HARMOSEARCH
the future of information services

FP7-SME-1
Project no. 262289

HARMOSEARCH

Harmonised Semantic Meta-Search in
Distributed Heterogeneous Databases

SEVENTH FRAMEWORK PROGRAMME

D4.1

Semantic query – Query language specification

Due date of deliverable: 2011-06-30
Actual submission date: 2011-06-30

Start date of project: 2010-12-01 Duration: 24 month

Project funded by the European Commission within the Seventh Framework Programme

Dissemination Level		
PU	Public	**X**
PP	Restricted to other participants (including the Commission Services)	
RE	Restricted to a group specified by the Consortium (including the Commission Services)	
CO	Confidential, only for members of the Consortium (including the Commission Services)	

HARMOSEARCH
the future of information services

PROJECT ACRONYM: **HARMOSEARCH**

Project Title: Harmonised Semantic Meta-Search in Distributed Heterogeneous Databases

Grant Agreement: 262289

Starting date: December 2010 **Ending date:** November 2012

Deliverable Number: D4.1, Version 1.0

Title of the Deliverable: Semantic query – Query language specification

Lead Beneficiary: CPR

Task/WP related to the Deliverable: WP 4, Task 4.1

Type (Internal or Restricted or Public): Public

Author(s): Adriano Venturini, Albert Rainer, Alessandro Forti, Christoph Herzog, Claudio Prandoni, Federico Galeazzi, Sabine Schneider

Partner(s) Contributing: eCTRL, CPR, TU-WIEN

Contractual Date of Delivery to the CEC: June 30, 2011

Actual Date of Delivery to the CEC: June 30, 2011

PROJECT CO-ORDINATOR

Company name: [X+O]

Name of representative: Manfred Hackl

Address: Hamburgerstrasse 10/7, A-1050 Vienna, Austria

Phone number: +43-676-842755-100

Fax number: +43-676-842755-599

E-mail: manfred.hackl@xpluso.com

Project WEB site address: www.harmosearch.org

TABLE OF CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

In order to allow searching data in a network of data bases with different query languages in use, not only data but also queries need to be transformed. A HarmoSearch query language needs therefore to be identified to provide the necessary specifications on how to encode a query made through HarmoSearch. This will allow to translate queries described in the language of the organization querying the system into the one required for the organizations providing data.

Starting from the reference model defined in D3.1, this document presents an evaluation of the most prominent query languages covering a broad range of queries – such as XQuery, SQL, SML, etc. The outcome is the definition and the specification of the HarmoSearch query language, which is based on the query language that best fulfils the requirements.

1.2 RELATIONSHIP WITH OTHER DOCUMENTS

Inputs to this document come from the deliverable D3.1 Ontology for the Query Model, which defines the technical requirements for the development of the query language and the reference model which allows to represent a Harmonise query in an abstract form.

The query language described in this document will be used for the implementation of the Query Processor (D4.2) and of the Metasearch Application (D4.3).

1.3 STRUCTURE OF THE DOCUMENT

This document is structured in the following main sections:

- Query languages analysis and selection, which presents an overview of the most important query languages, analysing them and choosing the most appropriate to be reused and adapted to become the HarmoSearch query language
- HarmoSearch query reference model, which reports the HarmoSearch query reference model (defined in D3.1)
- HarmoSearch query language, which presents and describes the query language that will be used in the HarmoSearch project

2 QUERY LANGUAGES ANALYSIS

2.1 QUERY LANGUAGES OVERVIEW

2.1.1 Query by Example

"Query by example" (QBE) is a database query language for relational databases developed by IBM in the 1970s in parallel to what has become SQL. It is different from SQL, and from most other database query languages, in having a graphical user interface that allows users to write queries by creating example tables on the screen. It is easy to use, but at the same time, best suited for not too complex query and queries that involve a few tables.

In this case, the query is not represented by text like in SQL, but by a table structure. Thus, it is often also called a graphical query language. The reason why this could be of interest in the HarmoSearch case is that in this case the query language could be standardized for the whole system, independent of the kind of database each participant is actually using. In this way, how to create queries and interpret them could be unified, while each participant could still keep its own query language.

Anyway it is important to remark that every QBE query can be expressed using another language such as SQL, XQuery, etc.

As already said, a user writes queries by selecting *skeleton tables* and filling them with *example rows.* An example row consists of constants and example elements which are really *domain variables*. The domain of a variable is determined by the column in which it appears, and variable symbols are prefixed with underscore ("_") to distinguish them from constants. Constants, including strings, appear unquoted, in contrast to SQL. The fields that should appear in the answer are specified by using the command P., which stands for *print*. The fields containing this command are analogous to the *target-list* in the SELECT clause of an SQL query.

That said, if we use a schema like this:

`Sailors(`*`sid:`* `integer, `*`sname:`* `string, `*`rating:`* `integer, `*`age:`* `real)`

`Boats(`*`bid:`* `integer, `*`bname:`* `string, `*`color:`* `string)`

`Reserves(`*`sid:`* `integer, `*`bid:`* `integer, `*`day:`* `dates)`

where the key fields are underlined, and the type of each field is listed after the field name, an example of QBE to print names and ages of all sailors with rating = 10, is:

Sailors	sid	sname	rating	age
		P._N	10	P._A

Comparison operations (<, >,<=,>=, ┐) are allowed as well as ordering, using the notation .AO(i) (ascending order, priority *i*), or .DO(j), i.e. Descending Order, priority *j*. Therefore to print all fields for sailors with *rating* > 8, in ascending order by (*rating, age*) the query will look like:

Sailors	sid	sname	rating	age
P.			AO(1) > 8	AO(2)

Like SQL, QBE supports the aggregate operations AVG., COUNT., MAX., MIN., and SUM.

