSession, CatalogError);
      StatusResponse status(in StatusRequest request)
            raises(InvalidSession, InvalidRequest, CatalogError);
      CancelResponse cancel(in CancelRequest request)
            raises(InvalidSession, InvalidRequest, CatalogError);
   };\
```

9.6.13 Complete IDL

```
//--------------------------------------------------------------------
-----------------
// Module       : CORBA protocol binding of the OpenGIS Catalogue Services
Specification 2.0
//                described in IDL (interface definition language) of the
OMG (the Object
//                Management Group).
//--------------------------------------------------------------------
-----------------
// Purpose      : The intention of this CORBA protocol binding is to
follow the General
//                Model closely.
```

```
//------------------------------------------------------------------------
-----------------
// Authors      :
//      Uwe Voges, con terra GmbH, Germany
//      Barend Gehrels, Geodan IT b.v., the Netherlands
//                Joined Catalogue Response Team
// Date         : july 13, 1999
//                july 26, 1999:  errata based upon minor GM changes
//                july 30, 2000:  Juergen Ebbinghaus (SICAD)
//                                and Barend Gehrels:
//                                changes based on SICAD Review
//                                - string -> wstring
//                                - long SessionID -> long long
//                                 - e.g. sequence<type> TypeSeq
//      april 2, 2003:  Uwe Voges (con terra)
//            added management-/transaction interface
//      feb 17,  2004:  Uwe Voges (con terra)
//            adapted to 2.0 general model: new types for new operations
like
//            getDomain,...
//      april 29, 2004: Uwe Voges (con terra)
//            minor changes for CS 2.0 r2
//------------------------------------------------------------------------
-----------------

#pragma prefix "opengis.org"

module DynamicAny
{
   struct NameValuePair
   {
      string name;
      any value;
   };

   typedef sequence<NameValuePair> NameValuePairSeq;
};

module OGC_CatalogService
{
   //------------------------------------------------------------------
   // Parameter type definitions
   //------------------------------------------------------------------
   enum CG_AttributeCategory {queriable, presentable, both};
   enum CG_BrokeredAccessRequestType {orderEstimate, orderQuoteAndSubmit,
           orderMonitor, orderCancel};
   enum CG_CatalogEntryType {product, collection, service};
   enum CG_CharacterSet {ASCII, UniCode, ShiftJIS};
   union CG_CollectionName
      switch(long)
      {
         case 1 : wstring collectionID;
         case 2 : wstring collectionName;
      };
   enum CG_MessageFormat {XML, HTML, TXT, NV};

   struct CG_OrderItem
   {
      // Note: datatypes not provided by GM
      any productID;
      any productPrice;
      any productDeliveryOptions;
      any processingOptions;
      any sceneSelectionOptions;
   };

   struct CG_OrderSpecification
   {
      // Note: datatypes not provided by GM
      any orderCentreID;
      any orderPrice;
      any orderDeliveryDate;
      any orderCancellationDate;
      any deliveryMethod;
      any package;
   };

   enum CG_OrderStatus {orderBeingEstimated, orderEstimated,
      orderBeingQuoted, orderBeingProcessed,
      orderCompleted, orderNotValid, orderCancelled};

   enum CG_PackagingType {predefinedPackage, adhocPackage};
   struct CG_PackageSpecification
   {
      // Note: datatypes not provided by GM
      any packageId;
      any packagePrice;
      CG_PackagingType package;
      any packageMedium;
      long packageSize;
   };

   enum CG_PaymentMethod {credit, cash, purchaseOrder};

   enum CG_PredefinedPresentationType {full, summary, brief};

   typedef sequence<wstring> StringSeq;

   union CG_PresentationDescription
      switch(long)
      {
         case 1 : StringSeq attributes; // CG_TupleType in GM
         case 2 : CG_PredefinedPresentationType presentationType; // name
in GM
      };

   enum CG_QueryLanguage {OGC_Common, OGC_Filter, Z3950_TypeOne,
SQL3_SimpleFeature};
   struct CG_QueryExpression
   {
      wstring theQuery;
      wstring theNamespace;
      CG_QueryLanguage theLanguage;
      any queryParameters;
   };

   enum CG_QueryScope {distributed, locale};

   struct CG_RequestID
   {
      long long sessionID;
      long counter;
   };

   enum CG_ResultType {validate, resultSetID, hits, results};

   struct CG_ReturnData
   {
      CG_MessageFormat encoding;
      any payload;
      // XML,HTML,TXT will return a string
      // NV will return a DynamicAny::NameValuePairSeq (from CORBA 2.3
Dynamic Any)
   };

   typedef DynamicAny::NameValuePairSeq CG_Schema;
   struct CG_SchemaID
   {
      wstring schemeName;
      CG_Schema schema;
   };

   enum CG_SortOrder {ascending, descending};
   struct CG_SortField
   {
      wstring attributeName;
      CG_SortOrder sortOrder;
   };

   enum CG_Status {success, successResultsAvailable, processingNormal,
processingQueued, processingPausedOrSuspended, failure,
failureAccessDenied};

   enum CG_StatusUpdateType {manual, automatic};

   struct CG_UserInformation
   {
      wstring userName;
      wstring userAddress;
      wstring phoneNumber;
      wstring faxNumber;
      wstring emailAddress;
      wstring netAddress;
      CG_PaymentMethod paymentMethod;
   };
   //----------------------------------------------------------------
   // Capabilities, 3.2.7.3
   //----------------------------------------------------------------
   enum CG_CapabilityType
      { ctAllSupportedRequest, ctDefaults, ctDefaultTimeOut,
         ctExplain, ctMessaging, ctQuery, ctSession,
         ctSoftwareInformation, ctSupportedCollections };

   typedef boolean CG_AllSupportedRequest;

   typedef boolean CG_Defaults;

   struct CG_DefaultTimeOut
   {
      unsigned long long timeOut;
   };
   typedef boolean CG_Explain;
   struct CG_Messaging
   {
      CG_CharacterSet characterSet;
      CG_MessageFormat messageFormat;
   };

   struct CG_Query
   {
      wstring version;
      CG_CharacterSet characterSet;
      CG_QueryLanguage queryLanguage;
   };

   struct CG_Session
   {
      wstring language;
      wstring catalogSpecificationVersion;
      CG_CharacterSet characterSet;
   };

   struct CG_SoftwareInformation
   {
      wstring vendor;
      wstring SVversionNumber;
      wstring IFversionNumber;
   };

   typedef sequence<CG_CollectionName> CG_SupportedCollections;

   union CG_Capability
      switch(CG_CapabilityType)
      {
         case ctAllSupportedRequest : CG_AllSupportedRequest
allSupportedRequest;
         case ctDefaults : CG_Defaults defaults;
         case ctDefaultTimeOut : CG_DefaultTimeOut timeOut;
         case ctExplain : CG_Explain explain;
         case ctMessaging : CG_Messaging messaging;
         case ctQuery : CG_Query query;
         case ctSession : CG_Session session;
         case ctSoftwareInformation : CG_SoftwareInformation
softwareInformation;
         case ctSupportedCollections : CG_SupportedCollections
supportedCollections;
      };
   //-----------------------------------------------------------------
   // DomainType
   //-----------------------------------------------------------------

   // Enumeration for the type of the metadata property or request
parameter
   enum CG_DomainType
      { domainTypeString, domainTypeDate, domainTypeInteger,
domainTypeReal,
         domainTypeSpatial };

   // Enumeration for the type how possible values can be described
   enum CG_DomainValuesType
      { valueRef, valueRange };

   // valid range, composed of lower and upper boundary value
   struct CG_ValueRange
   {
      wstring lowerValue;
      wstring upperValue;
   };
   typedef sequence<CG_ValueRange> CG_ValueRangeSeq;

   // Definition of the real type by totalDigits and fractionDigits
   struct CG_RealTypeDef
   {
      long totalDigits;
      long fractionDigits;
   };

   // type definition of the metadata property or request parameter
   union CG_DomainDef
      switch(CG_DomainType)
      {
         case domainTypeString : long length;
         case domainTypeDate : wstring dateFormat;
         case domainTypeInteger : long totalDigits;
         case domainTypeReal : CG_RealTypeDef realTypeDef;
         case domainTypeSpatial : long dimensions;
      };

   // definition of valid values (range, value list) of the type
   union CG_DomainValueDef
      switch(CG_DomainValuesType)
      {
         case valueRef : StringSeq valueList;
         case valueRange : CG_ValueRangeSeq rangeList;
      };

   // definition of domain values (typical value, valid values, value
name, value description)
   // of the type
   struct CG_DomainValue
   {
      wstring value;
      CG_DomainValueDef valuesDef;
      wstring title;
      wstring description;
      wstring metadataURL;
   };
   typedef sequence<CG_DomainValue> CG_DomainValueSeq;

   // Description of domain of one requested metadata property or request
parameter
   struct CG_Domain
   {
      wstring attributeName;
      CG_DomainDef domainDef;
      CG_DomainValueSeq domainValue;
   };
   typedef sequence<CG_Domain> CG_DomainSeq;

   //-------------------------------------------------------------------
   // Messages
   //-------------------------------------------------------------------
   struct CG_Message
   {
      long long sessionID;
      wstring destinationID;
      CG_RequestID requestID;
      wstring additionalInfo;
   };
   struct CG_InitSessionRequest
   {
      CG_Message base;
   };
   struct CG_InitSessionResponse
   {
      CG_Message base;
   };
   struct CG_TerminateRequest
   {
      CG_Message base;
   };
   struct CG_TerminateResponse
   {
      CG_Message base;
      CG_Status status;
   };
   typedef sequence<CG_CapabilityType> CG_CapabilityTypeSeq;
   struct CG_ExplainServerRequest
   {
      CG_Message base;
      CG_CapabilityTypeSeq capabilities;
   };
   struct CG_ExplainServerResponse
   {
      CG_Message base;
      CG_CapabilityTypeSeq capabilities;
   };
   struct CG_StatusRequest
   {
      CG_Message base;
      CG_RequestID requestIDtoStatus;
   };
   struct CG_StatusResponse
   {
      CG_Message base;
      CG_RequestID requestIDtoStatus;
      CG_Status status;
   };
   struct CG_CancelRequest
   {
      CG_Message base;
      CG_RequestID requestIDtoCancel;
      boolean freeResources;
   };
   struct CG_CancelResponse
   {
      CG_Message base;
      CG_Status status;
      CG_RequestID canceledRequest;
   };
   typedef sequence<CG_SortField> CG_SortFieldSeq;
   struct CG_QueryRequest
   {
      CG_Message base;
      CG_QueryExpression queryExpression;
      CG_ResultType resultType;
      long iteratorSize;
      long cursor;
      CG_MessageFormat returnFormat;
      CG_PresentationDescription presentation;
      CG_SortFieldSeq sortField;
      CG_QueryScope queryScope;
      CG_CollectionName collectionID;
      CG_CatalogEntryType catalogType;
      boolean asynchronous;
      long maxLevel;
   };
   struct CG_QueryResponse
   {
      CG_Message base;
      CG_ReturnData retrievedData;
      CG_CollectionName resultSetID;
      CG_Status status;
      long hits;
      long cursor;
   };
   struct CG_PresentRequest
   {
      CG_Message base;
      CG_CollectionName resultSetID;
      CG_PresentationDescription presentation;
      CG_SortFieldSeq sortField;
      CG_MessageFormat returnFormat;
      long iteratorSize;
      long cursor;
   };
   struct CG_PresentResponse
```

```
{
   CG_Message base;
   CG_ReturnData retrievedData;
   long cursor;
   long hits;
   CG_Status status;
};
struct CG_ExplainCollectionRequest
{
   CG_Message base;
   CG_AttributeCategory attributeCategory;
   CG_CollectionName collectionID;
   CG_MessageFormat returnFormat;
};
struct CG_ExplainCollectionResponse
{
   CG_Message base;
   CG_CollectionName collectionID;
   CG_SchemaID dataModel;
   CG_Status status;
};

struct CG_GetDomainRequest
{
   CG_Message base;
   StringSeq attributes;
};

struct CG_GetDomainResponse
{
   CG_Message base;
   CG_DomainSeq attributeDomains;
};

// Messages for access
struct CG_BrokeredAccessRequest
{
   CG_Message base;
   wstring productHandle;
   CG_OrderSpecification orderInformation;
   wstring orderID;
   CG_BrokeredAccessRequestType requestType;
   CG_UserInformation userInformation;
   CG_StatusUpdateType statusOrderUpdateType;
   CG_PackageSpecification packageSpecification;
};

typedef sequence<long> LongSeq;
struct CG_BrokeredAccessResponse
{
   CG_Message base;
   CG_OrderStatus orderStatus;
   LongSeq resourceEstimate;
   CG_CollectionName order;
   wstring orderID;
   CG_Status status;
   CG_BrokeredAccessRequestType requestType;
   };

   typedef sequence<DynamicAny::NameValuePairSeq> NameValuePairSeqSeq;

   // Messages for managing functions

   // Insert-Structure for inserting metadata (payload) to a collection
   // of a defined catalogType in a predefined format (encoding)
   struct CG_InsertMetadata
   {
      CG_CollectionName collectionID;
      CG_CatalogEntryType catalogType;

      CG_MessageFormat encoding;

      // the data to insert, e.g. a list of NV-Pairs or ISO19139-XML
      any payload;
   };

   // Update-Structure for updating a certain set (defined by the
queryExpression) of metadata
   // (payload) of a collection/catalogType in a predefined format
(encoding)
   struct CG_UpdateMetadata
   {
      CG_CollectionName collectionID;
      CG_CatalogEntryType catalogType;

      CG_MessageFormat encoding;

      CG_QueryExpression queryExpression;

      // the data to replace, e.g. a list of NV-Pairs or ISO19139-XML
      any payload;
   };

   // Delete-Structure for deleting a certain set (defined by the
queryExpression) of metadata
   // of a collection/catalogType
   struct CG_DeleteMetadata
   {
      CG_CollectionName collectionID;
      CG_CatalogEntryType catalogType;

      CG_QueryExpression queryExpression;
   };

   // Enumeration for the type of a single manipulation in a transaction
   enum CG_MetadataManipulationType
      { ctInsert, ctUpdate, ctDelete };

   // Structure for the content of a single manipulation in a transaction
   union CG_MetadataManipulation
      switch(CG_MetadataManipulationType)
      {
         case ctInsert : CG_InsertMetadata insertData;
         case ctUpdate : CG_UpdateMetadata updateData;
         case ctDelete : CG_DeleteMetadata deleteData;
      };
   typedef sequence<CG_MetadataManipulation> MetadataManipulationSeq;

   // Struct of a transaction, as a sequence of single manipulations
   struct CG_Transaction
   {
      MetadataManipulationSeq manipulations;
   };

   struct CG_TransactionRequest
   {
      CG_Message base;
      CG_Transaction transactionData;
   };
   struct CG_TransactionResponse
   {
      CG_Message base;
      CG_Status status;

      // value of -1 means not calculated
      long totalInserted;
      long totalUpdated;
      long totalDeleted;

      // List of newly generated catalogue entry identifiers
      // assigned to the new catalogue entry instances
      NameValuePairSeqSeq newKeyList;
   };

   //-------------------------------------------------------------------
   // Exceptions
   //-------------------------------------------------------------------
   exception InvalidSession{};
   exception InvalidRequest{};
   exception InvalidCollection{ wstring diagnostic; };
   exception InvalidQuery{ wstring diagnostic; };
   exception NotImplemented{ wstring diagnostic; };
   exception NotSupported{ wstring diagnostic; };
   exception CatalogError{ wstring diagnostic; };

   //-------------------------------------------------------------------
   // Interfaces
   //-------------------------------------------------------------------
   interface OGC_Observer;
   interface OGC_Subject
   {
      oneway void attachObserver(in OGC_Observer Observer);
      oneway void detachObserver(in OGC_Observer Observer);
      oneway void notifyObserver();
   };
   interface OGC_Observer
   {
      void updateSubject(in OGC_Subject ChangedSubject);
   };

   interface OGC_Service : OGC_Subject
```

```
   {
      CG_ExplainServerResponse explainServer(in CG_ExplainServerRequest 
request)
            raises(CatalogError);
   };

   interface OGC_StatefulService : OGC_Service
   {
      CG_InitSessionResponse initSession(in CG_InitSessionRequest request)
            raises(CatalogError);
      CG_TerminateResponse terminateSession(in CG_TerminateRequest 
request)
            raises(InvalidSession, CatalogError);
      CG_StatusResponse status(in CG_StatusRequest request)
            raises(InvalidSession, InvalidRequest, CatalogError);
      CG_CancelResponse cancel(in CG_CancelRequest request)
            raises(InvalidSession, InvalidRequest, CatalogError);
   };

   interface CG_Discovery
   {
      CG_QueryResponse query(in CG_QueryRequest request)
            raises(InvalidSession, InvalidQuery, InvalidCollection, 
NotSupported, CatalogError);
      CG_PresentResponse present(in CG_PresentRequest request)
            raises(InvalidSession, InvalidCollection, NotSupported, 
CatalogError);
      CG_ExplainCollectionResponse explainCollection(in 
CG_ExplainCollectionRequest request)
            raises(CatalogError);
      CG_GetDomainResponse getDomain(in CG_GetDomainRequest request)
            raises(CatalogError);
   };
   interface CG_Access
   {
      // Direct access is provided by the IOR fields in the meta-
information
      // itself
      // Brokered access
      CG_BrokeredAccessResponse
            brokeredAccess(in CG_BrokeredAccessRequest request)
               raises(NotImplemented, CatalogError);
   };
   interface CG_Manager
   {
      // This operation is used to by a client that has the appropriate 
user
      // privileges to execute a whole set of insert, update and delete 
operation
      // of metadata to a catalog.
      CG_TransactionResponse
            transaction(in CG_TransactionRequest request)
               raises(NotImplemented, CatalogError);
   };

   interface CG_CatalogServices : OGC_StatefulService, CG_Discovery, 
CG_Access, CG_Manager
```

```
    {
    };
};
```

10 HTTP protocol binding (Catalogue Services for the Web, CSW)

10.1 Architectural principles

The purpose of this clause is to describe the request and response messages that are common to all web-based catalogue services. The basic message exchange pattern is illustrated in Figure 28.

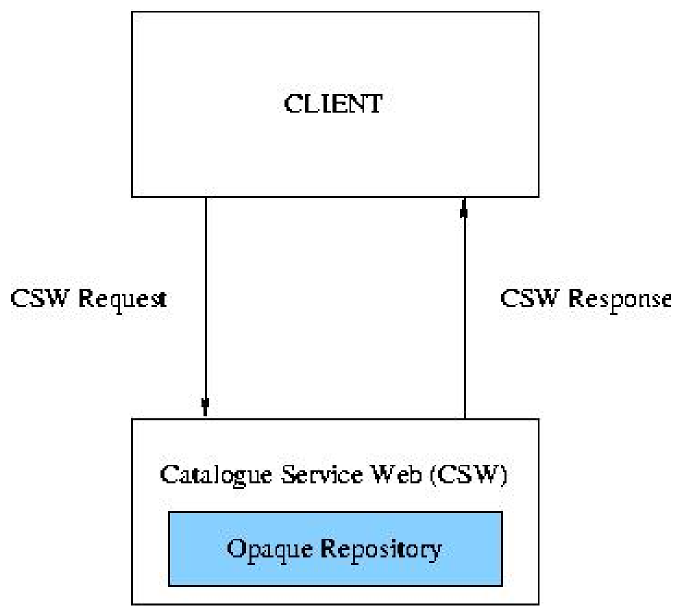

Figure 28 — Catalogue service web

The interaction between a client and a server is accomplished using a standard request-response model of the HTTP protocol. That is, a client sends a request to a server using HTTP, and expects to receive a response to the request or an exception message.

Request and response messages are encoded as keyword-value pairs within a request URI or using an XML entity-body. Requests may also be embedded in a messaging framework such as SOAP.

10.2 The HTTP protocol

10.2.1 Overview

The Hypertext Transfer Protocol (HTTP) is a generic, stateless, application-level protocol that is widely used to exchange information on the web. The HTTP/1.1 specification is published by the Internet Engineering Task Force (IETF) as RFC 2616: http://www.ietf.org/rfc/rfc2616. The "http" URI scheme is used to locate network resources using the HTTP protocol; consult Section 3.2 of RFC 2616 and RFC 2396 for details. The general syntax of the scheme is summarised below for convenience:

```
   http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

HTTP messages have a simple line-oriented structure. The three basic parts of a message are summarised in Table 51 (RFC 2616, 4.1):

Table 51 — HTTP message elements

Start line	Indicates what to do for a request or what happened for a response.
Header fields	Zero or more header fields, each consisting of a name and a value separated by a colon (:). Four categories of headers provide metainformation about the message: general, request, response, and entity. The headers part ends with a blank line.
Body	An optional body containing the message content that conforms to some Internet media type.