QBE supports grouping, by the use of G. command.

The SQL clause *DISTINCT* is represented by specifying "UNQ." in the table, under the relation name.

In case of multiple relations, an example of join query could be:

Sailors	sids	sname	rating	age
	_Id	P.		>25

Reserves	sid	bid	day
	_Id	_B	'8/4/96'

Boats	bid	bname	color
	_B	Interlake	P.

Which will return the colors of Interlake boats reserved by sailors who've reserved a boat for 8/24/96 and are older than 25.

A condition box can be introduced to:

- Express a condition involving two or more columns
- Express a condition involving an aggregate operation on a group
- Express conditions involving the AND and OR operators. i.e.

Sailors	Sid	Sname	Rating	Age	Conditions
		P.		_A	_A<20 OR 30 < _A

Indeed, it is possible to use queries involving AND and OR operator without using the condition column.

Sailors	Sid	Sname	Rating	age
		P.		<30
		P.		>20

Insertion, deletion, and modification of a tuple are specified through the commands I., D., and U., respectively. For example:

Sailors	Sid	Sname	Rating	Age
I.	74	John	7	23

2.1.2 SQL

SQL often referred to as Structured Query Language, is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon relational algebra and calculus. Its scope includes data insert, query, update and delete, schema creation and modification, and data access control.

The SQL language is sub-divided into several language elements, including:

- Clauses, which are constituent components of statements and queries. (In some cases, these are optional.)
- Expressions, which can produce either scalar values or tables consisting of columns and rows of data.
- Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) or Boolean (true/false/unknown) truth values and which are used to limit the effects of statements and queries, or to change program flow.
- Queries, which retrieve the data based on specific criteria. This is the most important element of SQL.
- Statements, which may have a persistent effect on schemata and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
    - SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

The most common operation in SQL is the query, which is performed with the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions. Standard SELECT statements have no persistent effects on the database. Some non-standard implementations of SELECT can have persistent effects, such as the SELECT INTO syntax that exists in some databases.

Queries allow the user to describe desired data, leaving the database management system (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to be included in the final result immediately following the SELECT keyword. An asterisk ("*") can also be used to specify that the query should return all columns of the queried tables. SELECT is the most complex statement in SQL, with optional keywords and clauses that include:

- The FROM clause which indicates the table(s) from which data is to be retrieved. The FROM clause can include optional JOIN subclauses to specify the rules for joining tables.
- The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.

- The GROUP BY clause is used to project rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.
- The HAVING clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.
- The ORDER BY clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

The SQL language operators belong to 4 categories:

- Comparison Operators, to determine the relationship among data, not only numerical data but also string operands.
- Arithmetic Operators, to perform calculation within a search, for instance in this query: Select Salary, (salary + 100) from employees
- Conditional Operators, consisting only of the WHERE clause, used to border the search.
- Logical Operators: the AND, OR, NOT, XOR operators, possibly used more than once in the same query.

The following is an example of a SELECT query that returns a list of expensive books. The query retrieves all rows from the Book table in which the price column contains a value greater than 100.00. The result is sorted in ascending order by title. The asterisk (*) in the select list indicates that all columns of the Book table should be included in the result set.

```
SELECT *
    FROM Book
    WHERE price > 100.00
    ORDER BY title;
```

The example below demonstrates a query of multiple tables, grouping, and aggregation, by returning a list of books and the number of authors associated with each book.

```
SELECT Book.title,
        COUNT(*) AS Authors
    FROM  Book JOIN Book_author
       ON Book.isbn = Book_author.isbn
    GROUP BY Book.title;
```

SQL includes operators and functions for calculating values on stored values.

Each column in an SQL table declares the type(s) that column may contain. ANSI SQL includes *character strings*, *bit strings*, *numbers*, *date* and *time*.

Finally SQL includes as sub-sets the following languages:

- Data Manipulation Language (DML) to insert, cancel or modify data (INSERT, UPDATE, DELETE and MERGE statements).
- Data Definition Language (DDL) to create or delete a database and to modify the table and index structure (CREATE, ALTER, RENAME, DROP and TRUNCATE statements).
- Data Control Language (DCL) to manage user and privileges (GRANT and REVOKE statements)

2.1.3 XQuery

XML Query Language – or simply XQuery – is a functional programming and query language that is designed to query XML databases or collections of XML data.

From w3schools "XQuery is to XML what SQL is to database tables".

XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents.

The XQuery query reads a sequence of XML nodes and returns a selected sequence of nodes.

XQuery uses XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN and it is a generalization of SQL's SELECT-FROM-HAVING-WHERE construct.

The language also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used; in other cases, expressions referred to as dynamic node constructors are available. All these constructs are defined as expressions within the language, and can be arbitrarily nested.

The language is based on a tree-structured model of the information content of an XML document, containing seven kinds of node: document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.

The type system of the language models all values as sequences (a singleton value is considered to be a sequence of length one). The items in a sequence can either be nodes or atomic values. Atomic values may be integers, strings, booleans, and so on: the full list of types is based on the primitive types defined in XML Schema.

XQuery 1.0 does not include features for updating XML documents or databases; it also lacks full text search capability. These features are both under active development for a subsequent version of the language.

Basic syntax:

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a $ followed by a name, e.g. $bookstore

- XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)
- Conditional expressions ("If-Then-Else") are allowed in XQuery.
- Comparison expressions:
  - General comparisons: =, !=, <, <=, >, >=
  - Value comparisons: eq, ne, lt, le, gt, ge

The sample XQuery code below lists the title (in alphabetical order) of the books encoded in the XML document *book.xml*, whose price is greater than 30 Euros.

This is how the *books.xml* file looks like:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

For an XQuery like this:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

the output is:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

2.1.4 XML Simple Query

XML Simple Query, also called XSQuery, is the query language adopted by eCTRL's Suggesto Recommender System to simply model a user query.

The query model of this language is based on a query by example approach and allows conjunction of constraints where each constraint is either a comparison condition (e.g., colour=red, or cost < 100), a "range" condition (e.g., cost BETWEEN 50 AND 100), or "like" condition (e.g., location LIKE "%fiemme").

Every XSQuery document contains one or more *oneColumnCond* element as direct children of the root *XSQuery*. Each *oneColumnCond* constrains a feature of the target collection. This feature is the content of the *identifier* element and it is specified using XPath syntax.

Let X be a collection and x be any element in this collection. Let $T_x = \{ e_1, \ldots, e_k \}$ be the type of x. To refer to any feature of x, the corresponding XPath notation would be $/X/x/e_i$ ($1<=i<=k$). For instance the *name* feature of a Destination in the DESTINATION collection would be referred by its XPath as

```
/DESTINATION/Destination/name
```

The following schema describes the XSQuery syntax used by Suggesto.

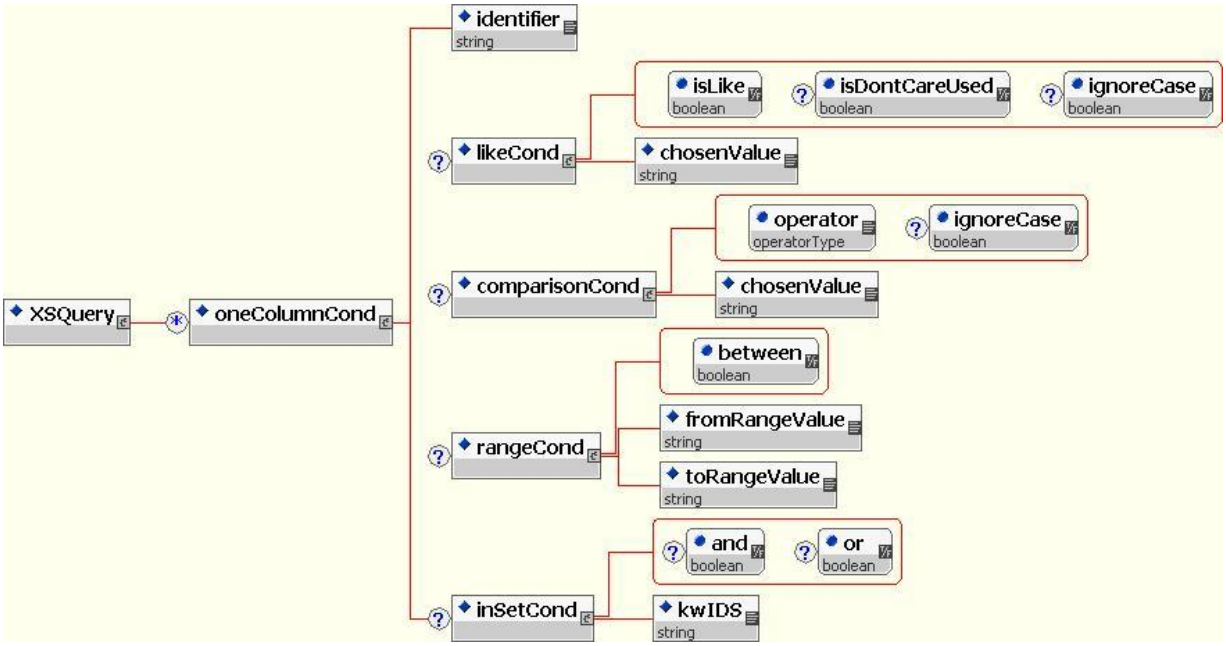

Figure 1 XSQuery schema definition

Other type of constraints as well as additional information (e.g. on the context of the query) can be handled too by defining and adding new elements to the schema.

2.2 QUERY LANGUAGES ANALYSIS AND SELECTION

In this section the query languages presented in the previous paragraphs are evaluated agaist the technical requirements identified in D3.1 in order to select the most appropriate query language to be used in HarmoSearch.

ID	Requirement	QBE	SQL	XQuery	XML
FR1	Simple data types	+++	+++	+++	+++
FR2	Complex data types	+	+	+	+
FR3	Basic operations	+++	+++	+++	+++
FR4	Complex operations	+++	+++	+++	+++
FR5	Logic operations	+++	+++	+++	+++
FR6	Data specific operations	++	++	++	++
FR7	Condition operations	++	+	+	+++
FR8	Selection operations	+++	+++	+++	+++
FR9	Sorting the results	+++	+++	+++	+++
FR10	Operations on the results	+++	+++	+++	+++
NFR1	Domain independence	+++	+++	+++	+++
NFR2	Extensibility	++	++	++	+++
NFR3	Robustness and security	+++	+++	+++	+++
NFR4	Human readable	+++	+	+	+
NFR5	Easily mappable and convertible	+	+	+	++
NFR6	Open standard/not vendor specific	+++	+++	+++	+++
AR1	Identifier of the sender	++	+	+	+++
AR2	Intended receivers	++	++	++	+++
AR3	Type of request	++	+	+	+++
AR4	Reference to the domains or collections	++	+	+	+++
AR5	Geographical region of interest	++	+	+	+++