Note that URIs are generally case-sensitive except for the scheme and host names; furthermore, if the port number is not specified it is equivalent to the default TCP port number (80). As noted in Section 3 of RFC 2396, certain special characters are reserved within various URI components; if the data within a URI component conflicts with the reserved purpose it must be escaped before forming the URI.

This subclause clarifies some aspects of HTTP usage for catalogue application profiles that employ the protocol to exchange request and response messages. In many cases this means turning a SHOULD level requirement into a SHALL level requirement; what results is effectively an HTTP profile for OGC catalogue services.

10.2.2 Message headers

The standard headers are defined in Section 14 of RFC 2616. Some of these are of particular significance to catalogue operations.

Any HTTP/1.1 message containing an entity-body shall include a Content-Type header field defining the media type of that body (RFC 2616, 7.2.1); the charset parameter shall also be specified for text.

EXAMPLES 1

```
   Content-Type: application/xml; charset=utf-8
   Content-Type: application/octet-stream
   Content-Type: multipart/related; boundary="part-boundary";
      start="<urn:uuid:e3fec7a9-cc5d-45ba-87a5-8a2a27f6fb5b>"
      type="application/xml"
```

A user agent may use the Accept request header to declare a set of preferred Internet media types for the response. The IANA registry of media types is available online: http://www.iana.org/assignments/media-types/.

EXAMPLES 2

```
   Accept: application/xml
   Accept: application/xhtml+xml, text/html; q=0.5
```

10.2.3 Content encoding

The Content-Encoding entity-header may be used to indicate any additional content encodings that have been applied to the entity body, usually for the purpose of data compression or encryption. This header shall be included if a non-identity encoding has been applied.

EXAMPLE 1

```
Content-Encoding: gzip
```

A user agent may specify a preferred content encoding using the Accept-Encoding header. If no such request-header is included, the server shall use the "identity" encoding.

EXAMPLE 2

```
Accept-Encoding: gzip;q=1.0, identity; q=0.5
```

10.2.4 Request methods

The HTTP/1.1 specification defines eight methods for manipulating and retrieving representations of resources. Within an application profile abstract catalogue operations shall be mapped to one or more of these methods; these mappings should be consistent with HTTP/1.1 semantics. The methods that are most relevant to catalogue services are summarised in Table 52 below:

Table 52 — Selected HTTP Request Methods

Method Name	Semantics
GET	Used to retrieve whatever information (in the form of an entity) is identified by the Request-URI
POST	Used to request that the origin server accept the entity enclosed in the request as data to be processed by the resource identified by the Request-URI in the Request-Line

The Request-URI is a Uniform Resource Identifier that identifies the target resource to which the request is applied; it has the following syntax:

```
Request-URI = "*" | absoluteURI | abs_path [ "?" query ] | authority
```

10.2.5 Message payload

An application profile shall specify allowable payloads that constitute the body of a request or response message (if applicable). Entities shall conform to a registered Internet media type, but there is otherwise no restriction on the content; the actual payload is dependent upon the information model supported by a profile. Representations of catalogue entries must be defined such that they may substitute for the element **csw:AbstractRecord** defined in the record.xsd schema.

The common CSW record syntax is an XML-based encoding of Dublin Core metadata terms; it encompasses the core metadata properties specified in Subclause 6.3.2. The XML encoding of those properties is defined by the following XML-Schema fragment:

```
<xsd:element name="Record"
         type="csw:RecordType"
         substitutionGroup="csw:AbstractRecord"/>
<xsd:complexType name="RecordType">
```

```
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
         This type extends DCMIRecordType to add the gml:boundedBy spatial
         property, the value of which is a bounding envelope, expressed
         using GML 3 syntax (ISO 19136).
         </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
         <xsd:extension base="csw:DCMIRecordType">
         <xsd:sequence>
            <xsd:element ref="gml:boundedBy" minOccurs="0"/>
         </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
```

Brief, summary, and full element sets for the core queryables are defined in the records.xsd XML Schema referenced in Subclause 10.13.

10.3 Operation request and response encoding

10.3.1 Introduction

Only the GET and POST methods are employed in the HTTP binding. Table 53 summarizes the allowed HTTP method bindings and request data encodings for all CSW requests; optional method bindings and data encodings are enclosed in parentheses.

Table 53 — HTTP method bindings

Request	HTTP method binding(s)	Data encoding(s)[a,b]
GetCapabilities	GET (POST)	KVP (XML)
DescribeRecord	POST (GET)	XML (KVP)
GetDomain	POST (GET)	XML (KVP)
GetRecords	POST (GET)	XML (KVP)
GetRecordById	GET (POST)	KVP (XML)
Harvest	POST	XML (KVP)
Transaction	POST	XML

[a] XML = application/xml using POST (with a charset parameter if necessary—UTF-8 is strongly recommended)

[b] KVP = URL-encoded key/value pairs using GET or application/x-www-form-urlencoded using POST

Operation	**HTTP method binding(s)**	**Data encoding(s)**[a,b]
GetCapabilities	GET (POST)	KVP (XML)
DescribeRecord	POST (GET)	XML (KVP)
GetDomain	POST (GET)	XML (KVP)
GetRecords	POST (GET)	XML (KVP)
GetRecordById	GET (POST)	KVP (XML)
Harvest	POST	XML (KVP)
Transaction	POST	XML

[a] XML = application/xml using POST (with a charset parameter if necessary: UTF-8 is strongly recommended)

[b] KVP = URL-encoded key/value pairs using GET or application/x-www-form-urlencoded using POST

10.3.2 Simple object access protocol (SOAP)

This subclause specifies the use of SOAP messages for communication between a catalogue client and a CSW.

The Simple Object Access Protocol (SOAP) is a communication protocol for communication between applications. It defines a format for sending messages between communicating applications via the Internet and specifically using HTTP. Soap is platform independent, language independent and SOAP messages are encoded using XML. This means that SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

A SOAP message is an ordinary XML document containing the following elements:

a) A required Envelope element that identifies the XML document as a SOAP message

b) An optional Header element that contains header information

c) A required Body element that contains call and response information

d) An optional Fault element that provides information about errors that occurred while processing the message

All the elements above are declared in the default namespace for the SOAP envelope:

```
http://www.w3.org/2003/05/soap-envelope
```

and the default namespace for SOAP encoding and data types is:

```
http://www.w3.org/2003/05/soap-encoding
```

The SOAP specification defines a number of syntax rules. Among the most important are:

a) A SOAP message shall be encoded using XML

b) A SOAP message shall use the SOAP Envelope namespace

c) A SOAP message shall use the SOAP Encoding namespace

d) A SOAP message shall not contain a DTD reference

e) A SOAP message shall not contain XML Processing Instructions

The following XML fragment illustrates a skeleton SOAP message:

```
<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap=http://www.w3.org/2003/05/soap-envelope
   soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

   <soap:Header>
      ...
      ...
   </soap:Header>

   <soap:Body>
      ...
      ...
      <soap:Fault>
         ...
         ...
      </soap:Fault>
   </soap:Body>

</soap:Envelope>
```

A client may send CSW requests to a compatible catalogue using the body of a SOAP envelope. The client simply encodes the CSW request as the content of the <soap:Body> element in the request message.

The CSW may then response by generating a SOAP message where the response to the client's request is the content of the **<soap:Body>** element.

In the event the an exception is encountered while processing a CSW request encoded in a SOAP envelope, the CSW server must generate a SOAP response message where the content of the **<soap:Body>** element is a **<soap:Fault>** element. The following skeleton XML fragment must be used when generating the **<soap:Body>** element in the event that the CSW server encounters an exception:

```
<soap:Body>
   <soap:Fault>
      <soap:faultcode>soap:Server</soap:faultode>
      <soap:faultstring>A server exception was
encountered.</soap:faultstring>
      <soap:faultactor>http://www.url_of_CSW_server.com/</soap:faultactor>
      <soap:detail>
       <ows:ExceptionReport>
       …
       </ows:ExpetionReport>
      </soap:detail>
   </soap:Fault>
</soap:Body>
```

The **<soap:faultcode>** element must have the content *soap:Server* indicating that this is a server exception. The **<soap:faultstring>** element must have the content “Server exception was encountered.”. This fixed string is used since the details of the exception will be specified in the **<soap:detail>** element using an **<ows:ExceptionRecport>** element as defined in document [OGC 04-016r2].

The **<soap:detail>** element must contain an **<ows:ExceptionReport>** element detailing the specific exception that the server encountered.

The use of the **<soap:Header>** element is not discussed in this version of this specification.

10.3.3 Namespaces

Namespaces [W3C Recommendation January 1999] are used to discriminate XML vocabularies from one another. For the CSW there are two normative namespace definitions, namely:

```
(http://www.opengis.net/cat/csw) - for the CSW interface vocabulary
(http://www.opengis.net/ogc)     - for the OGC Filter vocabulary
```

A given CSW implementation will make use of one or more XML Schemas describing the metadata that is being manipulated and these schemas will, in turn, use one or more namespaces (e.g. http://www.someserver.com/myns).

10.3.4 Predicate languages

The general model allows catalogue clients to specify the predicate language used to constrain operations. The HTTP protocol binding schemas define two predicate languages, based on the BNF in Subclause 6.2.2, that may be used. The two predicate languages are:

a) **CQL_TEXT** is a text encoding of the BNF.

b) **FILTER** is an XML encoding of the BNF grammar and is normatively defined in the *Filter Encoding Implementation Specification*, version 1.1.0 [OGC 04-095]. All CSW implementations are required to support this filter syntax.

Table 54 defines the parameters required to specify a predicate in keyword-value pair encoded CSW operation requests.

Table 54 — KVP encoding for constraints

Keyword [b]	Description	Data type and value	Optionality
CONSTRAINT_LANGUAGE	Identifies the predicate language used for the value of the Constraint	Code List with allowed values: *CQL_TEXT,* used to indicate CQL. *FILTER,* used to indicate OGC Filter.	Zero or one (Optional) [a] Must be specified with the Constraint
CONSTRAINT_LANGUAGE_VERSION	Identifies the version of the predicate language used.	Character String	Zero or one There is no default as the parameter is specified if required to indicate which version of a specification the value of the constraint parameter conforms to.
Constraint	Text of query constraint in the predicate language identified by the CONSTRAINT _LANGUAGE	Character String	Zero or one (Optional) Must be specified with the CONSTRAINT _LANGUAGE

a The CONSTRAINT_LANGUAGE parameter contains the same information as the contents of the <Constraint> element in XML encoding.

b Parameter keywords, for KVP encoding, are case insensitive.

The following XML schema fragments define how the predicate language may be XML encoded in CSW operations that allow constraints to be defined (Query, Update and Delete):

```
<xsd:complexType name="QueryConstraintType" id="QueryConstraintType">
  <xsd:choice>
    <xsd:element ref="ogc:Filter"/>
    <xsd:element name="CqlText" type="xsd:string"/>
  </xsd:choice>
    <xsd:attribute name="version" type="xsd:string" use="required">
  </xsd:attribute>
</xsd:complexType>
```

The **version** parameter may be used to specify a version number indicating which version of a specification the constraint conforms to. For example, in the XML encoding, if the **<ogc:Filter>** element is being used, the **version** parameter could be set to “1.0.0” indicating that the filter conforms to version 1.0.0 of the Filter Encoding Implementation Specification [OGC 02-059].

10.3.5 General model message mapping

Table 55 maps the general model operations, defined in Clause 7, to the Catalogue Service for the Web (CSW) operations. This table does not list the general model operations that are not mapped to CSW operations.

Table 55 — General model to CSW mapping

General Model Operation	CSW Operation
OGC_Service.getCapabilities	OGC_Service.GetCapabilities
Discovery.query	CSW-Discovery.GetRecords
Discovery.present	CSW-Discovery.GetRecordById
Discovery.describeRecordType	CSW-Discovery.DescribeRecord
Discovery.getDomain	CSW-Discovery.GetDomain
Manager.transaction	CSW-Publication.Transaction
Manager.hervestRecords	CSW-Publication.Harvest

10.3.6 Common request parameters

All CSW operation requests except for GetCapabilities shall include the three parameters specified in Table 20 of [04-016r2]. Only one of these parameters is included in the general catalogue model, the others are specific to the HTTP protocol binding.

In KVP encoding, these common parameters in CSW operation requests are encoded as shown in Table 56. Note that the parameter names in all KVP encodings must be handled in a **case insensitive** manner.

Table 56 — KVP encoding of common operation request parameters

Keyword	Datatype and value	Optionality	Parameter in general model
REQUEST	Character String type Value is operation name (e.g., "DescribeRecord")	One (Mandatory)	(none)
service	Character String type Fixed values of "CSW"	One (Mandatory)	serviceId
version	Character String type Fixed value of "2.0.0"	One (Mandatory)	(none)