AR6	Preferred language	++	+	+	+++
AR7	Priority	++	+	+	+++
AR8	Preferred output	++	+	+	+++
AR9	Paging preferences	++	+	+	+++
AR10	Get all available data providers	+	+++	+	++
AR11	Get all available collections	+	+++	+	++
AR12	Get collections of a given data provider	+	+++	+	++
AR13	Get fields of a given collection	+	+++	+	++
AR14	Get type of a given field	+	+++	+	++
AR15	Get pre-defined values of a given enumerated type	+	+++	+	++
AR16	Get primary and foreign keys	+	+++	+	++
Overall Evaluation		**66**	**69**	**55**	**83**

On the basis of the evaluation matrix presented above, the query language that will be used in the HarmoSearch project will be based on XML. Due to the flexible nature of XML (elements and attributes can be created and personalised according to the needs), this language seems to be the most suitable to implement the complexity of the HarmoSearch reference model, and in particular the concepts related to the context of the query. Moreover its extensibility makes it very easy to to add new elements or attributes in the future, in case additional concepts have to be modelled or new requirements arise.

3 HARMOSEARCH QUERY REFERENCE MODEL

Here below is reported a refined version of the HarmoSearch query reference model defined in D3.1. This model describes the concepts to allow representing a Harmonise query in an abstract form. These concepts will be formalised in the next paragraph, using the chosen XML notation, to produce the HarmoSearch query language.

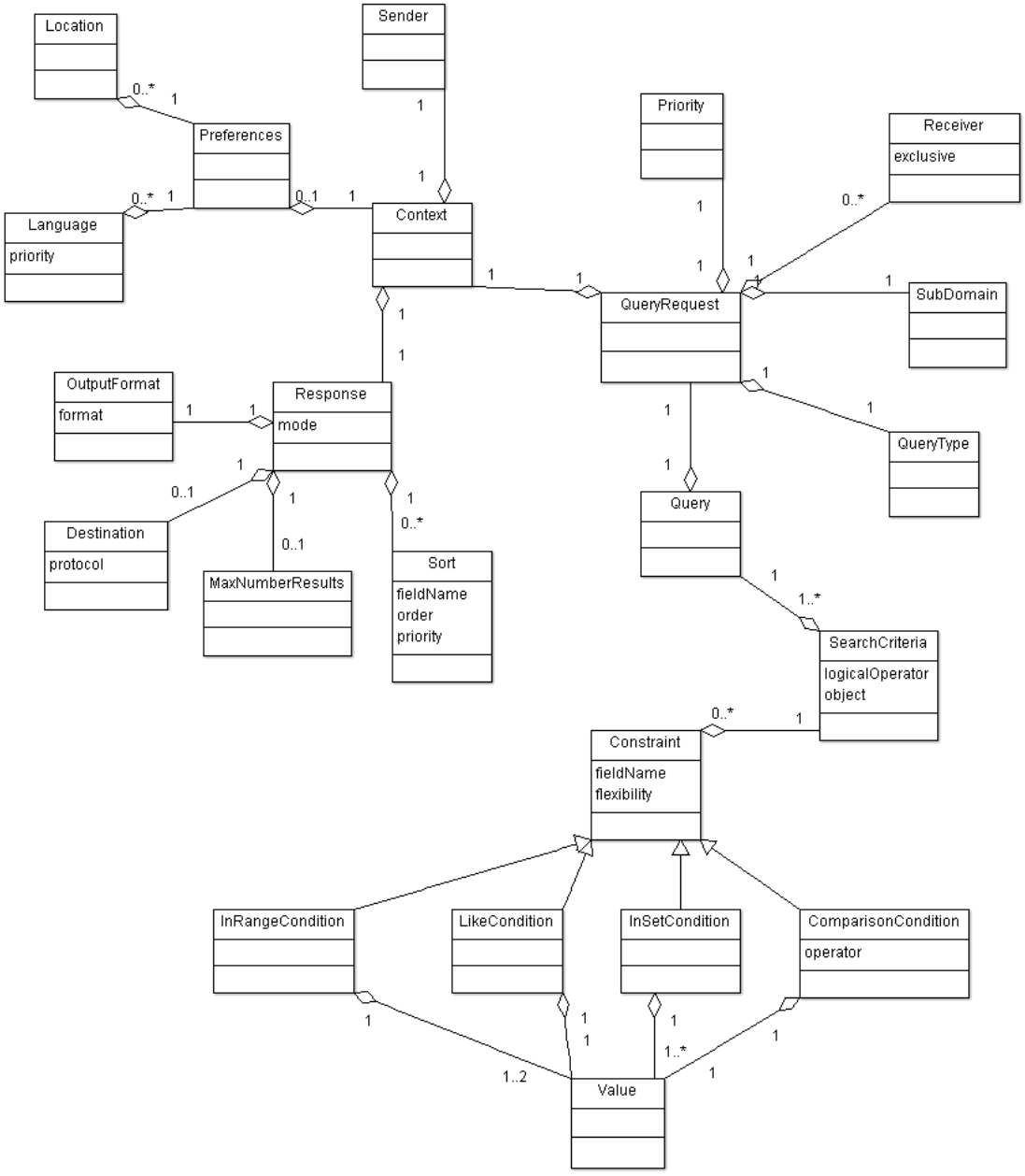

Figure 2 HarmoSearch query reference model

For the detailed description of the main concepts refer to D3.1.

4 HARMOSEARCH QUERY LANGUAGE

4.1 QUERY LANGUAGE SCHEMA

The following schema describes the HarmoSearch query language syntax.