In XML encoding, all operation request elements, except for GetCapabilities, are extended from the following XML Schema fragment:

```
<xsd:complexType name="RequestBaseType">
   <xsd:complexContent>
      <xsd:extension base="ows:RequestBaseType">
         <xsd:attribute name="service" type="xsd:string"
                        use="optional" default="CSW"/>
         <xsd:attribute name="version" type="xsd:string"
                        use="required" fixed="2.0.0"/>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

The “service” parameter is used to indicate that the request is a CSW request. This parameter must be specified for all CSW requests. The “version” parameter is used to indicate that the associated CSW request conforms to this specification. This is indicated by setting the value of the **version** parameter to 2.0.0. This XML Schema fragment does not include a “request” attribute, since the name of the operation requested is always the name of the XML element encoding the request.

10.4 Operations overview

Figure 29 shows the request/response message pairs for all the operations defined for the web catalogue service (CSW). There are three classes of operations: *service operations* which are operations a client may use to interograte the service to determine its capabilities; *discovery operations* which a client may use to determine the information model of the catalogue and query catalogue records; and *management operations* which are used to create or change records in the catalogue.

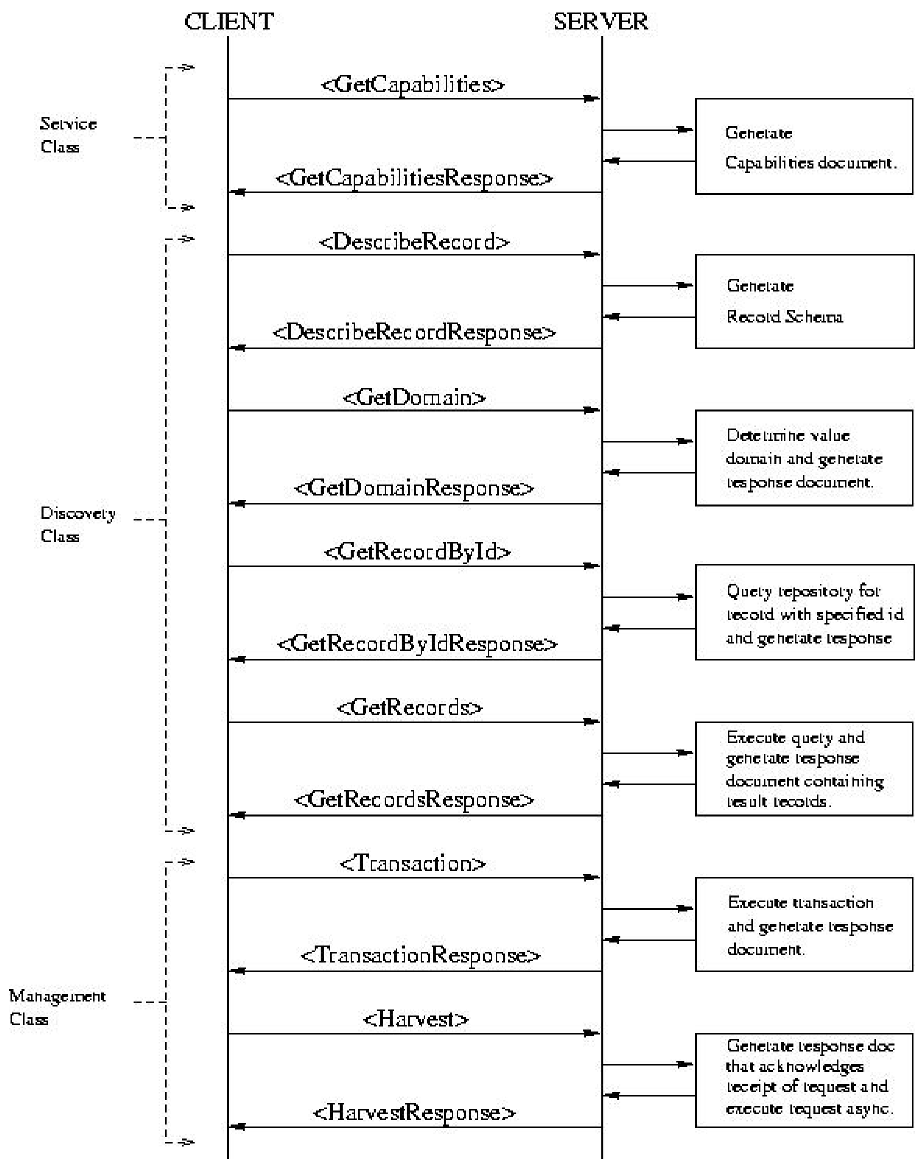
CLIENT
SERVER
Service Class
Discovery Class
Management Class
<GetCapabilities>
Generate Capabilities document.
<GetCapabilitiesResponse>
<DescribeRecord>
Generate Record Schema
<DescribeRecordResponse>
<GetDomain>
Determine value domain and generate response document.
<GetDomainResponse>
<GetRecordById>
Query repository for record with specified id and generate response
<GetRecordByIdResponse>
<GetRecords>
Execute query and generate response document containing result records.
<GetRecordsResponse>
<Transaction>
Execute transaction and generate response document.
<TransactionResponse>
<Harvest>
Generate response doc that acknowledges receipt of request and execute request async.
<HarvestResponse>

Figure 29 — Protocol sequence diagram

Figure 30 depicts a conceptual architecture to illustrate the relationship of these interfaces to service consumers and producers. The arrows represent the CSW requests that producers and consumers of metadata may generate. For example, to create metadata, a metadata producer may invoke the **Transaction** request or the **Harvest** request. Similarly, a consumer of metadata may invoke the **GetRecords** request to query the catalogue.

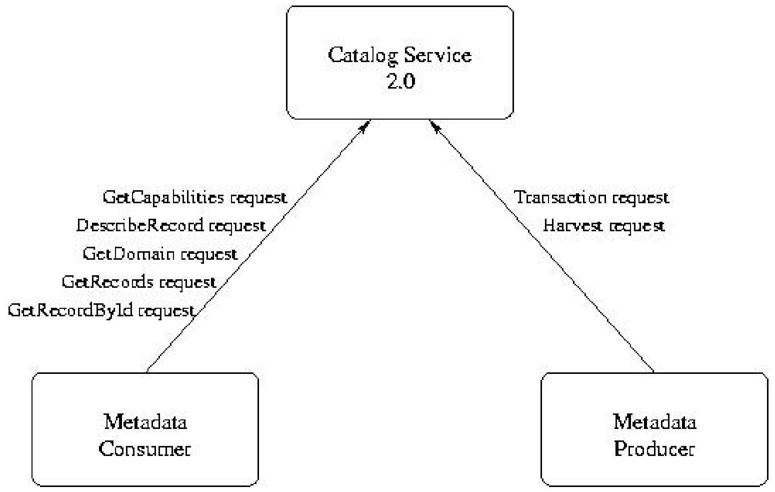

Figure 30 — Conceptual architecture

10.5 GetCapabilities operation

10.5.1 Introduction

The mandatory GetCapabilities operation allows CSW clients to retrieve service metadata from a server. The response to a GetCapabilities request shall be an XML document containing service metadata about the server. This subclause specifies the XML document that a CSW server must return to describe its capabilities.

10.5.2 Operation request

The GetCapabilities operation request is defined in Subclause 7.2 of the *OGC Web Services Common Specification 1.0* [OGC 05-008]. CSW servers shall implement the *request*, *service*, *Sections*, *AcceptVersions* and *AcceptFormats* operation request parameters, and may implement the *updateSequence* parameter. If the updateSequence parameter is not implemented, a catalogue server shall always return the most up-to-date version of the capabilities document. All CSW servers shall implement the HTTP GET transfer using the keyword-value pair encoding of the GetCapabilities operation. Servers may optionally implement the HTTP POST transfer using XML encoding only.

The value of the *service* parameter shall be "CSW" or the identifier of a specific CSW Application Profile as specified in that profile. The common service metadata elements that may be included in a Capabilities document are specified in Subclause 7.4 of OGC 05-008; a catalogue service that implements the CSW binding may also include the elements listed in Table 57. An application profile may introduce additional service information items as needed by extending the csw:CapabilitiesType definition.

Table 57 — Additional section name value and meaning

Section name	Meaning
Filter_Capabilities	A Filter_Capabilities section must be included in the service metadata to describe which elements of the predicate language are supported. All CSW implementations must support at least the following filter operators: • logical operators: And, Or, Not • comparison operators: PropertyIsEqualTo, PropertyIsNotEqualTo, PropertyIsLessThan, PropertyIsGreaterThan, PropertyIsLessThanOrEqualTo, PropertyIsGreaterThanOrEqualTo, PropertyIsLike • spatial operators: BBOX.

10.5.3 Operation response

The service metadata document shall contain the sections specified in Table 58. Depending on the values in the *Sections* parameter of the GetCapabilities operation request, any combination of these sections can be requested and returned. If the Sections parameter is not specified, then all sections shall be returned.

Table 58 — Section names and contents

Section name	Contents
ServiceIdentification	Metadata about a specified CSW implementation. The contents and schema of this section shall be as specified in Subclause 7.4.3 and owsServiceIdentification.xsd of [OGC 04-016r2].
ServiceProvider	Metadata about the organization offering the CSW service. The contents and schema of this section shall be as specified in Subclauses 7.4.4 and owsServiceProvide.xsd of [OGC 04-016r2].
OperationsMetadata	Metadata about the CSW operations offered by a specific CSW implementation, including the URLs for operation requests. The contents and schema of this section shall be as specified in Subclauses 7.4.5 and owsOperationsMetadata.xsd of [OGC 04-016r2]. The specific operations that may be listed in the OperationsMetadata section are specified in subclause 10.5.4 or this document.
Filter_Capabilities	Metadata about the filter capabilities of the server if the server implements the Filter predicate encoding as defined in [OGC 02-059]..

10.5.4 OperationsMetadata section standard contents

The OperationsMetadata element shall list all operations implemented by the service, as described in Subclause 7.4.5 of OGC 05-008. An application profile may restrict the <ExtendedCapabilities> element to provide additional computational metadata (e.g., WSDL service descriptions, OWL-S resource definitions). liant with this specification. Table 60 lists the optional values of OperationsMetadata section attributes for additional operations that a CSW may offer. If a specified server implementation offers one or more of these operations, they shall be listed in the OperationsMetadata section of the capabilities document. In both tables, the "Attribute name" column uses dot-separator notation to specify parts of a parent item. The "Attribute value" column references an operation parameter, and the meaning of including that value is listed in the right column.

Table 59

Table 59 specifies the required values of OperationsMetadata section attributes for operations that a CSW server shall implement to be minimally compliant with this specification. Table 60 lists the optional values of OperationsMetadata section attributes for additional operations that a CSW may offer. If a specified server implementation offers one or more of these operations, they shall be listed in the OperationsMetadata section of the capabilities document. In both tables, the "Attribute name" column uses dot-separator notation to specify parts of a parent item. The "Attribute value" column references an operation parameter, and the meaning of including that value is listed in the right column.

Table 59 — Required values of the OperationsMetadata section attributes

Attribute name	Attribute value	Meaning of attribute value
OperationsMetadata.Operation.name	GetCapabilities	The GetCapabilities operation is implemented by this server.
OperationsMetadata.Operation.name	DescribeRecord	The DescribeRecord operation is implemented by this server.
OperationsMetadata.Operation.name	GetRecords	The GetRecords operation is implemented by this server.

Table 60 — Optional values of the OperationsMetadata section attributes

Attribute name	Attribute value	Meaning of attribute value
OperationsMetadata.Operation.name	GetRecordById	The GetRecordById operation is implemented by this server.
OperationsMetadata.Operation.name	GetDomain	The GetDomain operation is implemented by this server.
OperationsMetadata.Operation.name	Harvest	The Harvest operation is implemented by this server.
OperationsMetadata.Operation.name	Transaction	The Transaction operation is implemented by this server.

In addition to the items listed in Table 60, there are many optional values of "name" attributes and "value" elements in the OperationsMetadata section, primarily for recording the domain of various parameters and quantities. For example, the domain of the exceptionCode parameter could record all the codes implemented for each operation by that specific server. Similarly, each of the GetCapabilities operation request parameters might have its domain recorded. For example, the domain of the Sections parameter could record all the sections implemented by that specific server.

10.5.5 Examples[12]

KVP Encoding:

```
   http://www.someserver.com/wrs.cgi?REQUEST=GetCapabilities&SERVICE=CSW&A
CCEPTVERSION=2.0.0,0.7.2&outputFormat=application/xml
```

XML Encoding:

[12] All examples in clause 10 are informative. In addition, the examples do not include all the XML syntax required to validate. This is done intentionally so as not to obfuscate the examples with XML syntax.

```
<GetCapabilities service="CSW">
   <AcceptVersions>
      <Version>2.0.0</Version>
      <Version>0.7.2</Version>
   </AcceptVersions>
   <AcceptFormats>
      <OutputFormat>application/xml</OutputFormat>
   </AcceptFormats>
</GetCapabilities>
```

Example response:

Annex D includes a sample capabilities document for a CSW server that supports only the default message payload. As described in clause 10.2.5, the default message payload is the core queryable elements defined by the **csw:Record** element; this default may be overridden in an application profile. The **csw:Record** element is the root element of the XML encoding of the core queryable element and is define in the schema file record.xsd.

10.6 DescribeRecord operation

10.6.1 Introduction

The mandatory **DescribeRecord** operation allows a client to discover elements of the information model supported by the target catalogue service. The operation allows some or all of the information model to be described.

10.6.2 KVP encoding

Table 62 specifies the keyword-value pair (KVP) encoding for the DescribeRecord operation request. This encoding is suitable for the HTTP GET binding.

NOTE To reduce the need for readers to refer to other parts of this document, the first three parameters listed below are copied from Table 56 in Subclause 10.3.5 of this document.

Table 61 — KVP encoding for DescribeRecord operation request

Keyword [c]	Data type and value	Optionality and use	Parameter in general model
REQUEST	Character String Fixed value of *DescribeRecord*, case insensitive	One (Mandatory) [a]	(none)
service	Character String Fixed value of *CSW*	One (Mandatory)	serviceId
version	Character String Fixed value of *2.0.0*	One (Mandatory)	(none)
NAMESPACE	List of Character String, comma separated Used to specify namespace(s) and their prefix(es) Format is [prefix:]url. If prefix is not specified, then this is the default namespace.	One (Mandatory) [b] Include declarations for each namespace used in a TypeName	(none)
TypeName	List of Character String, comma separated One or more qualified type names to be described	Zero or one (Optional) Default action is to describe all types known to server	typeName
outputFormat	Character String A MIME type indicating the format that the output document should have	Zero or one (Optional) Default value is application/xml	returnFormat
schemaLanguage	Character String	Zero or one (Optional) Default value is XMLSCHEMA	schemaLanguage

a The REQUEST parameter contains the same information as the name of the < DescribeRecord> element in XML encoding.

b The NAMESPACE parameter contains the same information as the xmlns attributes which may be used to define and bind namespaces in XML encoding.

c Parameter keywords are case insensitive for KVP encoding.

10.6.3 XML encoding

10.6.3.1 Overview

The following XML-Schema fragment defines the XML encoding for the **DescribeRecord** operation request:

```
<xsd:element name="DescribeRecord" type="csw:DescribeRecordType"/>
<xsd:complexType name="DescribeRecordType">
   <xsd:complexContent>
      <xsd:extension base="csw:RequestBaseType">
         <xsd:sequence>
            <xsd:element name="TypeName" type="csw:TypeNameType"
                         minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="outputFormat" type="xsd:string"
```

```
                        use="optional"
                        default="application/xml"/>
            <xsd:attribute name="schemaLanguage" type="xsd:anyURI"
                        use="optional"
                        default="http://www.w3.org/XML/Schema"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="TypeNameType">
      <xsd:simpleContent>
         <xsd:extension base="xsd:NCName">
            <xsd:attribute name="targetNamespace"
                  type="xsd:anyURI"
                  use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
```

10.6.4 Parameter descriptions

10.6.4.1 NAMESPACE parameter

The **DescribeRecord** operation depends on namespace declarations in order to know exactly which types to describe.

For XML encoded **DescribeRecord** requests, the namespace declarations are specified using standard XML conventions (**xmlns** attributes) and described in the document "Namespaces in XML" [http://www.w3.org/TR/1999/REC-xml-names-19990114].

For the KVP encoding, namespace declarations are specified using the **NAMESPACE** parameter. The **NAMESPACE** parameter is a comma separated list of namespace declarations of the form *alias:namespace*.

The following is an example delcaration:

```
   ...NAMESPACE=gml:http://www.opengis.org/gml,wfs:http://www.opengis.org/
wfs...
```

The value of the **NAMESPACE** parameter must be properly escaped for url encoding, which is not shown in this example for the sake of clarity

10.6.4.2 TypeName parameter

The **TypeName** parameter specifies a list of type names that are to be described by the catalogue.

Every type name must be fully qualified in order to indicate the target namespace for the type definition. If no type names are provided, then entire schemas from the target namespace are returned. For XML-encoded **DescribeRecord** requests, the namespace declarations are specified using the targetNamespace attribute of the TypeName element.

If the **DescribeRecord** request is XML encoded, then namespaces must be declared according to the conventions of XML. If the **DescribeRecord** request is KVP encoded, then the namespaces referenced must be declared using the **NAMESPACE** parameter.

10.6.4.3 outputFormat parameter

The **outputFormat** parameter specifies the MIME type of the response document. The default output format attribute is the MIME type *application/xml*. All supported output formats must be declared in the Capabilities document.

10.6.4.4 schemaLanguage parameter

The **schemaLanguage** parameter is used to specify the schema language that should be used to describe the specified types. The default value is *XMLSCHEMA***,** which indicates that the XML-Schema, schema description language should be used. Other schemas lanaguages are possible as long as they are declared in the Capabilities document.

10.6.5 Response

The following XML Schema fragment defines the response to a **DescribeRecord** operation when the **schemaLanguage** parameter is set to the value *XMLSCHEMA*.

```
<xsd:element name="DescribeRecordResponse"
        type="csw:DescribeRecordResponseType"/>
<xsd:complexType name="DescribeRecordResponseType">
   <xsd:sequence>
      <xsd:element name="SchemaComponent"
               type="csw:SchemaComponentType"
               minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SchemaComponentType" mixed="true">
   <xsd:sequence>
      <xsd:any namespace="##any" processContents="lax"/>
   </xsd:sequence>
   <xsd:attribute name="targetNamespace" type="xsd:anyURI"
               use="required"/>
   <xsd:attribute name="parentSchema" type="xsd:anyURI"
               use="optional"/>
   <xsd:attribute name="schemaLanguage" type="xsd:anyURI"
               use="required"/>
</xsd:complexType>
```

The <**DescribeRecordResponse**> element is a container for zero or more <**SchemaComponent**> elements, each of which contains the description of one or more type names in the requested schema language. The <**SchemaComponent**> element may contain any content so that schema language descriptions other than XML Schema may be accommodated. For example, the content could be SQL DDL.

10.6.6 Examples

KVP encoded example

```
http://www.someserver.com/csw/csw.cgi?request=DescribeRecord&version=2.0.0&
outputFormat=application/xml&schemaLanguage=XMLSCHEMA&typeName=csw:Record&n
amespace=csw:http://www.opengis.org/cat/csw
```

XML encoded example

```
<csw:DescribeRecord version="2.0.1"
  outputFormat="application/xml"
  schemaLanguage="http://www.w3.org/2001/XMLSchema">

  <csw:TypeName
    targetNamespace="http://www.opengis.org/cat/csw">Record</csw:TypeName>

</csw:DescribeRecord>
```

10.7 GetDomain operation

10.7.1 Introduction

The optional **GetDomain** operation is used to obtain runtime information about the range of values of a metadata record element or request parameter. The runtime range of values for a property or request parameter is typically much smaller than the value space for that property or parameter based on its static type definition. For example, a property or request parameter defined as a 16bit positive integer in a database may have a value space of 65535 distinct integers but the actual number of distinct values existing in the database may be much smaller.

This type of runtime information about the range of values of a property or request parameter is useful for generating user interfaces with meaningful pick lists or for generating query predicates that have a higher chance of actually identifying a result set.

It should be noted that the **GetDomain** operation is a “best-effort” operation. That is to say that a catalogue tries to generate useful information about the specified request parameter or property if it can. It is entirely possible that a catalogue may not be able to determine anything about the values of a property or request parameter beyond the basic type; in this case only a type reference or a type description will be returned.

10.7.2 KVP encoding

Table 64 specifies the keyword-value pair encoding for the **GetDomain** operation request.

NOTE To reduce the need for readers to refer to other parts of this document, the first three parameters listed below are copied from Table 56 in Subclause 10.3.5 of this document.

Table 62 — KVP encoding for GetDomain operation request

Keyword [b]	Data type and value	Optionality and use	Parameter in general model
REQUEST	Character String Fixed value of "GetDomain", case insensitive	One (Mandatory) [a]	(none)
service	Character String Fixed values of "CSW"	One (Mandatory)	serviceId
version	Character String Fixed value of "2.0.0"	One (Mandatory)	(none)
ParameterName	List of Character String, comma separated Unordered list of names of requested parameters, of the form *OperationName.ParameterName*	Zero or one (Conditional) Include when PropertyName not included	parameterName
PropertyName	List of Character String, comma separated Unordered list of names of requested properties, from the information model that the catalogue is using	Zero or one (Conditional) Include when ParameterName not included	parameterName

a The REQUEST parameter contains the same information as the name of the <GetDomain> element in XML encoding.

b Parameter keywords are case insensitive for KVP encoding.

10.7.3 XML encoding

The following XML-Schema fragment defines that XML encoding for the **GetDomain** operation request:

```
<xsd:element name="GetDomain" type="csw:GetDomainType"/>
<xsd:complexType name="GetDomainType">
  <xsd:complexContent>
    <xsd:extension base="csw:RequestBaseType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="PropertyName" type="xsd:anyURI" />
          <xsd:element name="ParameterName" type="xsd:anyURI" />
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

10.7.4 Parameter descriptions

10.7.4.1 PropertyName parameter

The **PropertyName** parameter is used to specify the name of a property that is defined in the information model for which value domain information is desired. An example of a property name might be the name of one of the core queryable properties described in Subclause 6.3.2. The PropertyName value may be specified using an absolute or a relative URI; the precise syntax or permissible values are defined in an application profile.

10.7.4.2 ParameterName parameter

The **ParameterName** parameter is used to specify the name of an interface parameter for which value domain information is desired. Table 63 defines the list of interface parameters that may be interrogated using the **GetDomain** operation.

Table 63 — Interface parameters that may be interrogated using GetDomain operation

Parameter Name
GetRecords.resultType
GetRecords.outputFormat
GetRecords.outputRecType
GetRecords.typeName
GerRecords.ElementSetName
GetRecords.ElementName
GetRecords.CONSTRAINTLANGUAGE
GetRecordsById.ElementSetName
DescribeRecord.typeName
DescribeRecord.schemaLanguage

10.7.5 Response

The following XML-Schema fragment defines the response to a **GetDomain** operation.

```
<xsd:element name="GetDomainResponse" type="csw:GetDomainResponseType">
<xsd:complexType name="GetDomainResponseType">
   <xsd:sequence>
      <xsd:element name="DomainValues"
               type="csw:DomainValuesType"
               maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DomainValuesType">
   <xsd:sequence>
      <xsd:choice>
         <xsd:element name="PropertyName" type="xsd:QName"/>
         <xsd:element name="ParameterName" type="xsd:QName"/>
      </xsd:choice>
      <xsd:choice minOccurs="0">
         <xsd:element name="ListOfValues"
                type="csw:ListOfValuesType"/>
         <xsd:element name="ConceptualScheme"
                type="csw:ConceptualSchemeType" />
         <xsd:element name="RangeOfValues"
                type="csw:RangeOfValuesType" />
      </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="type" type="xsd:QName" use="required" />
   <xsd:attribute name="uom" type="xsd:anyURI" use="optional" />
</xsd:complexType>
```

```
   <xsd:complexType name="ListOfValuesType">
      <xsd:sequence>
         <xsd:element name="Value" type="xsd:anyType"
                  maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="ConceptualSchemeType">
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string"/>
         <xsd:element name="Document" type="xsd:anyURI"/>
         <xsd:element name="Authority" type="xsd:anyURI"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="RangeOfValuesType">
      <xsd:sequence>
         <xsd:element name="MinValue" type="xsd:anyType"/>
         <xsd:element name="MaxValue" type="xsd:anyType"/>
      </xsd:sequence>
   </xsd:complexType>
```

The response is composed of one or more <**DomainValues**> elements. The domain values may be a list of enumerated values (i.e. **<ListOfValues**>), one or more ranges of values (i.e. <**RangeOfValues>**), or a reference to some authoritative vocabulary (i.e. **<ConceptualScheme>**). An example of an authoritative vocabulary might be a standard list of animal and plant species names.

If the only child element of the <**DomainValue**> element is the <**PropertyName**> or <**ParameterName**> element, this shall be taken to mean that the catalogue was unable to determine anything about the specified property or parameter.

10.7.6 Examples

KVP encoded example:

```
http://www.someserver.com/csw/csw.cgi?request=GetDomain&version=2.0.0&para
meterName=GetRecords.outputFormat
```

XML encoded example:

```
<csw:GetDomain version="2.0.0">
   <csw:ParameterName>GetRecords.outputFormat</csw:ParameterName>
</csw:GetDomain>
```

10.8 GetRecords operation

10.8.1 Introduction

The primary means of resource discovery in the general model are the two operations **search** and **present**. In the HTTP protocol binding these are combined in the form of the mandatory **GetRecords** operation, which does a search and a piggybacked present.

10.8.2 KVP encoding

Table 68 specifies the keyword-value pair encoding for the **GetRecords** operation request. This encoding is suitable for the HTTP GET binding.

NOTE To reduce the need for readers to refer to other parts of this document, the first three parameters listed below are copied from Table 56 in Subclause 10.3.5 of this document.

Table 64 — KVP encoding for GetRecords operation request

Keyword [d]	**Data type and value**	**Optionality and use**	**Parameter in general model**
REQUEST	Character String Fixed value of *GetRecord*, case insensitive	One (Mandatory) [a]	(none)
service	Character String Fixed values of "CSW"	One (Mandatory)	serviceId
version	Character String Fixed value of *2.0.0*	One (Mandatory)	(none)
NAMESPACE	List of Character String, comma separated Used to specify a namespace and its prefix Format must be [<prefix>:]<url>. If the prefix is not specified then this is the default namespace.	Zero or one (Optional) [b] Include value for each distinct namespace used by all qualified names in the request. If not included, all qualified names are in default namespace	(none)
resultType	CodeList with allowed values: *"hits"*, *"results"* or *"validate"*	Zero or one (Optional) Default value is *"hits"*	resultType

Keyword [d]	Data type and value	Optionality and use	Parameter in general model
outputFormat	Character String Value is Mime type The only value that must be supported is *application/xml*. Other suppored values may include *text/html* and *text/plain*	Zero or one (Optional) Default value is *application/xml*	returnFormat
outputSchema	Defined in a profile.	Zero or one (Optional) Default value is *csw:Record*.	responseSchema
startPosition	Positive Integer	Zero or one (Optional) Default value is 1	cursorPosition
maxRecords	Positive Integer	Zero or one (Optional) Default values is 10	iteratorSize
typeNames	List of Character String, comma separated Unordered List of object types implicated in the query	One (Mandatory)	collectionID
ElementSetName OR ElementName	List of Character String	Zero or one (Optional) Default action is to present all metadata elements	responseElements
CONSTRAINTLANGUAGE	CodeList with allowed values: *CQL_TEXT* or *FILTER*	Zero or one (Optional) [c] Include when Constraint included	queryLanguage
Constraint	Character String Predicate expression specified in the language indicated by the CONSTRAINTLANGUAGE parameter	Zero or one (Optional) Default action is to execute an unconstrained query	predicate
SortBy	List of Character String, comma separated Ordered list of names of metadata elements to use for sorting the response Format of each list item is *metadata_elemen_ name:A* indicating an ascending sort or *metadata_ element_name:D* indicating descending sort	Zero or one (Optional) Default action is to present the records in the order in which they are retrieved	sortField and sortOrder
DistributedSearch	Boolean	Zero or one (Optional) Default value is FALSE	queryScope
hopCount	Integer	Zero or one (Optional) Include only if DeistributeSearch parameter is included Default value is *2*	queryScope

Keyword [d]	Data type and value	Optionality and use	Parameter in general model
ResponseHandler	URL	Zero or one (Optional) If not included, process request synchronously	responseHandler

a The REQUEST parameter contains the same information as the name of the < GetRecords> element in XML encoding.

b The NAMESPACE parameter contains the same information as the xmlns attributes which may be used for encoding namespace information in XML encoding.

c The CONSTRAINTLANGUAGE parameter contains the same information as the root element of the content of the <Constraint> element indicates the predicate language being used in XML encoding.

d Parameter keywords are case insensitive for KVP encoding.

10.8.3 XML encoding

The following XML-Schema fragment defines the XML encoding of the GetRecords operation request:

```
<xsd:element name="GetRecords" type="csw:GetRecordsType"
             id="GetRecords"/>
<xsd:complexType name="GetRecordsType" id="GetRecordsType">
  <xsd:complexContent>
    <xsd:extension base="csw:RequestBaseType">
      <xsd:sequence>
        <xsd:element name="DistributedSearch"
                     type="csw:DistributedSearchType"
                     minOccurs="0"/>
        <xsd:element name="ResponseHandler"
                     type="xsd:anyURI"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:choice>
          <xsd:element ref="csw:AbstractQuery"/>
          <xsd:any processContents="strict"
                   namespace="##other" />
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="requestId" type="xsd:anyURI"
                     use="optional" />
      <xsd:attribute name="resultType" type="csw:ResultType"
                     use="optional" default="hits"/>
      <xsd:attributeGroup ref="csw:BasicRetrievalOptions"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

10.8.4 Parameter descriptions

10.8.4.1 NAMESPACE parameter

The **NAMESPACE** parameter is included in the KVP encoding to allow clients to bind any namespace prefixes that might be used for qualified names specified in other parameters. For example, the **typeName** parameter may include qualified names of the form *namespace prefix:name*.

The value of the **NAMESPACE** parameter is a comma separated list of character strings of the form *[namespace prefix:]namespace url*. Not including the name namespace prefix binds the specified URL to the default namespace. As in XML, only one default namespace may be bound.

This parameter is not required for the XML encoding since XML includes a mechanism for binding namespace prefixes.

10.8.4.2 resultType parameter

The **resultType** parameter may have the value *"hits"*, *"results",* or *"validate"*; the value determines whether the catalogue service returns just a summary of the result set, includes one or more records from the result set, or validates the request message and processes it asynchronously.

If the **resultType** parameter is set to *"*hits*"*, the catalogue service shall return a <**GetRecordsResponse**> element containing an empty <SearchResults> element that indicates the estimated size of the result set. Optional attributes may or may not be set accordingly.

If the **resultType** parameter is set to "results", the catalogue service must include any matching records within the <SearchResults> element, up to the maximum number of records specified in the request.

If the **resultType** parameter is set to "validate", the catalogue service must validate the request and return an <Acknowledgement> message if validation succeeds; a <ServiceExceptionReport> is returned if validation fails. If the catalogue supports asynchronous query processing, the acknowledgement response must include a RequestId element that may be subsequently used to retrieve the result set when processing is complete.

10.8.4.3 outputFormat parameter

The **outputFormat** parameter is used to control the format of the output that is generated in response to a **GetRecords** request. Its value must be a MIME type. The default value, "**application/xml**", means that the output shall be an XML document. All registries shall at least support XML as an output format. Other output formats may be supported and may include output formats such as TEXT (MIME type *text/plain*), or HTML (MIME type *text/html*). The list of output formats that a CSW instance provides must be advertised in the Capabilities document.

In the case where the output format is *application/xml*, the CSW must generate an XML document that validates against a schema document that is specified in the output document via the **xsi:schemaLocation** attribute defined in XML.

10.8.4.4 outputSchema parameter

The **outputSchema** parameter is used to indicate the schema of the output that is generated in response to a **GetRecords** request. The default value for this parameter shall be *OGCCORE* indicating that the schema for the core returnable properties (as defined in subclause 6.3.3) shall be used. Application profiles may define additional values for **outputSchema** and may redefine the default value but all profiles must support the value *OGCCORE.*

Examples values for the **outputSchema** parameter might be *FGDC*, or *ISO19119, ISO19139* or *ANZLIC*. The list of supported output schemas must be advertised in the capabilities document.

10.8.4.5 startPosition parameter

The **startPosition** paramater is used to indicate at which record position the catalogue should start generating output. The default value is *1* meaning it starts at the first record in the result set.

10.8.4.6 maxRecords attribute

The **maxRecords** parameter is used to define the maximum number of records that should be returned from the result set of a query. If it is not specified, then 10 records shall be returned. If its value is set to zero, then the behavior is indentical to setting "*resultType=HITS*" as described in Subclause 10.8.4.2.

10.8.4.7 typeName parameter

The **typeName** parameter is a list of record type names that define a set of metadata record element names which will be constrained in the predicate of the query. In addition, all or some of the these names may be specified in the query to define which metadata record elements the query should present in the response to the **GetRecords** operation.

10.8.4.8 ElementName or ElementSetName parameter

The **ElementName** parameter is used to specify one or more metadata record elements that the query should present in the response to the a **GetRecords** operation. Well known sets of element may be named, in which case the **ElementSetName** parameter may be used (e.g.brief, summary or full).

If neither parameter is specified, then a CSW shall present all metadata record elements.

As mentioned in Subclause 10.8.4.4, if the **outputFormat** parameter is set to *application/xml*, then the response to the **GetRecords** operation shall validate against a schema document that is referenced in the response using the **xmlns** attributes. If the set of metadata record elements that the client specifies in the query in insufficient to generate a valid XML response document, a CSW may augment the list of elements presented to the client in order to be able to generate a valid document. Thus a client application should expect to receive more than the requested elements if the output format is set to XML.

10.8.4.9 Predicate languages

Each request encoding (XML and KVP) has a specific mechanism for specifying the predicate language that will be used to constrain a query.

In the XML encoding, the element <**Constraint**> is used to define the query predicate. The root element of the content of the <**Constraint**> element defines the predicate language that is being used. Two possible root elements are **<ogc:Filter>** for the OGC XML filter encoding, and **<csw:CqlText>** for a common query language string. An example predicate specification in the XML encoding is:

```
<Constraint>
   <CqlText>prop1!=10</CqlText>
</Constraint>
```

In the KVP encoding, the parameter **CONSTRAINTLANGUAGE** is used to specify the predicate language being used.

The **Constraint** parameter is used to specify the actual predicate. For example, to specify a CQL predicate, the following parameters would be set in the KVP encoding:

```
...CONSTRAINTLANGUAGE=CQL_TEXT&CONSTRAINT=”prop1!=10”...
```

10.8.4.10 SortBy parameter

The result set may be sorted by specifying one or more metadata record elements upon which to sort.

In KVP encoding, the **SORTBY** parameter is used to specify the list of sort elements. The value for the **SORTBY** parameter is a comma-separated list of metadata record element names upon which to sort the result set. The format for each element in the list shall be either *element name:A* indicating that the element values should be sorted in ascending order or *element name:D* indicating that the element values should be sorted in descending order.

For XML encoded requests, the <**ogc:SortBy**> element is used to specify a list of sort metadata record elements. The attribute **sortOrder** is used to specify the sort order for each element. Valid values for the **sortOrder** attribute are *ASC* indicating an ascending sort and *DESC* indicating a descending sort.

10.8.4.11 DistributedSearch parameter

The **DistributedSearch** parameter may be used to indicate that the query should be distributed. The default query behaviour, if the **DistributedSearch** parameter is set to *FALSE* (or is not specified at all), is to execute the query on the local server. In the XML encoding, if the <**DistributedSearch**> element is not specified then the query is executed on the local server.

The **hopCount** parameter controls the distributed query behaviour by limiting the maximum number of message hops before the search is terminated. Each catalogue decrements this value by one when the request is received and does not propagate the request if the hopCount=0.

10.8.4.12 ResponseHandler parameter

The **ResponseHandler** parameter is a flag that indicates how the **GetRecords** operation should be processed by a CSW.

If the parameter is not present, then the **GetRecords** operation is processed synchronously meaning that the client sends the **GetRecords** request to a CSW and waits to receive a valid response or exception message. The CSW immediately processes the **GetRecords** request while the client waits for a response. The problem with this mode of operation is that the client may timeout waiting for the CSW to process the request.

If the **ResponseHandler** parameter is present, the **GetRecords** operation is processed asynchronously. In this case, the CSW responds immediately to a client's request with an acknowledgment message that tells the client that the request has been received and validated, and notification of completion will be sent to the URI specified as the value of the **ResponseHandler** parameter. The following XML Schema fragment defines the acknowledgement message:

```
<xsd:element name="Acknowledgement" type="csw:AcknowledgementType"/>
<xsd:complexType name="AcknowledgementType">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
      This is a general acknowledgement response message for all
```

```
        requests that may be handled in an asynchronous manner.
        Echo-Request - Echoes the submitted request message
        RequestId   - identifier for polling purposes (if no response
                handler is available, or the URL scheme is
                unsupported)
        </xsd:documentation>
     </xsd:annotation>
     <xsd:sequence>
        <xsd:element name="Echo-Request"
                type="csw:Echo-RequestType" />
        <xsd:element name="RequestId"
                type="xsd:anyURI" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="timeStamp" type="xsd:dateTime" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="Echo-RequestType">
     <xsd:sequence>
        <xsd:any namespace="##any" processContents="lax" />
     </xsd:sequence>
  </xsd:complexType>
```

The acknowlegment message echos the client's request, using the <**EchoedRequest**> element, and may include an optionally generated request identifier using the <**RequestId**> element.

The **GetRecords** request may then be processed at some later time, taking as much time as is required to complete the operation. When the operation is completed, a **<csw:GetRecordsResponse>** message or exception message (if a problem was encountered) is sent to the URI specified as the value of the **ResponseHandler** parameter using the protocol encoded therein. Common protocols include ***ftp*** and ***mailto***.

10.8.5 Response

The following XML-Schema fragment defines the XML format response to a **GetRecords** operation:

```
  <xsd:element name="GetRecordsResponse"
               type="csw:GetRecordsResponseType"
               id="GetRecordsResponse"/>
  <xsd:complexType name="GetRecordsResponseType">
     <xsd:sequence>
        <xsd:element name="RequestId"
                     type="xsd:anyURI" minOccurs="0"/>
        <xsd:element name="SearchStatus"
                     type="csw:RequestStatusType"/>
        <xsd:element name="SearchResults"
                     type="csw:SearchResultsType"/>
     </xsd:sequence>
     <xsd:attribute name="version" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="RequestStatusType" id="RequestStatusType">
     <xsd:attribute name="status"
                    type="csw:StatusType" use="required"/>
     <xsd:attribute name="timestamp"
                    type="xsd:dateTime" use="optional"/>
  </xsd:complexType>
  <xsd:simpleType name="StatusType" id="StatusType">
     <xsd:restriction base="xsd:string">
```

```
         <xsd:enumeration value="complete"/>
         <xsd:enumeration value="subset"/>
         <xsd:enumeration value="interim"/>
         <xsd:enumeration value="none"/>
         <xsd:enumeration value="processing"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="SearchResultsType" id="SearchResultsType">
      <xsd:sequence>
         <xsd:element ref="csw:AbstractRecord"
                      minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="requestId" type="xsd:anyURI"
                     use="optional"/>
      <xsd:attribute name="resultSetId"
                     type="xsd:anyURI"
                     use="optional"/>
      <xsd:attribute name="elementSet"
                     type="csw:ElementSetType"
                     use="optional"/>
      <xsd:attribute name="recordSchema"
                     type="xsd:anyURI"
                     use="optional"/>
      <xsd:attribute name="numberOfRecordsMatched"
                     type="xsd:nonNegativeInteger" use="required"/>
      <xsd:attribute name="numberOfRecordsReturned"
                     type="xsd:nonNegativeInteger" use="required"/>
      <xsd:attribute name="nextRecord"
                     type="xsd:nonNegativeInteger" use="required"/>
      <xsd:attribute name="expires"
                     type="xsd:dateTime"
                     use="optional"/>
   </xsd:complexType>
```

The <**GetRecordsResponse**> element is a container for the response of the **GetRecords** operation. Three levels of detail may be contained in the response document.

The <**RequestId**> element may be used to correlate the response to a **GetRecords** request for which a value was defined for the **requestId** attribute.

The <**SearchStatus**> element must be present and indicates the status of the response. The **status** attribute is used to indicate the completion status of the **GetRecords** operation. Table 65 shows the possible values for the **status** attribute.

Table 65 — Values of the "status" attribute

Value	Description
complete	The request was successfully completed and valid results are available or have been returned.
subset	The request was successfully completed and partial valid results are available or have been returned. In this case subsequest queries with new start positions may be used to see more results.
interim	The request was successfully completed and partial results are available or have been returned but the results may not be valid. For example, an intermediate server in a distributed search may have failed cause the partial, invalid result set to be generated.
processing	The request is still processing. When completed, the response will be sent to the specified response handler.
none	No records found.

The <**SearchResults**> element is a generic XML container for the actual response to a **GetRecords** request. The content of the <**SearchResults**> element is the set of records returned by the **GetRecords** operation. The actual records returned by the catalogue should substitute for the element <**csw:AbstractRecord**>. Table 66 describes the attributes that can appear on the <**SearchResults**> element.

Table 66 — <searchStatus> Parameters

Parameters	Description
resultSetId	A server-generated identifier for the result set. May be used in subsequent GetRecords operations to further refine the result set. If the server does not implement this capability then the attribute should be omitted.
elementSet	The element set returned (brief, summary or full).
recordSchema	A reference to the type or schema of the records returned.
rumberOfRecordsMatched	Number of records found by the GetRecords operation.
numberOfRecordsReturned	Number of records actually returned to client. This may not be the entire result set since some servers may limit the number of records returned to limit the size of the response package transmitted to the client. Subsequent queries may be executed to see more of the result set. The *nextRecord* attribute will indicate to the client where to begin the next query.
nextRecord	Start position of next record. A value of 0 means all records have been returned.
expires	An ISO 8601 format date indicating when the result set will expire. If this value is not specified then the result set expires immediately.

10.8.6 Examples

KVP encoded example

```
http://www.someserver.com/csw/csw.cgi?request=GetRecords&version=2.0.0&out
putFormat=application/xml&outputSchema=csw:Record&namespace=csw:http://www
.opengis.org/cat/csw&ResponseHandler="mailto:pvretano@cubewerx.com"&typeNa
me=csw:Record&elementSetName=brief&constraintlanguage=CQLTEXT&constrain="c
sw:AnyText Like '%polution%'"
```

XML encoded request

```
<csw:GetRecords version="2.0.1"
      outputFormat="application/xml"
      outputSchema="csw:Record">
   <csw:ResponseHandler>
   ftp://www.myserver.com/pub/MyQuery_Resp.xml
   </csw:ResponseHandler>
   <csw:Query typeName="csw:Record">
      <csw:ElementSetName>brief</csw:ElementSetName>
      <csw:Constraint>
         <ogc:Filter>
            <ogc:PropertyIsLike wildCard="%" singleChar="_" escape="\">
               <ogc:PropertyName>
                  /csw:Record/csw:AnyText
               </ogc:PropertyName>
               <ogc:Literal>%polution%</ogc:Literal>
            </ogc:PropertyIsLike>
         </ogc:Filter>
      </csw:Constraint>
   </csw:Query>
</csw:GetRecords>
```

10.9 GetRecordById operation

10.9.1 Introduction

The mandatory **GetRecordById** request retrieves the default representation of catalogue records using their identifier. The **GetRecordById** operation is an implementation of the **Present** operation from the general model. This operation presumes that a previous query has been performed in order to obtain the identifiers that may be used with this operation. For example, records returned by a **GetRecords** operation may contain references to other records in the catalogue that may be retrieved using the **GetRecordById** operation. This operation is also a subset of the **GetRecords** operation, and is included as a convenient short form for retrieving and linking to records in a catalogue.

10.9.2 KVP encoding

Table 74 specifies the keyword value pair encoding for the **GetRecordById** operation request.

NOTE To reduce the need for readers to refer to other parts of this document, the first three parameters listed below are copied from Table 56 in Subclause 10.3.5 of this document.

Table 67 — KVP encoding for GetRecordById operation request

Keyword [b]	Data type and value	Optionality and use	Parameter in general model
REQUEST	Character String Fixed value of "GetRecordById", case insensitive.	One (Mandatory) [a]	(none)
service	Character String Fixed values of "CSW"	One (Mandatory)	serviceId
version	Character String Fixed value of "2.0.0"	One (Mandatory)	(none)
ElementSetName	CodeList with allowed values: "brief", "summary" or "full"	Zero or one (Optional) Default value is "summary"	responseElements
Id	Comma separated list of anyURI	One (Mandatory)	

a The REQUEST parameter contains the same information as the name of the < GetRecordById> element in XML encoding.

b Parameter keywords are case insensitive for KVP encoding

10.9.3 XML encoding

The following XML-Schema fragment defines the XML encoding for the **GetRecordById** operation request:

```
<xsd:element name="GetRecordById" type="csw:GetRecordByIdType"/>
<xsd:complexType name="GetRecordByIdType">
  <xsd:complexContent>
    <xsd:extension base="csw:RequestBaseType">
      <xsd:sequence>
        <xsd:element name="Id" type="xsd:anyURI" maxOccurs="unbounded"/>
        <xsd:element ref="csw:ElementSetName" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

10.9.4 Parameter descriptions

10.9.4.1 ElementSetName parameter

The **ElementSetName** parameter is used to specify a named, predefined set of metadata record elements from each source record that should be presented in the response to the operation. The predefined set names of *brief*, *summary* and *full* are meant to represent different level of detail of the source record with *brief* representing the least amount of detail and *full* representing all the metadata record elements. The sets of metadata record element names that correspond to *brief*, *summary* and *full* shall be defined in an Application Profile.