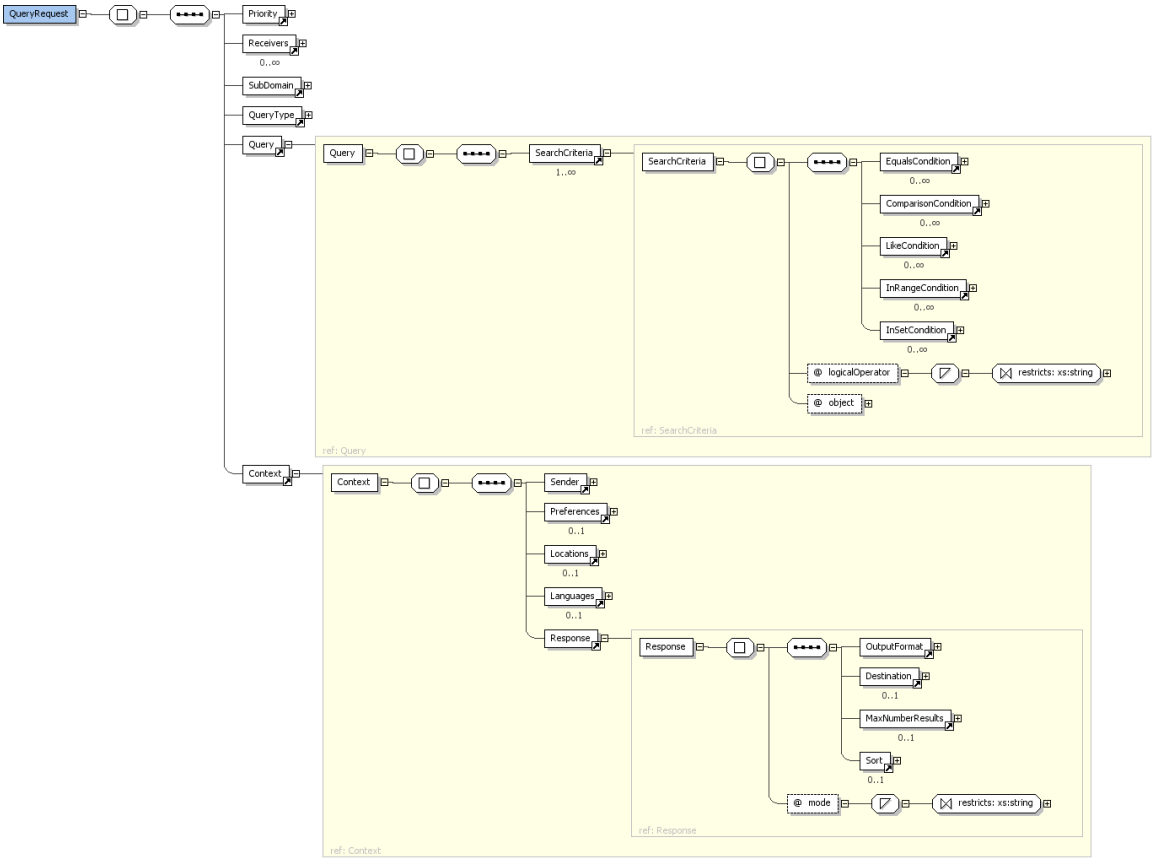

Figure 3 HarmoSearch query language schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="QueryRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Priority"/>
        <xs:element ref="Receivers" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element ref="SubDomain"/>
        <xs:element ref="QueryType"/>
        <xs:element ref="Query" maxOccurs="1"/>
        <xs:element ref="Context"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Context">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element ref="Sender"/>
        <xs:element ref="Preferences" minOccurs="0"/>
        <xs:element ref="Locations" minOccurs="0"/>
        <xs:element ref="Languages" minOccurs="0"/>
        <xs:element ref="Response"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Sender" type="xs:string"/>
  <xs:element name="Preferences">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>In order to extend the preferences add new
elements to the sequence.</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element ref="Locations" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="Languages" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Locations">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Location" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Location" type="xs:string"/>
  <xs:element name="Languages">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Language" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Language">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="priority" type="xs:int" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Response">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="OutputFormat"/>
        <xs:element ref="Destination" minOccurs="0"/>
        <xs:element ref="MaxNumberOfResults" minOccurs="0"/>
        <xs:element ref="Sort" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="mode" default="SYNC">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="SYNC"/>
            <xs:enumeration value="ASYNC"/>
          </xs:restriction>
        </xs:simpleType>
```

```
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="OutputFormat">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Field" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="format" default="XML">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="XML"/>
          <xs:enumeration value="HARMO"/>
          <xs:enumeration value="CSV"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="Field">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="as" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="Destination">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="protocol" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="REST"/>
              <xs:enumeration value="SOAP"/>
              <xs:enumeration value="FTP"/>
              <xs:enumeration value="EMAIL"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="MaxNumberOfResults" type="xs:int"/>
<xs:element name="Sort">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="OrderBy" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="OrderBy">
  <xs:complexType>
    <xs:attribute name="order" default="ASC">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ASC"/>
```

```
            <xs:enumeration value="DESC"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="priority" type="xs:int" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Priority" default="NORMAL">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="HIGH"/>
        <xs:enumeration value="NORMAL"/>
        <xs:enumeration value="LOW"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Receivers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Receiver" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="exclusive" type="xs:boolean" default="true"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Receiver" type="xs:string"/>
  <xs:element name="SubDomain" type="xs:string">
    <xs:annotation>
      <xs:documentation>
        Contains a reference to a subdomain described in the
        semantic registry. The following subdomains are standard
        values: "Event", "Accommodation", "Attraction", "Gastro"
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="QueryType" default="METASEARCH">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="IMPORT"/>
        <xs:enumeration value="METASEARCH"/>
        <xs:enumeration value="RECOMMEND"/>
        <xs:enumeration value="AD-HOC"/>
        <xs:enumeration value="METADATA"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Query">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="SearchCriteria" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SearchCriteria">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="EqualsCondition" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element ref="ComparisonCondition" minOccurs="0"
maxOccurs="unbounded"/>
```