10.9.4.2 Id parameter

The **Id** parameter is a comma-separated list of record identifiers for the records that a CSW must return to the client. In the XML encoding, one or more <**Id**> elements may be used to specify the record identifier to retrieve.

If any of the identifiers specified in the operation is invalid, then the operation should fail and an exception message should be returned as described in Subclause 10.3.2.3.

10.9.5 Response

The following XML Schema fragment defines the response to a GetRecordById request:

```
<xsd:element name="GetRecordByIdResponse"
         type="csw:GetRecordByIdResponseType"/>
<xsd:complexType name="GetRecordByIdResponseType"/>
   <xsd:sequence>
      <xsd:element ref="csw:AbstractRecord" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

This is simply the list of requested records. The response records should substitute for the element **<csw:AbstractRecord>**.

10.9.6 Examples

KVP encoded request:

```
http://www.someserver.com/csw/csw.cgi?request=GetRecordById&version=2.0.0&i
d=REC-10,REC-11,REC-12
```

XML encoded request:

```
<csw:GetRecordById version="2.0.1">
   <csw:Id>REC-10</csw:Id>
   <csw:Id>REC-11</csw:Id>
   <csw:Id>REC-12</csw:Id>
</csw:GetRecordById>
```

10.10 Record locking

This specification does not define a locking interface, instead relying on the underlying repository to mediate concurent access to catalogue records. A profile of this specification may define a locking interface if it is found to be needed.

10.11 Transaction operation

10.11.1 Introduction

The optional **Transaction** operation defines an interface for creating, modifying and deleting catalogue records. The specific payload being manipulated must be defined in a profile.

10.11.2 KVP encoding

There is no KVP encoding for transaction operation request, because there is no convenient way to encode the transaction payloads using keyword-value pairs. In addition, the actual text of a transaction message may be very long, again making it inconvenient to use KVP encoding.

10.11.3 XML encoding

10.11.3.1 Overview

The following XML schema fragment defines the XML encoding of the Transaction operation request:

```
<xsd:element name="Transaction" type="csw:TransactionType"/>
<xsd:complexType name="TransactionType">
   <xsd:complexContent>
      <xsd:extension base="csw:RequestBaseType">
         <xsd:sequence>
            <xsd:choice minOccurs="1" maxOccurs="unbounded">
               <xsd:element name="Insert" type="csw:InsertType"/>
               <xsd:element name="Update" type="csw:UpdateType"/>
               <xsd:element name="Delete" type="csw:DeleteType"/>
            </xsd:choice>
         </xsd:sequence>
         <xsd:attribute name="requestId" type="xsd:anyURI"
                        use="optional"/>
         <xsd:attribute name="verboseResponse"
                        type="xsd:boolean"
                        use="optional" default="false"/>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

The <**Transaction**> element defines an atomic unit of work and is a container for one or more insert, update and/or delete actions.

The **requestId** attribute may be used by a client application to associate a user-defined identifier with the operation.

The **verboseResponse**attribute is a boolean that may be used by a client to indicate to a server the amount of detail to generate in the rsponse. A value of FALSE means that a CSW should generate a terse or brief transaction response. A value of TRUE, or the absence of the attribute, means that the normal detailed transaction response should be generated. The schema of transaction response is defined in Subclause 10.11.4.

10.11.3.2 Insert action

The following XML-Schema fragment defines an insert action:

```
<xsd:complexType name="InsertType" id="InsertType">
  <xsd:sequence>
    <xsd:any processContents="strict" namespace="##other"
             maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The <**Insert**> element is a container for one or more records that are to be inserted into the catalogue. The schema of the record(s) must conform to the schema of the information model that the catalogue supports as described using the DescribeRecord operation.

The **handle** attribute is an additional parameter not defined in the general model. It is used in the XML encoding to associate a mnemonic name with each action contained in a <**Transaction**> element for the purpose of error handling. If a CSW encounters an error processing a transaction request and the **handle** attribute is defined, the CSW can localize the source of the problem for the client by specifying the handle in the exception response.

10.11.3.3 Update action

The following XML Schema fragment defines an update action:

```
<xsd:complexType name="UpdateType" id="UpdateType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:any processContents="strict" namespace="##other" />
      <xsd:sequence>
        <xsd:element ref="csw:RecordProperty" maxOccurs="unbounded"/>
        <xsd:element ref="csw:Constraint"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The <**csw:Update**> element is used to specify values to be used to change existing information in the catalogue. If the <**csw:Update**> element contains a child element declared in some other namespace (i.e., **not** "http://www.opengis.net/cat/csw"), then the corresponding record in the catalogue shall be replaced. The record type must be defined in an application profile. If individual record property values are specified in the <**Update**> element, using the <**csw:RecordProperty**> element, then those individual property values of the catalogue record shall be updated.

The <**csw:RecordProperty**> element contains a <**csw:Name**> element and a <**csw:Value**> element. The <**csw:Name**> element is used to specify the name of the record property to be updated. The value of the <**csw:Name**> element may be a path expression to identify complex properties. The <**csw:Value**> element contains the value that will be used to update the record in the catalogue.

The number of records affected by an <**csw:Update**> action is determined by the contents of the <**csw:Constraint**> element. The <**csw:Constraint**> element is defined in Subclause 10.3.7 and is used to define the set of catalogue records that the update operation will affect. In order to prevent all records in a catalogue from inadvertently being updated, the <**csw:Constraint**> element must be specified.

The optional **typeName** attribute may be used to specify the collection name from which records will be updated.

The **handle** attribute is described in Subclause 10.11.3.2.

10.11.3.4 Delete action

The following XML Schema fragment defines an delete action:

```
<xsd:complexType name="DeleteType" id="DeleteType">
  <xsd:sequence>
    <xsd:element ref="csw:Constraint" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="typeName" type="xsd:anyURI" use="optional"/>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

The <**Delete**> element contains a <**csw:Constraint**> element (see Subclause 10.3.7) that identifies a set of records that are to be deleted from the catalogue. The <**csw:Constraint**> element must be specified in order to prevent every record in the catalogue from inadvertently being deleted.

The **typeName** attribute is used to specify the collection name from which records will be deleted.

The **handle** attribute is described in Subclause 10.11.3.2.

10.11.4 Response

The following XML Schema fragment defines the response that must be generated after a CSW server executes a transaction request:

```
   <xsd:element name="TransactionResponse"
            type="csw:TransactionResponseType"/>
   <xsd:complexType name="TransactionResponseType">
      <xsd:sequence>
         <xsd:element name="TransactionSummary"
                  type="csw:TransactionSummaryType"/>
         <xsd:element name="InsertResult"
                  type="csw:TransactionResultType"
                  minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" use="optional"/>
   </xsd:complexType>

   <xsd:complexType name="TransactionSummaryType">
      <xsd:sequence>
         <xsd:element name="totalInserted"
                  type="xsd:nonNegativeInteger"
                  minOccurs="0"/>
         <xsd:element name="totalUpdated"
                  type="xsd:nonNegativeInteger"
                  minOccurs="0"/>
         <xsd:element name="totalDeleted"
                  type="xsd:nonNegativeInteger"
                  minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="requestId" type="xsd:anyURI" use="optional"/>
   </xsd:complexType>

   <xsd:complexType name="TransactionResultType">
      <xsd:sequence>
      <xsd:element ref="csw:AbstractRecord" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="handleRef" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
```

The transaction response message conveys two pieces of information. First, it reports a summary of the transaction by indicating the number records created, updated or deleted by the transaction. Second, the transaction response message indicates the results of each insert operation found in the transaction in the form of the <**InsertResult**> element.

The <**InsertResult**> element may appear zero or more times in the transaction response. It is used to report to the client a brief representation of each new record, including the record identifier, created in the catalogue. The records must be reported in the same order in which the <**Insert**> elements appear in a transaction request and must map 1-to-1. Optionally, the handle attribute may be used to correlate a particular <**Insert**> element in the Transaction request with an <**InsertResult**> element found in the transaction response.

10.12 Harvest operation

10.12.1 Introduction

The general model defines two operations in the Manager class that may be used to create or update records in the catalogue. They are the **transaction** operation and the **harvestRecords** operation. The **transaction** operation may be used to "push" data into the catalogue and is defined in Subclause 10.11. This subclause defines the optional **Harvest** operation, which is an operation that "pulls" data into the catalogue. That is, this operation only references the data to be inserted or updated in the catalogue, and it is the job of the catalogue service to resolve the reference, fetch that data, and process it into the catalogue.

The **Harvest** operation had two modes of operation, controlled by a flag in the request. The first mode of operation is a synchronous mode in whice the CSW receives a **Harvest** request from the client, processes it immediately, and sends the results to the client while the client waits. The second mode of operation is asynchronous in that the server receives a **Harvest** request from the client, and sends the client an immediate acknowledgement that the request has been successfully received. The server can then process the **Harvest** request whenever it likes, taking as much time as is required and then send the results of the processing to a URI specified in the original **Harvest** request. This latter mode of operation is included to support **Harvest** requests that could run for a period of time longer than most HTTP timeout's will allow.

Processing a **Harvest** request means that the CSW resolves the URI pointing to the metadata resource, parses the resource, and then creates or modifies metadata records in the catalogue in order to register the resource. This operation may be performed only once or periodically depending on how the client invokes the operation.

10.12.2 KVP encoding

Table 76 specifies the keyword-value pair encoding for the Harvest operation request.

NOTE To reduce the need for readers to refer to other parts of this document, the first three parameters listed below are copied from Table 56 in Subclause 10.3.5 of this document.

Table 68 — KVP encoding for Harvest operation request

Keyword [c]	Data type and value	Optionality and use	Parameter in general model
REQUEST	Character String Fixed value of *HarvestRecords*, case insensitive	One (Mandatory) [a]	(none)
service	Character String Fixed values of "CSW"	One (Mandatory)	serviceId
version	Character String Fixed value of *2.0.0*	One (Mandatory)	(none)
NAMESPACE	List of Character String, comma separated Used to specify a namespace and its prefix Format must be [*prefix:*]*url*. If the *prefix* is not specified then this is the default namespace.	Zero or one (Optional) [b] Include value for each distinct namespace used by all qualified names in the request. If not included, all qualified names are in default namespace	(none)
Source	URI Reference to the source from which the resource is to be harvested	One (Mandatory)	Source
ResourceType	URI Referenence to the type of resource being harvested	Zero or one (Optional) [d] If not included, see Subclause 10.12.4.2	Type
ResourceFormat	Character String MIME type indicating format of the resource being harvested	Zero or one (Optional) Default value is *application/xml*	resourceFormat
ResponseHandler	URL A reference to a person or entity that the CSW should respond to when it has completed processing Harvest request anynchronously	Zero or one (Optional) If not included, process request synchronously	responseHandler
HarvestInterval	Period Must conform to ISO8601 Period syntax.	Zero or one (Optional) If not specified, then harvest only once in response to the request.	harvestInterval

a The REQUEST parameter contains the same information as the name of the <Harvest> element in XML encoding.

b The NAMESPACE parameter contains the same information as the xmlns attributes which may be used to convey namespace information in XML encoding.

c Parameter keywords are case insensitive for KVP encoding.

d If not specified, the server will have to determine the type by some other means. For example, schema references in the document being harvested.

10.12.3 XML encoding

The following XML-Schema fragment defines the XML encoding for a **Harvest** operation request:

```
<xsd:element name="Harvest" type="csw:HarvestType"/>
<xsd:complexType name="HarvestType">
   <xsd:complexContent>
      <xsd:extension base="csw:RequestBaseType">
         <xsd:sequence>
            <xsd:element name="Source" type="xsd:anyURI"/>
            <xsd:element name="ResourceType" type="xsd:anyURI"
                         minOccurs="0"/>
            <xsd:element name="ResourceFormat" type="xsd:string"
                         minOccurs="0" default="application/xml"/>
            <xsd:element name="HarvestInterval" type="xsd:duration"
                         minOccurs="0"/>
            <xsd:element name="ResponseHandler" type="xsd:anyURI"
                         minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

10.12.4 Parameter descriptions

10.12.4.1 Source parameter

The **Source** parameter is used to specify a URI reference to the metadata resource to be harvested.

10.12.4.2 ResourceType parameter

The **ResourceType** parameter is a reference to a schema document that defines the structure of the resource being harvested. This is an optional parameter and if it not specified then the catalogue must employee other means to determine the type of resource being harvested. For example, the catalogue may use schema references in the input document to determine the resource type, or perhaps parse the root element to determine the type of metadata being harvested (e.g. <fgdc:metadata> is the root element of an FGDC document).

10.12.4.3 ResourceFormat parameter

The **ResourceFormat** paramter is used to indicate the encoding used for the resource being harvested. This parameter is included to support the harvesting of metadata resources available in various formats such as plain text, XML or HTML. The values of this parameter must be a MIME type. If the parameter is not specified then the default value of *application/xml* should be assumed.

10.12.4.4 ResponseHandler parameter

The **ResponseHandler** parameter is a flag that indicates how the **Harvest** operation should be processed by a CSW server.

If the parameter is not present, then the **Harvest** operation is processed synchronously meaning that the client sends the **Harvest** request to a CSW and then waits to receive a **HarvestResponse** or exception message. The CSW immediately processes the **Harvest** request, while the client waits for a response. The problem with this mode of operation is that the client may timeout waiting for the server to process the request.

If the parameter is present, the **Harvest** operation is processed asynchronously. In this case, the server responds immediately to a client's request with an acknowledgement message as defined in Subclause 10.8.4.13. The acknowlegment message echos the client's request, using the <**EchoedRequest**> element, and may include an optionally generated request identifier using the <**RequestId**> element. The acknowledgement message tells the client that the request has been received and notification of completion will be send to the URL specified as the value of the **ResponseHandler** parameter. The **Harvest** request may then be processed at some later time taking as much time as is required to complete the operation. When the operation is completed, a **HarvestResponse** message or exception message (if a problem was encountered) is sent to the URL specified as the value of the **ResponseHandler** parameter.

10.12.4.5 HarvestInterval Parameter

The **HarvestInterval** parameter is used to specify the period of time, in ISO 8601 period format, that should elapse before a CSW attempts to re-harvest the specified resource thus refreshing it copy of a resouce.

If no **HarvestInterval** parameter is specified then the resource is harvested only once in response to the **Harvest** request.

10.12.5 Response

The **Harvest** operation can respond in one of two ways depending on the presence or absence of the **ResponseHandler** parameter.

If the **ResponseHandler** parameter is present, then the CSW server should verify the request syntax and immediately respond to the client with a acknowledgment message as defined in Subclause 10.12.4.4. Later, after the server has processed the request, it should generate a **HarvestResponse** message and send it to the URI specified by the **ResponseHandler** parameter using the protocol encoded therein. Common protocols are ***ftp*** for sending the response to ftp server and ***mailto*** which may be used to send the response to an email address.

If the **ResponseHandler** parameter is not present, then the CSW server should process the **Harvest** request immediately and respond to the waiting client with a **HarvestResponse** message.

The following XML-Schema fragment defines the **HarvestResponse** message:

```
<xsd:element name="HarvestResponse" type="csw:TransactionResponseType"/>
```

If the **Harvest** attempt is successful, the response may include summary representations of the newly created or modified catalogue object(s). The response is the same as the **TransactionResponse**.

10.12.6 Examples

KVP encoded example:

```
http://www.myserver.com/csw/csw.cgi?request=Harvest&version="2.0.0"&source
=http://www.yourserver.com/metadata.xml&resourcetype=FGDC&resourceformat=a
pplication/xml&responsehandler=mailto:pvretano@cubewerx.com&harvestinterva
l=P2W
```

XML encoded example:

```
<csw:Harvest version="2.0.0">

   <csw:Source>http://www.yourserver.com/metadata.xml</csw:Source>
   <csw:ResourceType>FGDC</csw:ResourceType>
   <csw:ResourceFormat>application/xml</csw:ResourceFormat>
   <csw:HarvestInterval>P2W</csw:HarvestInterval>
   <csw:ResponseHandler>
ftp://ftp.myserver.com/HarvestResponses>/csw:ResponseHandler>
</csw:Harvest>
```

10.13 XML Schemas

The CSW abilities specified in this Clause directly and indirectly use several newly specified XML Schemas, included in the zip file with this document. These new XML Schema files are named:

CSW-discovery.xsd

CSW-publication.xsd

sort.xsd

records.xsd

rec-dcmes.xsd

rec-dcterms.xsd

After this document is approved, these new XML Schema files will be posted at the URL http://schemas.opengis.net/ for electronic access. In the event of a discrepancy between the attached and online versions of the XML Schema files, the online files shall be considered authoritative.

These new XML Schemas build on the XML Schemas defined in the Filter Encoding Implementation Specification [OGC 02-059], and the OWS Common Implementation Specification [OGC 04-016r2], and described in those documents.

11 Specializing general model through protocol bindings and profiles

11.1 Introduction

This subclause provides an overview of the core elements of the General Catalogue Model and how these may be used in protocol bindings and application profiles.

The General Catalogue Model consists of an abstract model and a General Interface Model. The abstract query model specifies a BNF grammar for a minimal query syntax and a set of core search attributes (names, definitions, conceptual datatypes).The General Interface Model specifies a set of service interfaces that support the discovery, access, maintenance and organization of catalogues of geospatial information and related resources; these interfaces may be bound to multiple application protocols, including the HTTP protocol that underlies the World Wide Web. This specification also specifies bindings to CORBA/IIOP and Z39.50. Furthermore, all behaviour requiring sessions is expressed by a dynamic model of conversation state and state transitions. The model expresses the states and messages that trigger the changes in state.