```
        <xs:element ref="LikeCondition" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element ref="InRangeCondition" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element ref="InSetCondition" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="logicalOperator" default="AND">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="AND"/>
            <xs:enumeration value="OR"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="object" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="EqualsCondition">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="fieldName" type="xs:string"
use="required"/>
          <xs:attribute name="flexibility" type="xs:boolean"
default="false"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="ComparisonCondition">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:int">
          <xs:attribute name="fieldName" type="xs:string"
use="required"/>
          <xs:attribute name="flexibility" type="xs:boolean"
default="false"/>
          <xs:attribute name="operator" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="lt"/>
                <xs:enumeration value="lte"/>
                <xs:enumeration value="gt"/>
                <xs:enumeration value="gte"/>
                <xs:enumeration value="eq"/>
                <xs:enumeration value="diff"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="LikeCondition">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="fieldName" type="xs:string"
use="required"/>
```

```
          <xs:attribute name="flexibility" type="xs:boolean"
default="false"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="InRangeCondition">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="From" minOccurs="0"/>
        <xs:element ref="To" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="fieldName" type="xs:string" use="required"/>
      <xs:attribute name="flexibility" type="xs:boolean"
default="false"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="From" type="xs:date"/>
  <xs:element name="To" type="xs:date"/>
  <xs:element name="InSetCondition">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Value" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="fieldName" type="xs:string" use="required"/>
      <xs:attribute name="flexibility" type="xs:boolean"
default="false"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Value" type="xs:string"/>
</xs:schema>
```

4.2 QUERY LANGUAGE DESCRIPTION

This section provides a description of the elements and options of the query language.

4.2.1 QueryRequest

`QueryRequest` is the root element of the HarmoSearch query. It contains the priority, the receivers, the domain of reference, the type of query, the context information and the search criteria. All these sub-elements are detailed in the following paragraphs.

4.2.2 Priority

The `Priority` element represents the level of importance of a query request: requests with low priority could be cheaper than high priority jobs. Possible values are:

- "HIGH"
- "NORMAL" (default)
- "LOW" Example:

```
<Priority>NORMAL</Priority>
```

4.2.3 Receivers

If the `Receivers` element is present, it specifies the data providers whom to constrain the search to. The `exclusive` attribute specifies whether the receivers are optional or mandatory, i.e. if additional relevant providers should be contacted too (`exclusive="false"`) or if only providers from the set should be considered (`exclusive="true"`). Default is "true". The value of each `Receiver` element is the unique ID of the provider within the HarmoSearch environment.
Example:

```
<Receivers exclusive="true">
    <Receiver>10801</Receiver>
    <Receiver>10802</Receiver>
</Receivers>
```

4.2.4 SubDomain

The SubDomain element represents the particular domain of the actual request. The value of such an element is a reference to a description of the used part of the harmonise ontology. As an example, the data exchanged in the Euromuse network could contain events, containing an English title and subtitle, an English short and long description and a number of other data features.

These data descriptions are stored in the Harmonise semantic registry and referenced when registering a mapping (i.e., a data provider provides data from a certain part of the harmonise ontology). Furthermore, these references are used when querying, specifying the relevant part of the Harmonise ontology regarding the specific query.

Especially in the metasearch scenario, the Harmonise semantic registry then has the task of reasoning whether the data provided by the data providers matches the data content desired in the query. This is done through subsumption reasoning on an OWL description of the specific data formats.

There can be many different (and also equal) data subformats defined, where one definition (e.g., for the Euromuse use case) does not need to be aware of any other defined subdomains except for the ones specified in the Harmonise ontology.

As a basic data subdomain the main concepts of the Harmonise ontology are defined. The semantic of these definitions is "data describing items of the given subdomain in any depth" and thus may contain insufficient details for practical purposes.

- "Events"
- "Accommodation"
- "Attractions"
- "Gastro"

Further subdomain specifications can be created (or reused) as required by the usage scenarios without having to look for the correct inheritance hierarchy or possibly already existing equivalent specifications. This is covered by the OWL reasoned of the semantic registry.

Example:

```
<SubDomain>Events</SubDomain>

<SubDomain>Euromuse</SubDomain>
```

4.2.5 QueryType

The `QueryType` element denotes the type of request, i.e. the use case which the query refers to. Possible values are:

- "IMPORT", which means batch transfer of static data
- "METASEARCH", which means search for data items across distributed data sources (default)
- "RECOMMEND", which means to get recommendations about items related to a specific topic of interest
- "AD-HOC", which means sending an ad-hoc query to a number of data providers
- "METADATA", which means to query schema and metadata information

Different or additional query types are to be developed by the HarmoSearch community.

Example:

```
<QueryType>METASEARCH</QueryType>
```

4.2.6 Context

The `Context` element denotes the environment of a query. It contains the sender of the request, his profile preferences, the geographical area of interest, the preferred language and how the response should look like. All these sub-elements are detailed in the following paragraphs.

4.2.7 Sender

The Sender element represents the actor that initiates a query request. Its value is the unique ID of the provider within the HarmoSearch environment.

Example:

```
<Sender>10803</Sender>
```

4.2.8 Preferences

The `Preferences` element may specify additional preferences which have a corresponding value in the user profile of the data providers. At the moment the `Preferences` field can have a `Languages` and a `Locations` element specified. These options should be seen as being flexible and volatile, thus they can be extended or restricted as the need arises.

Example:

```
<Preferences>
    <Locations> ... </Locations>
    <Languages> ... </Languages>
</Preferences>
```

4.2.9 Locations

If it is present, the `Locations` element specifies the geographical regions for which the provider claims a special focus. This is not a restriction for the data to be searched for but for the provider description (e.g. find a provider who specializes in the Tuscany region and has Italian as a first language). More than one region can be specified. HarmoSearch query language uses the ISO 3166 two-letter (or 'A2') country codes.

Example:

```
<Locations>
    <Location>Florence</Location>
    <Location>Pisa</Location>
</Locations>
```

4.2.10 Languages

If it is present, the `Languages` element specifies the "main" languages of the provider for the content of the results. This indicates that the content is created primarily in these languages by native speakers and domain experts. Thus the quality should be high in comparison to translations by non-experts from a translation company.
The HarmoSearch query language uses the ISO 639-1 standard: two lower-case letters represent a language.

Example:

```
<Languages>
    <Language priority="1">it</Language>
    <Language priority="2">en</Language>
</Languages>
```

4.2.11 Response

The `Response` element specifies how the list of result items should look like. It contains the preferred output, the return address for asynchronous results, paging and sorting preferences. All these sub-elements are detailed in the following paragraphs.
The `mode` attribute specifies whether the search results have to be returned to the requester synchronously ("SYNC" – default), i.e. all in a bunch after they have been collected from the data providers, or asynchronously ("ASYNC"), i.e. returning immediately portion of the results as soon as they become available.

4.2.12 OutputFormat

The `OutputFormat` element specifies the preferred output format of the response. The value of the `format` attribute indicates the output format, e.g.

- "XML", which means XML based on the requester's schema (default)
- "HARMO", which means XML based on Harmonise ontology (i.e. without reconciliation)
- "CSV"

Different or additional output formats are to be developed by the HarmoSearch community.

Normally, all fields present in the result data sets are returned. Only if a set of `Field` elements are specified, they represent which particular fields should be returned in the response. Each `Field` may have an `as` attribute, whose value – if present - specifies a different name to be used for the field instead of the default one.

The value of the `Field` element follows the XPath syntax rules described in the paragraph "SearchCriteria", with the object of consideration being the object specified through the `Subdomain` field.

If a field is selected which has sub-elements in the Harmonise ontology, then this indicates that all subelements should be returned.

Example – base object being "Events":

```
<OutputFormat format="XML">
    <Field>/eventTitle/mainTitle/languageText/text</Field>
    <Field>/location/address/city/languageText/text</Field>
    <Field>/price/priceRange/max/amount</Field>
</OutputFormat>
```

4.2.13 Destination

By default search results are sent back to the sender of the query. If the `Destination` element is present, it specifies a different way to send the response to the user who originates the request. The value of the `protocol` attribute indicates the communication technology which is used to send the results, e.g.

- "REST", to send the results to an HTTP service
- "SOAP", to send the results to a Web Service
- "FTP", to send the results via FTP
- "EMAIL", to send the results via email

The value of the `Destination` element denotes the return address of the response, i.e. the url of the service or the email address where to send the results.

Different or additional ways to return the response are to be developed by the HarmoSearch community.

Example:

```
<Destination protocol="EMAIL">c.prandoni@cpr.it</Destination>
```

4.2.14 MaxNumberOfResults

The `MaxNumberOfResults` element – if present - denotes the maximum number of result items that should be returned.
Example:

```
<MaxNumberOfResults>500</MaxNumberOfResults>
```

4.2.15 Sort

If it is present, the `Sort` element specifies the criteria according to which to sort the search results. It is composed of one or more `OrderBy` sub-elements representing the fields to be used for sorting. Each `OrderBy` element has two attributes:

- `order`, which can be "ASC" (ascending - default) or "DESC" (descending)
- `priority`, which says which is the field to be sorted first

The value of the `OrderBy` element follows the XPath syntax rules described in the paragraph "SearchCriteria".

Example – base object being "Events":

```
<Sort>
    <OrderBy order="ASC"
priority="1">/price/priceRange/max/amount</OrderBy>
    <OrderBy order="ASC"
priority="2">/eventTitle/mainTitle/languageText/text</OrderBy>
</Sort>
```

4.2.16 Query

The `Query` element models the content of a query, with its search conditions. It contains several, at least one, `SearchCriteria` element. All these `SearchCriteria` elements have to be fulfilled in order to fulfill the query. Therefore, they are implicitly AND connected.

4.2.17 SearchCriteria

The `SearchCriteria` element models the different search criteria, i.e. how to constrain the search for each of the specified fields. It contains one or more conditions, which are described in detail in the following paragraphs.

If no object is stated, then all paths are considered to be relative to the object selected in the `subdomain` field. The descriptions of these subdomains are stored in the semantic registry. Each subdomain description specifies a restriction on the elements "Event", "Accommodation", "Attraction" and "Gastro". Therefore, the corresponding elements of the Harmonise ontology are seen as the root of all path statements for which no specific object is designated.

If an object is stated explicitly for the `SearchCriteria` through the `object` attribute, then the object has to be specified in the same way. In this case, the `fieldname` and `object` attributes of all subsequent the conditions are relative to the specified object.

For example, let's assume that "Euromuse" is selected as subdomain. "Euromuse" is specified in the registry to be a kind of "Event" description. Therefore, the path "/eventTitle/mainTitle/languageText/text" corresponds to "Event/eventTitle/..."

All the condition elements share two common attributes:

- `fieldname`, whose value is specified using XPath syntax and represents the feature of the target collection to be constrained. Let X be a collection and x be any element in this collection. Let Tx = { e1 ,…, ek } be the type of x. To refer to any feature of x, the corresponding XPath notation would be /X/x/ei

(1<=i<=k). For instance the name feature of a Destination in the DESTINATION collection would be referred by its XPath as: "/DESTINATION/Destination/name"

- `flexibility`, which specifies whether the constraint is mandatory (flexibility="false" - default) or optional (flexibility="true"), i.e. it is not necessary that the search results match these latter criteria but it is a desiderata

Different search criteria can be combined according to the value of the `logicalOperator` attribute:

- "AND" means that the search results have to match all the different conditions
- "OR" means that the search results have to match at least one of the different conditions

Example – for a subdomain of type Event the following statement returns all events which have a maximum price lower than 100 Euros:

```
<SearchCriteria logicalOperator="AND" object="/price/priceRange/max">
    <EqualsCondition fieldName="/currency"
flexibility="false">EUR</EqualsCondition>
    <ComparisonCondition operator="lte"
fieldName="/amount" flexibility="false">100</ComparisonCondition>
</SearchCriteria>
```

4.2.18 EqualsCondition

The `EqualsCondition` element represents a match condition. The value of the element is the value specified by the user which to constrain the search to.

Example - base object being "Events":

```
<EqualsCondition fieldName="/location/address/region/languageText/text"
flexibility="false">Tuscany</EqualsCondition>
```

4.2.19 ComparisonCondition

The `ComparisonCondition` element represents a comparison condition. The value of the element is the value specified by the user. It has an `operator` attribute, whose possible values are:

- "lt" representing <
- "lte" representing <=
- "gt" representing >
- "gte" representing >=
- "eq" representing =
- "diff" representing <>

Examples – base object being "Accommodation":

```
<ComparisonCondition fieldName="/award/awardAchieved"
flexibility="false" operator="gte">3</ComparisonCondition>
```

4.2.20 LikeCondition

The `LikeCondition` element represents a partial match condition for symbolic features. The value of the element is the substring for partial matching. Let "chosenValue" be the desired substring that the client would like to search for values of the given feature:

- "chosenValue%" means any value starting with "chosenValue"
- "%chosenValue%" means any value which contains "chosenValue" anywhere in the value of the feature (default)
- "%chosenValue" means any value ending with "chosenValue"

Example - base object being "Accommodation":

```
<LikeCondition fieldName="/name/languageText/text"
flexibility="false">%Best Western%</LikeCondition>
```

4.2.21 InRangeCondition

The `InRangeCondition` element represents a date range condition. The content of the `From` element is the lower range value and, analogously, the content of the `To` element is the upper range value.

Examples - base object being "Events":

```
<InRangeCondition fieldName="/timeline/dateRange/startDate"
flexibility="false">
    <To>2011-08-15</To>
</InRangeCondition>

<InRangeCondition fieldName="/timeline/dateRange/endDate"
flexibility="false">
    <From>2011-08-01</From>
</InRangeCondition>
```

4.2.22 InSetCondition

The `InSetCondition` element represents a membership condition which allows to determine whether the value of an expression is equal to any of several values in a specified list. These values are the values of the `Value` sub-elements.

Example – base object being "Events":

```
<InSetCondition fieldName="/category/value" flexibility="false">
    <Value>modern art</Value>
    <Value>painting</Value>
</InSetCondition>
```

4.3 EXAMPLE

Here is an example of how a complete query request looks like.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<QueryRequest>
    <Priority>normal</Priority>
    <Receivers exclusive="true">
        <Receiver>10801</Receiver>
        <Receiver>10802</Receiver>
    </Receivers>
    <SubDomain>Events</SubDomain>
    <QueryType>METASEARCH</QueryType>
    <Query>
        <SearchCriteria logicalOperator="AND"
    object="/price/priceRange/max">
            <EqualsCondition fieldName="/currency"
    flexibility="false">EUR</EqualsCondition>
            <ComparisonCondition operator="lte"
    fieldName="/amount" flexibility="false">100</ComparisonCondition>
        </SearchCriteria>
        <SearchCriteria logicalOperator="OR"
      object="/timeline/dateRange">
            <InRangeCondition fieldName="/startDate"
    flexibility="false">
                <To>2011-08-15</To>
            </InRangeCondition>
            <InRangeCondition fieldName="/endDate" flexibility="false">
                <From>2011-08-01</From>
            </InRangeCondition>
        </SearchCriteria>
        <SearchCriteria logicalOperator="AND">
            <EqualsCondition
        fieldName="/location/address/region/languageText/text"
        flexibility="false">Tuscany</EqualsCondition>
            <InSetCondition fieldName="/category/value"
    flexibility="false">
                <Value>modern art</Value>
                <Value>painting</Value>
            </InSetCondition>
        </SearchCriteria>
    </Query>
    <Context>
        <Sender>10803</Sender>
        <Preferences>
            <Locations>
                <Location>Florence</Location>
                <Location>Pisa</Location>
            </Locations>
            <Languages>
                <Language priority="1">it</Language>
                <Language priority="2">en</Language>
            </Languages>
        </Preferences>
```

```
        <Response>
            <OutputFormat format="XML">
                <Field>/eventTitle/mainTitle/languageText/text</Field>
                <Field>/location/address/city/languageText/text</Field>
                <Field>/price/priceRange/max/amount</Field>
            </OutputFormat>
            <Destination
protocol="EMAIL">c.prandoni@cpr.it</Destination>
            <MaxNumberOfResults>20</MaxNumberOfResults>
            <Sort>
                <OrderBy order="ASC"
priority="1">/price/priceRange/max/amount</OrderBy>
            </Sort>
        </Response>
    </Context>
</QueryRequest>
```

5 LIST OF FIGURES