Implementations are constrained by the protocol binding clauses of this specification. Each protocol binding includes a mapping from the general interfaces, operations, and parameters specified in this clause to the constructs available in a chosen protocol. Application profiles are intended to further document implementation choices.

An Application Profile is based on one of the protocol bindings in the base specification. In the case of the Catalogue Services Specification, a profile could reference CORBA/IIOP, Z39.50, or the HTTP/1.1 protocol bindings. In most, but not all, protocol bindings, there may be restrictions or refinements on implementation agreed within an implementation community. A graphic model of the relationships is shown in Figure 31.

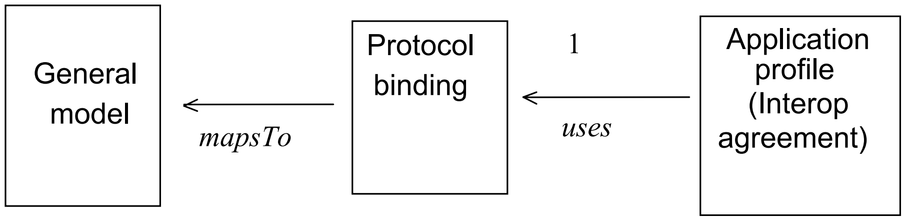

Figure 31 — Relationship of general model, protocol binding, and application profile

11.2 Interface definitions

The various elements of the General Catalogue Interface Model provide functional behaviours and capabilities to address particular areas of concern. A protocol binding may realise specific configurations of these components to serve different purposes (e.g. a read-only catalogue for discovery, a transactional catalogue for discovery and publication, or a 'stateful' catalogue that also supports session management).

A compliant protocol binding of the catalogue service is required to implement the OGC_Service, Catalogue Service, and Discovery classes. A protocol binding may also include any of the optional classes associated with the Catalogue Service class. A compliant implementation of a protocol binding must recognise all operations defined within each class included in the protocol binding, and shall generate a service exception report indicating when a particular operation is not implemented (in such cases the operation is abstract—an implementation is not required).

The protocol binding clauses of this specification provides more detail on the implementation of the general interfaces. In effect, each binding maps these interfaces to a particular application protocol. For example, the names of the classes and operations in this general UML model are changed in some of the protocol bindings. The names of some operation parameters are also changed in some protocol bindings. However, the interfaces and operations specified in all Protocol Bindings shall be consistent with the semantics and granularity of interaction specified in the General Interface Model.

Application profiles, which will appear as separate documents may further specialise the implementation of these interfaces and their operations, including adding classes and parameters. However, the application profile is a specialization of the parent protocol binding, in that the names of the operations and the parameters cannot be changed.

11.3 Query model components

11.3.1 Query language/model

Many OGC service operations have the requirement to pass and process a query as a structure to perform a search. There are several query languages and messaging mechanisms identified within OGC specifications. Binding protocols and application profiles should be explicit about the selected query languages and any features peculiar to a scope of application. The following items should be addressed in specialization of a Protocol Binding or an Application profile with respect to query language support:

a) *Support for "abstract" queries*, against well-known *access* points (e.g. core search properties). Some specifications promote or require the exposure of well-known field-like objects as common search targets (queryables), allowing interrogation of a service without prior negotiation on information content. The mandatory queryable attributes which must be recognised by all OGC Catalogue Services is discussed in Subclause TBD.

b) *Selection of a query language*. Identify the name and version of required query language(s) anticipated by this Protocol Binding or Application Profile for use.

c) *Supported data types* (e.g. character, integer, coordinate, date, polygon) and operator types (e.g. inequality, proximity, partial string, spatial, temporal). Query languages may be restricted in their implementation or extended with functions not described in the base specification. This would need to be done if the base query language did not support a data type required by the OGC Common Query Language discussed in Clause 6 such as envelope.

In addition, an application profile may extend the OGC CQL or Filter syntax with functions not described in the base specification through use of the "function "construct in CQL and the "Filter " language. Use of this construct is discussed in Subclause TBD of this document. If an application protocol uses this extension method, the profile documentation should include an updated BNF grammar in addition to lists or reference documents with the enumerated data types and operator types required by this Application Profile. In addition, any description of special techniques (e.g. supporting joins or associations) that are expected by an Application Profile should be described.

11.3.2 Common search and retrieval elements

The abstract information model is discussed in Clause 6; this model consists of a small set of abstract search elements and the specification of a common "summary" element set to allow queries across protocol bindings and even from outside the OGC domain. Each Protocol Binding should specialize this model by:

a) Specify the syntax of the globally unique Identifiers including any registration authority information

b) Map the core search (queryable) elements into a concrete syntax based on the chosen record format(s)

c) Define a "summary"element set that corresponds to the "summary" element set in the Catalogue general model

An application profile is expected to fully specify the conceptual information model adopted by the user community. This process and resulting artefacts are further discussed in Subclause 6.2.5 and the remainder of this clause.

11.4 Catalogue Application Profiles

ISO TR 10000-1:1998 describes a general framework for functional standardization and defines the concept of a profile. A profile identifies the use of particular options available in one or more base standards and it also provides a basis for developing conformance tests; a compliant profile must not contradict the base specifications or otherwise give rise to non-conforming conditions. An *application profile* specifies the use of an application-layer protocol (e.g., Z39.50, HTTP/1.1, CORBA/IIOP) in order to provide for the structured transfer of information between systems (ISO/IEC TR 10000-2:1998).

A catalogue application profile binds a set of functional components (with interfaces specified as part of a protocol binding) to an abstract information model—expressed using UML—that has one or more concrete representations of catalogue content. Each representation is an Internet media type that conforms to a schema defined using some schema language (e.g., ASN.1, XML Schema, RDF Schema). An application profile specifies a set of functional components that are provided by a conforming implementation (Figure 32).

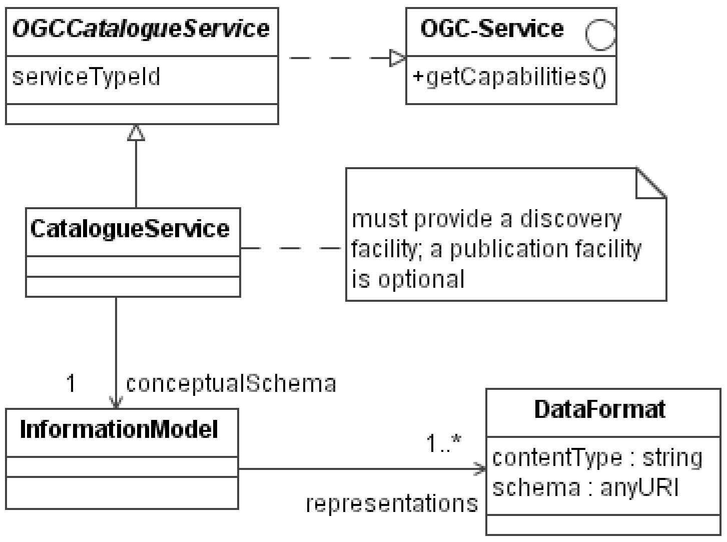

Figure 32 — Application profiles specify concrete catalogue services

An application profile is derived from one or more base specifications in order to address particular needs or requirements. The general OGC catalogue model defines common behaviours and interfaces that have general utility, but in practice there is no single solution that fits everyone's needs. Catalogue application profiles specify refinements or extensions that are targeted toward specific implementation communities; for these communities it is the application profile that represents the standard for conformance. Following ISO 19106, a **Level 1** profile is defined as a pure subset of one or more ISO standards; a **Level 2** profile includes allowable extensions and may also depend on non-ISO specifications.

Clause 10 in the ISO 19119 standard distinguishes *platform-neutral* from *platform-specific* specifications and assumes that one of the former will constitute the basis for one or more of the latter. That is, a single platform-neutral specification will give rise to multiple platform-specific specifications, each of which is bound to a particular distributed computing protocol. The OGC catalogue framework upholds this basic distinction: the general interface model is a platform-neutral description of catalogue operations; each application profile is platform-specific—it makes use of one of the protocol bindings defined in the catalogue specification.

The relationships between base specifications, application profiles, and catalogue service implementations are illustrated in Figure 33. The platform-neutral specification is one of the base specifications with which the application profile complies. A given catalogue implementation in turn conforms to one or more application profiles. The relationships shown in Figure 33 are consistent with the standard terms defined in ISO 10746 (*Information technology – Open Distributed Processing*) and ISO 9646-1 (*Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts*)[13].

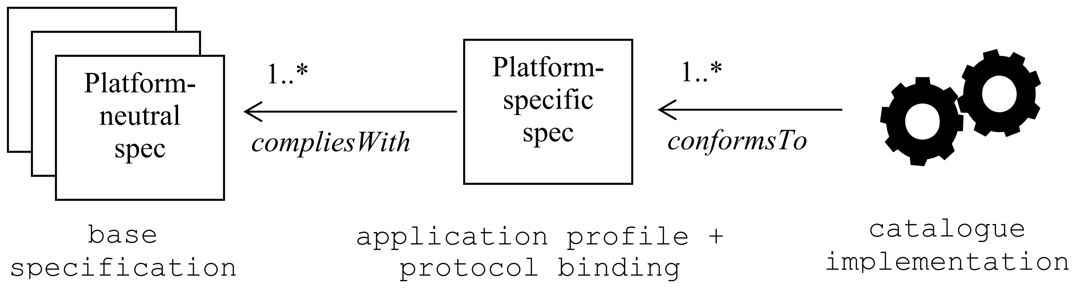

Figure 33 — Relationships between base specs, profiles, and implementations

Note that in the figure application profiles will reflect differing degrees of "thickness". For example, if a profile employs a very simple conceptual model that embodies a limited set of simple properties then its 'native' representation may include little more than the common search and retrieval elements. Profiles that utilize more sophisticated models will define a native representation that provides more information; in this case the common search and retrieval elements must be mapped to the catalogue information model.

11.5 Structure and format

11.5.1 Introduction

All application profiles must be structured as shown in Table 69. This organization complies with clause 12.3 of ISO 19106 (*Geographic information – Profiles*). A profile may introduce additional (sub)clauses as required.

[13] This usage is also closely aligned with the notion of profiles expressed in Appendix 16 of the Z39.50 information retrieval standard: "The overall goal of profiles is to improve interoperability between systems conforming to a specific profile. The implication is that an implementor does not "implement the standard" but rather, configures a Z39.50 client and/or Z39.50 server to conform to one or more profiles."

Table 69 — Structure of an application profile

Clause	Title
(front matter)	Preface
	Submitting organizations
	Revision history
	Changes to the OpenGIS Catalogue Services Specification
	Future work
	Forward
	Introduction
1	Scope
2	Conformance
3	Normative references
4	Terms and definitions
5	Symbols and abbreviations
6	System context 6.1 Application domain 6.2 Essential use cases
7	Information models 7.1 Capability classes 7.2 Catalogue information model 7.3 Supported data bindings 7.4 Service information model 7.5 Native language support
8	External interfaces 8.1 Imported protocol bindings 8.2 Interface A 8.3 Interface B . . . 8.i Query facilities 8.j General implementation guidance 8.k Security considerations
Annex A	Abstract test suite (normative)
Annex B	Design rationale (informative)

Clauses 6 through 8 convey the particulars of the application profile in terms of three ‘views’ (these correspond to the following standard ODP viewpoints: Enterprise, Information, and Computational). The three views describe various aspects of the catalogue service with respect to the base specifications; taken together they constitute the basic application architecture. The essential content of these views is summarized in the following subclauses; additional guidance can be found in the annotated profile template (OGC Document 03-101).

11.5.2 System context

This view focuses on the purpose, scope, and policies of the catalogue service (i.e., what is the system used for). It documents special requirements[14] and describes the context of use as suggested in Table 70.

Table 70 — System context: required subclauses

Subclause	Topical content
Application domain	The subject domain being addressed—identify whether this profile has a specific disciplinary focus (e.g. oceanography), or is of interest to a broader community (e.g. research, public access, or libraries) The prospective stakeholders or community of practice
Essential use cases	What the system should be able to do, what it will be used for, who will use it Typical scenarios that encompass a series of interactions between users and the catalogue system being described in order to fulfill the needs of stakeholders. The inclusion of narrative use cases with accompanying interaction and/or sequence diagrams is recommended

11.5.3 Information models

This view primarily focuses on the information structures and the semantics of information processing (i.e., what the system is about); it describes the public information model that is employed by the catalogue service and the interfaces through which it is accessed. The syntax for all supported representations of the catalogued resources must also be defined (Table 71).

Table 71 — Information models: required subclauses

Subclause	Topical content
Capability classes	Capabilities provided by the application profile, including a high-level summary of the interfaces provided (and conformance classes/levels if these are distinguished)
Catalogue information model	Kinds of information objects managed by the catalogue using UML notation—a catalogue may offer discovery and publication support for many different types of information resources (services, data sets, schemas, style sheets, reference documents, software components, ontologies, thesauri, etc.) Mappings to the common XML Record format
Supported data formats	Supported representations of the information objects using an appropriate syntax, one of which must be designated as the default representation Supported element sets (schemas) for each format
Service information model	Content model and syntax for service information
Native language support	How the catalogue service supports multiple languages and character encodings (i.e. internationalization and localization issues)

[14] Clause 7 of ISO 19106 stipulates that a profile must clearly identify the specific user requirements that are satisfied by the profile.

11.5.4 External interfaces

This view primarily focuses on documenting the externally visible behaviour of the system, including the interfaces provided by its components and the supported protocol binding(s). This view must define the request and response message structures as part of the operation signatures; it also documents supported query facilities and any relevant security considerations (Table 72). Most of the request and response message elements are imported with the protocol binding, but a (Level 2) profile may introduce extensions to meet more specialized requirements.

Table 72 — Public interfaces: required subclauses

Subclause	Topical content
Imported protocol binding	How the interfaces or functions specified for the profile are related to the imported protocol binding.
Interface specifications	Syntax and semantics of the operations provided by each interface, including relevant preconditions, postconditions, and other usage constraints Formal, language-independent interface specifications that admit multiple programming language bindings (e.g. W3C WSDL, OMG IDL) Error conditions that can be raised and how they're handled Any restrictions or variations on the use of the supported protocol binding (e.g. CORBA/IIOP, Z39.50, HTTP/1.1)
Query facilities	Supported query languages (e.g. OGC CQL/Filter, SQL-92, XPath, XQuery, etc.) extensions or restrictions to any of the above languages
Implementation guidance	Any additional information (typically non-normative) that may be helpful to implementers
Security considerations	Information regarding the provision of security functions: authentication, access control, message integrity, confidentiality, non-repudiation, audit trails

The inclusion of a UML diagram is recommended to provide an overview of the interfaces provided by a given service type, where each type provides a different—perhaps overlapping—set of interfaces (e.g. a read-only catalogue, a catalogue that allows a 'push' style of publication).

11.6 Compliance

A compliant application profile shall:

a) Include the (sub)clauses indicated in Table 72 (additional clauses MAY also be included);

b) Define the supported catalogue information model using UML as the conceptual schema language;

c) Define a set of mappings for the common XML record format data format;

d) Specify the 'native' representation of information model elements (additional representations MAY also be specified);

e) Define any extensions to the imported protocol binding.

f) Indicate how the elements of the general model are related to the corresponding elements of the profile-specific interfaces;

g) Include a conformance test suite (web-based services can do so using the OGC CITE notation).

Annex A
(normative)

Abstract conformance test suite

EDITOR'S NOTE In a future version, this annex will specify conformance checking requirements for each protocol binding.

Annex B (informative)

Description of Distributed Search

To enable Distributed Searching, the following items are needed:

a) A multi-tier Reference Architecture as provided by this specification (as defined in Subclause 7.1)

b) A data model to define how searches are to be distributed as defined by an information community

c) Messages with elements applicable to Distributed Searching as provided by this specification

To support distributed searching, a community develops a data model that determines how a search will be distributed to coordinated data servers. The OGC Catalogue General Model allows data model neutrality with respect to distributed searching.

Several of the Discovery messages defined in Subclause 10.8 contain elements that pertain to distributed searching. The query message contains elements that allow the client to request certain search behaviour with respect to distribution. The request and response messages define elements that allow for the retrieval and comprehension of a distributed result set. The request and response messages contain elements that allow for understanding the status of distributed searches.

Distributed searches can cause specific problems that should be addressed in detail by an application profile. These problems result from the possibility that within one distributed, or cascading, query a catalogue service node may be approached multiple times, resulting in duplication results or, even worse, in loops causing the whole distributed system to potentially fail. Other problems are caused by duplicate metadata entries that are served by different catalogue services.

Figure B.1 displays a case resulting in duplicates due to the same catalogue service node being queried twice.

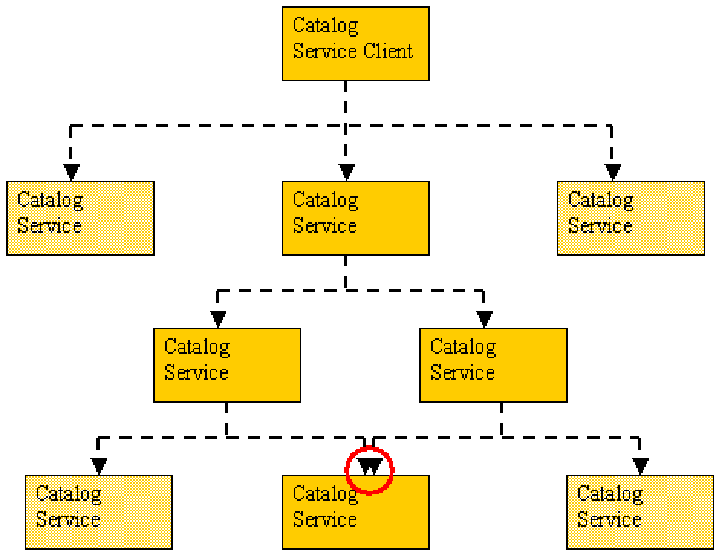

Figure B.1 — Query network topology resulting in duplicates

Unnecessary duplicates are a nuisance for the user but do not normally cause the system to fail. More problematic are loops, as shown in Figure B.2.

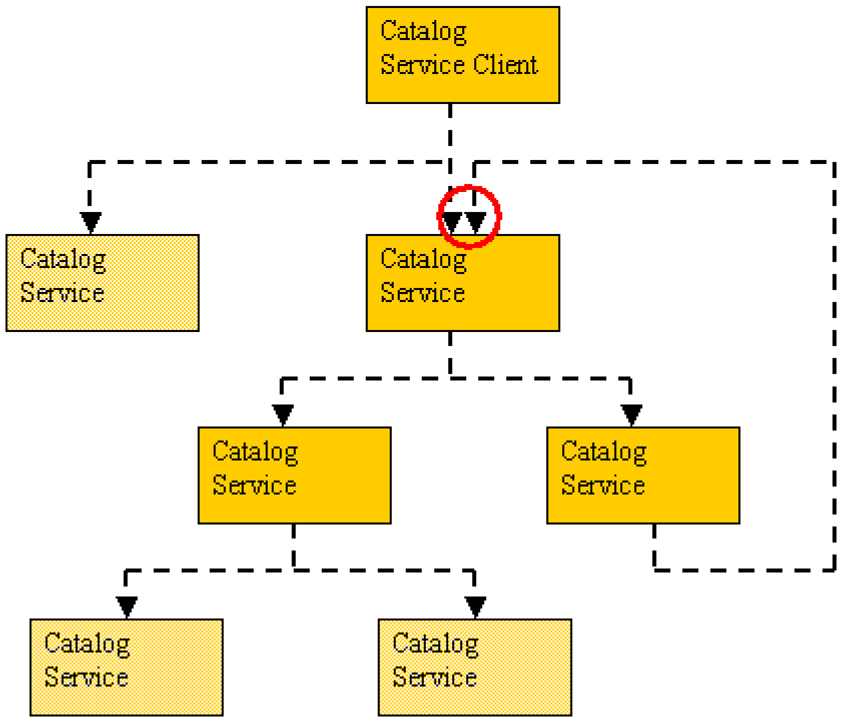

Figure B.2 — Query network topology resulting in a loop

In this case the loop causes infinite recursion – the same query is sent again and again resulting in system failure and/or timeout.

It is important to notice that both of these problems can be solved by restricting the search hierarchy to two levels – one client queries a number of catalogue services that are allowed to cascade once. Generally speaking one method to avoid these problems is to control the network topology manually. Before a query is issued, the query topology is checked for duplicates or loops.

To allow an automatic solution to this problem, mechanisms should be specified in Application Profiles of the Catalogue Specification. To make sure that one catalogue node is not approached more than once within one query, whether this happens in a loop or not, one may wish to track the nodes already accessed. A cascading catalogue service would make sure that the list of already accesses nodes of the query gets added its own identifier – most likely as a URI. By the same mechanism, a counter could be implemented, reflecting the number of catalogue services the query already passed. This could help restricting the search depth of one query. An application profile could state the maximum number of cascadings a query would traverse.

A possible solution to the problem of duplicates of metadata entries is to define unique identifiers for metadata entries that are applicable to a whole network, e.g. based on a combination of the server URL and the file identifier specific to the catalogue. Node-specific File identifiers alone are not the solution to the problem, as different catalogues can use the same ID without referring to the same dataset.

Annex C (informative) Details of order operation

C.1 Introduction

This annex provides the details of the Order operation messages from OGC Catalogue Specification 1.1.1. It is included here as background for profiles or protocol bindings that have maintained backwards compatibility with these more detailed specifications. Note the message and parameter names are the names from Catalogue Version 1.1.1.

C.2 BrokeredAccessRequest

C.2.1 Message

The BrokeredAccessRequest is a service requesting data that cannot be made available directly.

```
BrokeredAccessRequest ::= sessionID destinationID requestID additionalInfo
                          productHandle orderInformation orderID requestType
                          userInformation statusOrderUpdateType

        sessionID ::= Integer

        destinationID ::= CharacterString

        requestID ::= RequestID

        additionalInfo ::= CharacterString

        productHandle ::= CharacterString

        orderInformation ::= OrderSpecification

        orderID ::= CharacterString

        requestType ::= BrokeredAccessRequestType

        userInformation ::= UserInformation

        statusOrderUpdateType ::= StatusUpdateType
```

C.2.2 Message Parameters:

productHandle: Type = CharacterString

The product handle is the identifier for a specific product taken from the catalogue metadata for that product.

orderInformation: Type = OrderSpecification

For BrokeredAccessRequestType = orderEstimate or OrderQuoteAndSubmit,the specification of the current order request as provided as by the client or modified by the server during the estimation process.

For BrokeredAccessRequestType = orderMonitor or orderCancel, OrderSpecification is ignored and may not be supplied.

orderID: type = CharacterString

The orderID parameter provides a unique identifier for an order in progress. This ID can be used to inquire about the status of the order as it is being processed. For BrokeredAccessRequestType = orderMonitor or orderCancel, orderID shall be supplied. For requestType = orderEstimate or OrderQuoteAndSubmit, orderID shall be empty.

requestType: Type = BrokeredAccessRequestType

The request type parameter identifies the type of service the client needs from the server. Valid values are estimate, submit, monitor and cancel. Estimate is used to check if the order is valid and to request an estimate of resources required to fill the order. Submit is a request to order and deliver the products(s). Monitor provides the current status of the order. Cancel requests that the order be cancelled. The server must grant cancellation of the order. BrokeredAccessRequestType is formally defined in Subclause C.4.2.

userInformation: Type = UserInformation

To receive products it is necessary to provide requester identification, billing and delivery data as part of the order. This parameter is used to provide that data.

statusOrderUpdateType : Type = StatusUpdateType

How a given client likes to be kept informed about the status of a given order.

C.2.3 Message Operations: None

C.3 BrokeredAccessResponse

C.3.1 Response

The server generates the BrokeredAccessResponse message in response to a BrokeredAccessRequest.

BrokeredAccessResponse ::= sessionID destinationID requestID additionalInfo diagnostic format orderStatus resourceEstimate order orderID status requestType

sessionID ::= Integer

destinationID ::= CharacterString

requestID ::= RequestID

additionalInfo ::= CharacterString

diagnostic ::= CharacterString

format ::= MessageFormat

orderStatus ::= OrderStatus

resourceEstimate ::= CharacterString

order ::= CollectionName

orderID ::= CharacterString

status ::= Status

requestType ::= BrokeredAccessRequestType

orderInformation ::= OrderSpecification

C.3.2 Message Parameters:

orderStatus Type ::= OrderStatus

This parameter indicates the status of the order. The status of the order is different than the status of an Access message. The status of the message is reported in the response in the status parameter. The OrderStatus type is formally defined in Subclause C.4.6 of this specification.

resourceEstimate: Type = CharacterString

This parameter reports back on the resources needed to process and/or deliver the requested resource. Examples of these resources are time until delivery and cost.

order: Type = CollectionName

The order parameter returns a name or id of the requested product object online. This parameter can be used for direct access (such as through simple features) to the online product. The CollectionName type is formally defined in Subclause C.4.3 of this specification.

orderID: type = CharacterString

The orderID parameter provides a unique identifier for an order in progress. This ID can be used to inquire about the status of the order as it is being processed. This number is generated by the server in response to a BrokeredAccessRequest where requestType = orderEstimate or OrderQuoteAndSubmit

status: Type = Status

The Status parameter conveys the status of the requested product. The Status type is formally defined in Subclause C.4.11.

requestType: Type = BrokeredAccessRequestType

The request type parameter identifies the type of service the client needs from the server. BrokeredAccessRequestType is formally defined in Subclause C.4.2.

orderInformation: Type ::= OrderSpecification

For BrokeredAccessRequestType = orderEstimate or OrderQuoteAndSubmit, the specification of the current order request as provided as by the client or modified by the server during the estimation process. .

For BrokeredAccessRequestType = orderMonitor or orderCancel, OrderSpecification is ignored and may not be supplied.

C.3.3 Message Operations: None

C.4 Parameter Type Definitions

C.4.1 Introduction

This annex provides definitions for all of the parameter data types used in Request-Response Message Pairs. These definitions assume the use of the OGC well known data types where applicable.

C.4.2 BrokeredAccessRequestType

Recommended Implementation Type: Code_List

Used By: BrokeredAccessRequest

BrokeredAccessRequestType is a code list for identifying the nature of a brokered access request. Valid values for this type are shown in Table C.1.

Table C.1 — Brokered Access Request Types

Value	Explanation
orderEstimate	Validate and obtain the estimate of an order specification
orderQuoteAndSubmit	Obtain a quote and subsequently submit an order specification
orderMonitor	Monitor the progress of an order request
orderCancel	Cancel an order request

C.4.3 CollectionName

Recommended Implementation Type: Union data

Used By: BrokeredAccessResponse

Collection Name is a type that identifies a catalogue data resource. It can point to a catalog, catalogue entry, named catalogue subspace, named catalogue superspace or a result set. This type is a "C" union of two base types:

collection ID (character string)

collection Name (character string).

C.4.4 OrderItem

Recommended Implementation Type: Data Structure

Used by: GC_BrokeredAccessRequestType

This data structure contains the specification of a single order item (i.e. e. the product that is ordered and that is to be delivered):

a) productId, which is the identifier of the ordered product.

b) productPrice, which is the price of the product.

c) productDeliveryOptions, which contains delivery options for the product.

d) processingOptions, which specifies the processing options that are to be applied on the product before delivery.

e) sceneSelectionOptions, which specifies the selection of the scene from the whole product that is to be delivered.

C.4.5 OrderSpecification

Recommended Implementation Type: Data Structure

Used By: BrokeredAccessRequest

The specification of the order request as provided as input by the client if BrokeredAccessRequestType = orderEstimate or OrderQuoteAndSubmit.

The structure contains the following information about the product specification:

a) orderCentreID – identifies the order center at which the order will be performed

b) orderPrice –the price for the whole order

c) orderDeliveryDate - the latest date at which the order can be expected to be delivered to the user.

d) orderCancellationDate – the latest date at which the user can cancel the order.

e) deliveryMethod – how the order will be delivered to the user: e-mail, ftp or mail.

f) package – contains the definition of how the packages which compose the order

C.4.6 OrderStatus

Recommended Implementation Type: Code_List

Used By: BrokeredAccessResponse

OrderStatus is a code list for identifying the status of an order. Valid values for this type are:

Table C.2 — Order Status Codes

Value	Explanation
orderBeingEstimated	the order is currently being estimated by the target order handling system. An Estimate is an approximation only.
orderEstimated	indicates that the order has been successfully validated and that an estimate is provided.
orderBeingQuoted	the order is currently being quoted by the target order handling system. A Quote shall be considered contractually binding.
orderBeingProcessed	the order is currently being processed by the target order handling system.
orderCompleted	processing of order has been completed.
orderNotValid	the order has not been successfully validated.
orderCancelled	the order has been cancelled

C.4.7 PackageSpecification

Recommended Implementation Type: Data Structure

Used By: BrokeredAccessRequest, OrderSpecification, PackagingType

The specification of a single package or multiple packages.

The structure contains the following information about the packaging order:

a) packageId – the identifier of the ordered package

b) packagePrice –the price for the package

c) package – the detailed information concerning the specification of package. (See packagingType)

d) packageMedium –the medium on which the package will be delivered to a user.

e) packageSize – the size of the package in kilobytes.

C.4.8 PackagingType

Recommended Implementation Type: Code List

Used By: PackageSpecification, BrokeredAccessRequest

The specification of the packaging method used to deliver an order to a user.

a) predefinedPackage: A package predefined by the given catalogue service

b) adhocPackage: A package constructed of OrderItems to fulfill a particular order

C.4.9 PaymentMethod

Recommended Implementation Type: Code_List

Used By: UserInformation

This code list contains the payment methods for an order secured through using a Access operation. The supported methods are the following:

a) credit

b) cash

c) purchaseOrder

C.4.10 QueryScope

Recommended Implementation Type: Code_List

Used By: QueryRequest

QueryScope is a code list describing the size of the search space for a query. Current valid values for this type are:

distributed

local

The Reference Architecture for the OGC Catalogue allows for catalogue requests to be distributed to multiple catalogs. The architecture allows for a Catalogue to accept a request from a client and distribute the request to other Catalogs. For the OGC Catalogue Service, Distributed Catalogue Searching is defined as a service that involves services of multiple Catalogue Servers, in addition to the primary client-server interaction. A catalogue server may be able to perform Distributed Searching by propagating secondary catalogue service requests to other catalogue servers. See Annex D for more explanation.

C.4.11 Status

Recommended Implementation Type: Code_List

Used By: BrokeredAccessResponse

Status is a code list for representing the current status of a resource or request. The valid values for this type are the following:

a) success: the request has been processed without error.

b) successResultsAvailable: the request has been processed without error and outputs of the processing can be retrieved.

c) processingNormal: the requested operations have begun but are not completed. No errors have been identified.

d) processingQueued: the requested operations have begun but are not completed. No errors have been identified. The processing has been temporally suspended and will resume when other processing has been completed.

e) processingPausedOrSuspended: the requested operations have begun but are not completed. No errors have been identified. The processing has been temporally suspended and will resume when triggered by an external event.

f) failure: the request could not be completed due to errors being encountered. On a best effort basis the server has returned to the state prior to the request.

g) failureAccessDenied : the request could not be completed because the privileges of the client did not permit the operation. On a best effort basis the server has returned to the state prior to the request.

C.4.12 StatusUpdateType

Recommended Implementation Type: Code List

Used By: OrderStatusUpdateType

This parameter defines how the user requesting the order desires to be kept informed about the order processing.

a) manual: The user performs the status request using the Catalogue Interface

b) automatic: The OHS filling the order provides status updates for the user via email

C.4.13 UserInformation

Recommended Implementation Type: Data Structure

Used By: BrokeredAccessRequest

This parameter type is a data structure used to provide information about the user.

a) userName: (type = Character String) – name of the user

b) userAddress: (type = CharacterString) – billing, home or delivery address of user

c) phoneNumber: (type = CharacterString) – home or office phone number for user

d) faxNumber: (type = CharacterString) – home or office fax number for user

e) emailAddress: (type = CharacterString) – e-mail address for the user

f) NetAddress: (type = CharacterString) – Address of the users' primary computer.

g) PaymentMethod: (type = PaymentMethod) – defines the payment method

Annex D
(informative)

Sample CSW capabilities document

```
<csw:Capabilities
   version="2.0.1"
   updateSequence="0"
   xmlns:ows="http://www.opengis.net/ows"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:csw="http://www.opengis.net/csw"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://www.opengis.net/ows
       http://www.pvretano.com/schemas/common/0.2.0/owsCommon.xsd
       http://www.opengis.net/ogc
       http://www.pvretano.com/schemas/filter/1.0.0/filterCapabilities.xsd
       http://www.opengis.net/csw
       http://www.pvretano.com/schemas/csw/0.2.0/CSW-discovery.xsd
       http://www.w3.org/1999/xlink
       http://www.pvretano.com/schemas/gml/3.0.0/xlink/xlinks.xsd">
   <ows:ServiceIdentification>
      <ows:ServiceType>CSW</ows:ServiceType>
      <ows:ServiceTypeVersion>0.7.8</ows:ServiceTypeVersion>
      <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
      <ows:Title>CubeWerx CSW</ows:Title>
      <ows:Abstract>
         A catalogue service that conforms to the HTTP protocol
         binding of the OpenGIS Catalogue Service specification
         version 2.0.0.
      </ows:Abstract>
      <ows:Keywords>
         <ows:Keyword>CSW</ows:Keyword>
         <ows:Keyword>CubeWerx</ows:Keyword>
         <ows:Keyword>geospatial</ows:Keyword>
         <ows:Keyword>catalogue</ows:Keyword>
      </ows:Keywords>
      <ows:Fees>NONE</ows:Fees>
      <ows:AccessConstraints>NONE</ows:AccessConstraints>
   </ows:ServiceIdentification>
   <ows:ServiceProvider>
      <ows:ProviderName>CubeWerx Inc.</ows:ProviderName>
      <ows:ProviderSite xlink:href="http://www.cubewerx.com"/>
      <ows:ServiceContact>
         <ows:IndividualName>Panagiotis A. Vretanos</ows:IndividualName>
         <ows:PositionName>Senior Developer</ows:PositionName>
         <ows:ContactInfo>
            <ows:Phone>
               <ows:Voice>+1-819-771-8303</ows:Voice>
               <ows:Facsimile>+1-819-771-8303</ows:Facsimile>
            </ows:Phone>
            <ows:Address>
               <ows:DeliveryPoint>200 rue Montcalm</ows:DeliveryPoint>
               <ows:DeliveryPoint>Suite R-13</ows:DeliveryPoint>
               <ows:City>Gatineau</ows:City>
               <ows:AdministrativeArea>Quebec</ows:AdministrativeArea>
               <ows:PostalCode>J8Y 3B5</ows:PostalCode>
               <ows:Country>CANADA</ows:Country>
               <ows:ElectronicMailAddress>
               pvretano@cubwerx.com
               </ows:ElectronicMailAddress>
            </ows:Address>
            <ows:OnlineResource xlink:href="mailto:pvretano@cubewerx.com"/>
         </ows:ContactInfo>
      </ows:ServiceContact>
   </ows:ServiceProvider>
   <ows:OperationsMetadata>
      <ows:Operation name="GetCapabilities">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsw.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsw.cgi"/>
            </ows:HTTP>
         </ows:DCP>
      </ows:Operation>
      <ows:Operation name="DescribeRecord">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsw.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsw.cgi"/>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="typeName">
            <ows:Value>csw:Record</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="outputFormat">
            <ows:Value>application/xml</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="schemaLanguage">
            <ows:Value>XMLSCHEMA</ows:Value>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="GetRecords">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsw.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsw.cgi"/>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="TypeName">
            <ows:Value>csw:Record</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="outputFormat">
           <ows:Value>application/xml</ows:Value>
           <ows:Value>text/html</ows:Value>
           <ows:Value>text/plain</ows:Value>
         </ows:Parameter>
```

```
         <ows:Parameter name="outputSchema">
            <ows:Value>csw:Record</ows:DefaultValue>
         </ows:Parameter>
         <ows:Parameter name="resultType">
            <ows:Value>hits</ows:DefaultValue>
            <ows:Value>results</ows:Value>
            <ows:Value>validate</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="ElementSetName">
            <ows:Value>brief</ows:Value>
            <ows:Value>summary</ows:Value>
            <ows:Value>full</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="CONSTRAINTLANGUAGE">
            <ows:Value>Filter</ows:Value>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="GetRecordById">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsg.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsg.cgi"/>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="ElementSetName">
            <ows:Value>brief</ows:Value>
            <ows:Value>summary</ows:Value>
            <ows:Value>full</ows:Value>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="GetDomain">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsg.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsg.cgi"/>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="ParameterName">
            <ows:Value>GetRecords.resultType</ows:Value>
            <ows:Value>GetRecords.outputFormat</ows:Value>
            <ows:Value>GetRecords.outputRecType</ows:Value>
            <ows:Value>GetRecords.typeNames</ows:Value>
            <ows:Value>GetRecords.ElementSetName</ows:Value>
            <ows:Value>GetRecords.ElementName</ows:Value>
            <ows:Value>GetRecords.CONSTRAINTLANGUAGE</ows:Value>
            <ows:Value>GetRecordById.ElementSetName</ows:Value>
            <ows:Value>DescribeRecord.typeName</ows:Value>
            <ows:Value>DescribeRecord.schemaLanguage</ows:Value>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="Harvest">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsg.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsg.cgi"/>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="ResourceType">
            <ows:Value>csw:Record</ows:DefaultValue>
            <ows:Value>fgdc:metadata</ows:Value>
            <ows:Value>wfs:Capabilities</ows:Value>
            <ows:Value>wms:Capabilities</ows:Value>
         </ows:Parameter>
         <ows:Parameter name="ResourceFormat">
            <ows:Value>application/xml</ows:DefaultValue>
            <ows:Value>text/plain</ows:Value>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="Transaction">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://www.cubewerx.com/cwcsg.cgi?"/>
               <ows:Post xlink:href="http://www.cubewerx.com/cwcsg.cgi"/>
            </ows:HTTP>
         </ows:DCP>
      </ows:Operation>
      <ows:Parameter name="service">
         <ows:Value>CSW</ows:DefaultValue>
      </ows:Parameter>
      <ows:Parameter name="version">
         <ows:Value>2.0.1</ows:DefaultValue>
         <ows:Value>2.0.0</ows:Value>
      </ows:Parameter>
      <ows:ExtendedCapabilities></ows:ExtendedCapabilities>
   </ows:OperationsMetadata>
   <ogc:Filter_Capabilities>
      <ogc:Spatial_Capabilities>
         <ogc:Spatial_Operators>
            <ogc:BBOX/>
            <ogc:Equals/>
            <ogc:Intersect/>
            <ogc:Touches/>
            <ogc:Crosses/>
            <ogc:Contains/>
            <ogc:Overlaps/>
         </ogc:Spatial_Operators>
      </ogc:Spatial_Capabilities>
      <ogc:Scalar_Capabilities>
         <ogc:Logical_Operators/>
         <ogc:Comparison_Operators>
            <ogc:Simple_Comparisons/>
            <ogc:Like/>
            <ogc:Between/>
            <ogc:NullCheck/>
         </ogc:Comparison_Operators>
         <ogc:Arithmetic_Operators>
            <ogc:Simple_Arithmetic/>
            <ogc:Functions>
               <ogc:Function_Names>
                  <ogc:Function_Name nArgs="1">COUNT</ogc:Function_Name>
                  <ogc:Function_Name nArgs="1">DISTINCT</ogc:Function_Name>
                  <ogc:Function_Name nArgs="1">MIN</ogc:Function_Name>
                  <ogc:Function_Name nArgs="1">MAX</ogc:Function_Name>
                  <ogc:Function_Name nArgs="1">UPPER</ogc:Function_Name>
               </ogc:Function_Names>
            </ogc:Functions>
         </ogc:Arithmetic_Operators>
      </ogc:Scalar_Capabilities>
   </ogc:Filter_Capabilities>
</csw:Capabilities>
```

Annex E
(normative)

Technical corrigendum 1

E.1 All subclauses

Replace every occurrence of "text/xml" with "application/xml".

E.2 Subclause 6.3.2

In Table 1 and the title for Table 2 change "Envelope" to "BoundingBox".

In Table 1 change the text in the "Data type" column for the following elements:

Format – Codelist: application/xml, text/html, text/plain
Type – Codelist: Dataset, DatasetCollection, Service

E.3 Subclause 10.3.1

Change the first sentence to:

Only the GET and POST methods are employed in the HTTP binding. Table 53 summarizes the allowed HTTP method bindings and request data encodings for all CSW requests; optional method bindings and data encodings are enclosed in parentheses.

Replace Table 53 with the following table:

Table 53 — HTTP method bindings

Request	HTTP method binding(s)	Data encoding(s)[a,b]
GetCapabilities	GET (POST)	KVP (XML)
DescribeRecord	POST (GET)	XML (KVP)
GetDomain	POST (GET)	XML (KVP)
GetRecords	POST (GET)	XML (KVP)
GetRecordById	GET (POST)	KVP (XML)
Harvest	POST	XML (KVP)
Transaction	POST	XML

[a] XML = application/xml using POST (with a charset parameter if necessary—UTF-8 is strongly recommended)

[b] KVP = URL-encoded key/value pairs using GET or application/x-www-form-urlencoded using POST

E.4 Subclause 10.3.4

Replace the second item in the list of predefined predicate languages with:

b) **FILTER** is an XML encoding of the BNF grammar and is normatively defined in the *Filter Encoding Implementation Specification*, version 1.1.0 [OGC 04-095]. All CSW implementations are required to support this filter syntax.

Replace the schema fragment with the following:

```
<xsd:complexType name="QueryConstraintType" id="QueryConstraintType">
  <xsd:choice>
    <xsd:element ref="ogc:Filter"/>
    <xsd:element name="CqlText" type="xsd:string"/>
  </xsd:choice>
    <xsd:attribute name="version" type="xsd:string" use="required">
  </xsd:attribute>
</xsd:complexType>
```

E.5 Subclause 10.3.6

Append the following sentence to the end of the second paragraph:

Note that the parameter names in all KVP encodings must be handled in a **case insensitive** manner.

E.6 Subclause 10.5.2

Change the first sentence to:

The GetCapabilities operation request is defined in Subclause 7.2 of the *OGC Web Services Common Specification 1.0* [OGC 05-008].

Change the last sentence of the second paragraph to:

The common service metadata elements that may be included in a Capabilities document are specified in Subclause 7.4 of OGC 05-008; a catalogue service that implements the CSW binding may also include the elements listed in Table 57. An application profile may introduce additional service information items as needed by extending the csw:CapabilitiesType definition.

In Table 57 replace the text in the "Meaning" column with the following text:

A Filter_Capabilities section must be included in the service metadata to describe which elements of the predicate language are supported. All CSW implementations must support at least the following filter operators:

- logical operators: And, Or, Not
- comparison operators: PropertyIsEqualTo, PropertyIsNotEqualTo, PropertyIsLessThan, PropertyIsGreaterThan, PropertyIsLessThanOrEqualTo, PropertyIsGreaterThanOrEqualTo, PropertyIsLike
- spatial operators: BBOX

E.7 Subclause 10.5.4

Change the first paragraph to:

The OperationsMetadata element shall list all operations implemented by the service, as described in Subclause 7.4.5 of OGC 05-008. An application profile may restrict the <ExtendedCapabilities> element to provide additional computational metadata (e.g., WSDL service descriptions, OWL-S resource definitions).

E.8 Subclause 10.5.5

Move the sample capabilities document to a new informative Annex D.

E.9 Subclause 10.6.2

In Table 61 change the "Optionality and Use" for the NAMESPACE parameter to:

One (Mandatory) [b.] Include declarations for each namespace used in a TypeName.

E.10 Subclause 10.7.1

Replace the last sentence in the final paragraph with:

It is entirely possible that a catalogue may not be able to determine anything about the values of a property or request parameter beyond the basic type; in this case only a type reference or a type description will be returned.

E.11 Subclause 10.7.3

Replace the schema fragment with the following:

```
<xsd:element name="GetDomain" type="csw:GetDomainType"/>
<xsd:complexType name="GetDomainType">
  <xsd:complexContent>
    <xsd:extension base="csw:RequestBaseType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="PropertyName" type="xsd:anyURI" />
          <xsd:element name="ParameterName" type="xsd:anyURI" />
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

E.12 Subclause 10.7.4.1

Append the following sentence to the end of the first paragraph:

The PropertyName value may be specified using an absolute or a relative URI; the precise syntax or permissible values are defined in an application profile.

E.13 Subclause 10.8.2

In Table 64 update the column entries for the resultType parameter as follows:

resultType	CodeList with allowed values: "hits", "results", "validate"	Zero or one (Optional) Default value is "hits"	resultType

In Table 64 change the default value of the outputSchema parameter to "csw:Record".

E.14 Subclause 10.8.3

Replace the schema fragment with the following:

```
<xsd:element name="GetRecords" type="csw:GetRecordsType"
             id="GetRecords"/>
<xsd:complexType name="GetRecordsType" id="GetRecordsType">
  <xsd:complexContent>
    <xsd:extension base="csw:RequestBaseType">
      <xsd:sequence>
        <xsd:element name="DistributedSearch"
                     type="csw:DistributedSearchType"
                     minOccurs="0"/>
        <xsd:element name="ResponseHandler"
                     type="xsd:anyURI"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:choice>
          <xsd:element ref="csw:AbstractQuery"/>
          <xsd:any processContents="strict"
                   namespace="##other" />
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="requestId" type="xsd:anyURI"
                     use="optional" />
      <xsd:attribute name="resultType" type="csw:ResultType"
                     use="optional" default="hits"/>
      <xsd:attributeGroup ref="csw:BasicRetrievalOptions"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

E.15 Subclause 10.8.4.2

Replace the entire text of the subclause with the following text:

The **resultType** parameter may have the value *"hits"*, *"results",* or *"validate"*; the value determines whether the catalogue service returns just a summary of the result set, includes one or more records from the result set, or validates the request message and processes it asynchronously.

If the **resultType** parameter is set to *"*hits*"*, the catalogue service shall return a <**GetRecordsResponse**> element containing an empty <SearchResults> element that indicates the estimated size of the result set. Optional attributes may or may not be set accordingly.

If the **resultType** parameter is set to "results", the catalogue service must include any matching records within the <SearchResults> element, up to the maximum number of records specified in the request.

If the **resultType** parameter is set to “validate”, the catalogue service must validate the request and return an <Acknowledgement> message if validation succeeds; a <ServiceExceptionReport> is returned if validation fails. If the catalogue supports asynchronous query processing, the acknowledgement response must include a RequestId element that may be subsequently used to retrieve the result set when processing is complete.

E.16 Subclause 10.11.3.2

Replace the schema fragment with the following:

```
<xsd:complexType name="InsertType" id="InsertType">
  <xsd:sequence>
    <xsd:any processContents="strict" namespace="##other"
             maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

E.17 Subclause 10.11.3.3

Replace the UpdateType definition in the schema fragment with the following:

```
<xsd:complexType name="UpdateType" id="UpdateType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:any processContents="strict" namespace="##other" />
      <xsd:sequence>
        <xsd:element ref="csw:RecordProperty" maxOccurs="unbounded"/>
        <xsd:element ref="csw:Constraint"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

Change the second sentence in the paragraph following the schema fragment to:

If the <**csw:Update**> element contains a child element declared in some other namespace (i.e., **not** “http://www.opengis.net/cat/csw”), then the corresponding record in the catalogue shall be replaced. The record type must be defined in an application profile.

E.18 Subclause 10.11.3.4

Replace the schema fragment with the following type definition:

```
<xsd:complexType name="DeleteType" id="DeleteType">
  <xsd:sequence>
    <xsd:element ref="csw:Constraint" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="typeName" type="xsd:anyURI" use="optional"/>
  <xsd:attribute name="handle" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

Change the first sentence in the paragraph following the schema fragment to:

The <**Delete**> element contains a <**csw:Constraint**> element (see Subclause 10.3.7) that identifies a set of records that are to be deleted from the catalogue.

E.19 Subclause 6.2.3

Replace the Filter example with the following:

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
  <ClassifiedAs>
    <EntryPoint>RECORD/<EntryPoint>
      <KeyName>/GeoClass/Continent/Country/State</KeyName>
      <KeyValue>/GeoClass/NorthAmerica/%/Ontario</KeyValue>
  </ClassifiedAs>
</ogc:Filter>
```

E.20 Subclause 6.2.5.3

Replace the Filter example with the following:

```
<Filter xmlns="http://http://www.opengis.net/ogc"
        xmlns:foo="http://foo/">
  <And>
    <Or>
      <PropertyIsEqualTo>
        <PropertyName>foo:FIELD1</PropertyName>
        <Literal>10</Literal>
      </PropertyIsEqualTo>
      <PropertyIsEqualTo>
         <PropertyName>foo:FIELD1</PropertyName>
         <Literal>20</Literal>
      </PropertyIsEqualTo>
    </Or>
    <PropertyIsEqualTo>
      <PropertyName>foo:STATUS</PropertyName>
      <Literal>VALID</Literal>
    </PropertyIsEqualTo>
  </And>
</Filter>
```

E.21 Subclause 6.2.5.4

Replace the two Filter examples with the following:

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
   <ogc:PropertyIsLessThan>
      <ogc:PropertyName>cloudcover</ogc:PropertyName>
      <ogc:Literal>5</ogc:Literal>
   </ogc:PropertyIsLessThan>
</ogc:Filter>

[ ... ]

<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
   <ogc:Or>
      <ogc:PropertyIsLessThan>
         <ogc:PropertyName>cloudcover</ogc:PropertyName>
         <ogc:Literal>50</ogc:Literal>
      </ogc:PropertyIsLessThan>
      <ogc:PropertyValueDoesNotExist>
```

```
         <ogc:PropertyName>cloudcover</ogc:PropertyName>
      </ogc:PropertValueDoesNotExist>
   </ogc:Or>
</ogc:Filter>
```

E.22 Subclause 6.3.3

Declare a namespace for the “cat” prefix in example record, and delete the “>” character after the iso19115TopicCategory namespace prefix declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
<cat:Record
  xmlns:cat="http://www.opengis.net/cat"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dct="http://purl.org/dc/terms/"
  xmlns:csw="http://www.opengis.net/csw"
  xmlns:iso19115TopicCategory=”http://www.isotc211.org/”
  xmlns:dcmiBox=”http://dublincore.org/documents/2000/07/11/dcmi-box/“>
  <dc:creator>U.S. Geological Survey</dc:creator>
[...]
```

Replace the <dc:coverage> element and content with the following XML fragment beginning with <dct:spatial>:

```
<dct:spatial>
  <Box projection="EPSG:4326" name="Geographic">
    <northlimit>34.353</northlimit>
    <eastlimit>-96.223</eastlimit>
    <southlimit>28.229</southlimit>
    <westlimit>-108.44</westlimit>
  </Box>
</dct:spatial>
```

E.23 Subclause 10.5.3

Delete the “Contents” section name reference and row from Table 58.

In Table 58, delete the last sentence in the “Contents” column for Filter_Capabilities (since it is a required element).

E.24 Annex D

Rename the <ows:DefaultValue> elements with <ows:Value> in the “outputFormat” parameter section of the GetRecords operation in the capabilities response example:

```
<ows:Parameter name="outputFormat">
  <ows:Value>application/xml</ows:Value>
  <ows:Value>text/html</ows:Value>
  <ows:Value>text/plain</ows:Value>
</ows:Parameter>
```

Rename the <ows:DefaultValue> elements with <ows:Value> in the "outputSchema" parameter section and replace the "ows:record" with "csw:Record":

```
<ows:Parameter name="outputSchema">
      <ows:Value>csw:Record</ows:Value>
</ows:Parameter>
```

Rename the <ows:DefaultValue> elements with <ows:Value> in the "resultType" parameter section:

```
<ows:Parameter name="resultType">
      <ows:Value>hits</ows: Value>
      <ows:Value>results</ows:Value>
      <ows:Value>validate</ows:Value>
</ows:Parameter>
```

Rename the <ows:DefaultValue> elements with <ows:Value> in the "ResourceType" parameter section of the Harvest operation; replace the "ows:record" with "csw:Record"

```
<ows:Parameter name="ResourceType">
      <ows:Value>csw:Record</ows:DefaultValue>
      <ows:Value>fgdc:metadata</ows:Value>
```

Rename the <ows:DefaultValue> elements with <ows:Value> in the "ResourceFormat" parameter section:

```
<ows:Parameter name="ResourceFormat">
      <ows:Value>application/xml</ows:Value>
      <ows:Value>text/plain</ows:Value>
</ows:Parameter>
```

Rename the <ows:DefaultValue> elements with <ows:Value> in the "service" and "version" parameter sections of the OperationsMetadata section:

```
<ows:Parameter name="service">
      <ows:Value>CSW</ows:Value>
</ows:Parameter>
<ows:Parameter name="version">
      <ows:Value>2.0.1</ows:Value>
      <ows:Value>2.0.0</ows:Value>
</ows:Parameter>
```

Remove the <csw:Contents> element from the capabilities response example.

E.25 Subclause 10.6.4.2

Replace the last sentence in the second paragraph with the following:

For XML-encoded **DescribeRecord** requests, the namespace declarations are specified using the targetNamespace attribute of the TypeName element.

E.26 Subclause 10.6.6

Replace the KVP encoded example with the following:

```
http://www.someserver.com/csw/csw.cgi?request=DescribeRecord&version=2.0.0&
outputFormat=application/xml&schemaLanguage=XMLSCHEMA&typeName=csw:Record&n
amespace=csw:http://www.opengis.org/cat/csw
```

“csw:record” was replaced with “csw:Record” and the “http://www.opengis.org/csw” namespace was replaced with “http://www.opengis.org/cat/csw”.

Replace the DescribeRecord XML encoded example with the following:

```
<csw:DescribeRecord version="2.0.1"
  outputFormat="application/xml"
  schemaLanguage="http://www.w3.org/2001/XMLSchema">
  <csw:TypeName
    targetNamespace="http://www.opengis.org/cat/csw">Record</csw:TypeName>
</csw:DescribeRecord>
```

A attribute “targetNamespace” attribute was added to the **<csw:TypeName>** element – this is mandatory – and “csw:record” was replaced with the bareword “Record”.

E.27 Subclause 10.8.4.6

Replace the subclause reference at the end of the paragraph from “10.8.4.3 “ to “10.8.4.2”.

E.28 Subclause 10.8.6

Replace the KVP encoded example with the following:

```
http://www.someserver.com/csw/csw.cgi?request=GetRecords&version=2.0.0&out
putFormat=application/xml&outputSchema=csw:Record&namespace=csw:http://www
.opengis.org/cat/csw&ResponseHandler="mailto:pvretano@cubewerx.com"&typeNa
me=csw:Record&elementSetName=brief&constraintlanguage=CQLTEXT&constrain="c
sw:AnyText Like '%polution%'"
```

Two occurrences of “csw:record” were replaced with “csw:Record” and the namespace “http://www.opengis.org/csw” was replaced with “http://www.opengis.org/cat/csw”. The filter property name “csw:AnyTextLike” was replaced with “csw:AnyText Like” (space added).

Replace the XML encoded example with the following:

```
<csw:GetRecords version="2.0.1"
      outputFormat="application/xml"
      outputSchema="csw:Record">
   <csw:ResponseHandler>
ftp://www.myserver.com/pub/MyQuery_Resp.xml</csw:ResponseHandler>
   <csw:Query typeName="csw:Record">
      <csw:ElementSetName>brief</csw:ElementSetName>
      <csw:Constraint>
         <ogc:Filter>
            <ogc:PropertyIsLike wildCard="%" singleChar="_" escape="\">
               <ogc:PropertyName>
                  /csw:Record/csw:AnyText
               </ogc:PropertyName>
               <ogc:Literal>%polution%</ogc:Literal>
            </ogc:PropertyIsLike>
         </ogc:Filter>
      </csw:Constraint>
   </csw:Query>
```

```
</csw:GetRecords>
```

Three occurrences of “csw:record” were replaced with “csw:Record”.

E.29 Subclause 10.9.3

Replace the schema fragment with the following:

```
<xsd:element name="GetRecordById" type="csw:GetRecordByIdType"/>
<xsd:complexType name="GetRecordByIdType">
   <xsd:complexContent>
      <xsd:extension base="csw:RequestBaseType">
         <xsd:sequence>
            <xsd:element name="Id" type="xsd:anyURI"
                         maxOccurs=”unbounded”/>
            <xsd:element ref="csw:ElementSetName" minOccurs="0"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

E.30 Subclause 10.9.6

Replace the KVP encoded request example to the following:

```
http://www.someserver.com/csw/csw.cgi?request=GetRecordById&version=2.0.0&i
d=REC-10,REC-11,REC-12
```

In the XML example change the value of the version attribute to “2.0.1”.

Bibliography

[1] ISO/IEC 8825:1990, Information technology -- Open Systems Interconnection -- Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

[2] ISO/IEC TR 10000-1:1998, Information technology – Framework and taxonomy of International Standardised Profiles – Part 1: General principles and documentation framework.

[3] ISO/IEC TR 10000-2:1998, Information technology – Framework and taxonomy of International Standardised Profiles – Part 2: Principles and Taxonomy for OSI Profiles

[4] ISO 19101:2002, Geographic information -- Reference model

[5] ISO 19103 (DTS), Geographic information - Conceptual schema language, (Draft Technical Specification)

[6] ISO 19106:2002 (DIS), Geographic information - Profiles

[7] ISO 19108:2002, Geographic information - Temporal schema

[8] ISO 19109:2002 (DIS), Geographic information - Rules for application schema

[9] ISO 19110:2001 (DIS), Geographic information - Methodology for feature cataloguing

[10] ISO 19113:2002, Geographic information - Quality principles

[11] ISO 19114:2001, (DIS) Geographic information - Quality evaluation procedures

[12] ISO 19118:2002, (DIS) Geographic information - Encoding

[13] ISO 23950:1998, Information and documentation -- Information retrieval (Z39.50) -- Application service definition and protocol specification