GBIF Data Access and Database Interoperability

A unified protocol for search and retrieval of distributed data

Markus Döring & Renato De Giovanni

September 2004

Contents

1 Summary

This report is the result of a study promoted by the Global Biodiversity Information Facility (GBIF) to integrate two of the main protocols being used for search and retrieval of biodiversity data. Together, DiGIR and BioCASe are currently serving approximately 40 million records from distributed and heterogeneous databases of biological collections and observation data holders.

The unification of DiGIR and BioCASe will bring greater interoperability between the existing networks, therefore facilitating development of tools, stimulating participation of new data providers, and increasing access to biodiversity data. This effort is also seen as a considerable step towards having a single standard protocol to be used within the biodiversity area.

This document contains a review of both protocols, a general study about other specifications and technologies that could serve as a potential source for new ideas, an integration proposal that could be implemented by the existing tools in a short term, and finally some additional recommendations.

2 Acknowledgements

The authors of this report wish to thank all individuals and institutions that contributed to and participated in various aspects of this work, in special:

Anton Güntsch, Dave Vieglais, Donald Hobern, Javier de la Torre, John Wieczorek, Stan Blum for their permanent contribution and assistance during the whole process.

Gregor Hagedorn, Mauro Muñoz, Ronald Bourret, Wolfgang Lipp for all suggestions and healthy discussions.

Global Biodiversity Information Facility (GBIF) for promoting and supporting the whole work.

University of Kansas Natural History Museum and Biodiversity Research Center, California Academy of Sciences, and Museum of Vertebrate Zoology (MVZ) in Berkeley for creating and promoting DiGIR.

Botanischer Garten und Botanisches Museum Berlin-Dahlem (BGBM) for creating and promoting BioCASE and for all assistance during this work.

Centro de Refêrencia em Informação Ambiental (CRIA) for its contributions to DiGIR and for all assistance during this work.

3 Revision history

Date	**Responsible**	**Notes**
23 Sep 2004	Markus Döring & Renato De Giovanni	Initial version
16 Nov 2004	Renato De Giovanni	Minor corrections

4 Introduction

During GBIF's Data Access and Database Interoperability subcommittee meeting held in Oaxaca 2004, the importance of integrating DiGIR and BioCASe has been fully recognized and considered a priority. Since then, GBIF promoted a study involving representatives from both protocols.

Specialists from the biodiversity community and representatives from the existing biodiversity networks have been contacted to provide comments and suggestions about the protocol they were using. Many people participated, and the whole work has been carried out by meetings, electronic messages and a public wiki web site[1].

Both original protocols have been reviewed and documented, and the study also included considerations taken from other existing standards such as XQuery, OGC specifications and SOAP.

The results from that work have been documented in this report, including a complete proposal for the new integrated protocol.

[1] http://ww3.bgbm.org/protocolwiki/

5 Review of the main protocols

5.1 DiGIR protocol

5.1.1 Brief history

DiGIR[2] was conceived as a replacement for the Z39.50[3] protocol being used by the Species Analyst network (TSA)[4]. TSA was created as part of a research project that aimed at providing access to distributed natural history collections and observation databases with over 120 connected data sources.

The Z39.50 protocol was considered too complicated, resulting in a steep learning curve for developers, and in difficulties to be accepted by network administrators. Other technical reasons at that time included the lack of a more formal language to define conceptual schemas, and also limited support for XML and Unicode.

The DiGIR protocol specification, its "default" conceptual schema known as DarwinCore[5], and the development of the first related software originally involved three institutions: University of Kansas Natural History Museum and Biodiversity Research Center[6], California Academy of Sciences[7], and Museum of Vertebrate Zoology[8] in Berkeley. The general strategy for the project was:

- To use open protocols and standards (HTTP, XML and UDDI).
- To clearly de-couple protocol, software and semantics.
- To ease software installation and configuration for data providers as much as possible.

Software tools were developed in a collaborative environment[9] following the "open source" model, and the results were made available under public license.

The Species Analyst data providers are gradually migrating to DiGIR and building separate thematic networks such as Manis[10], HerpNet[11], FishNet[12]

[2] http://digir.net
[3] http://www.loc.gov/z3950/agency/
[4] http://speciesanalyst.net
[5] http://speciesanalyst.net/docs/dwc/index.html
[6] http://www.nhm.ku.edu
[7] http://www.calacademy.org
[8] http://www.mip.berkeley.edu/mvz/
[9] http://sourceforge.net/projects/digir
[10] http://elib.cs.berkeley.edu/manis/
[11] http://herpnet.org/
[12] http://habanero.nhm.ku.edu/fishnet/

and ORNIS[13]. Other networks around the world have also adopted DiGIR as the main protocol (speciesLink[14], OBIS[15]) or as one of the supported protocols (GBIF[16]).

5.1.2 Description

DiGIR is an XML-based protocol over HTTP to retrieve data from independent and heterogeneous databases. To achieve a uniform virtual view from network participants, each DiGIR resource needs to choose one or more conceptual schemas to use and then map some fields from a local database against the elements from those schemas. A conceptual schema is a common data model that serves as a reference on networks of federated databases.

Different conceptual schemas can be defined by each network, and therefore the protocol can be used by any community regardless the discipline. In DiGIR, a conceptual schema is represented as an XML Schema that follows a specific format and defines a set of elements (concepts) in a flat list. An example of such a schema is DarwinCore[17], the conceptual schema originally proposed to be used by biological data sources in DiGIR-enabled networks.

A typical DiGIR network using the components developed by the original project involves three different software. The first one is a client software, known as the "presentation layer". The client software interacts with end users through some interface, and communicates with another component known as the "portal engine". The portal engine is a query dispatcher. It knows the addresses of several data providers either by manual configuration or by interaction with UDDI registries. Queries are then distributed to the selected data providers, which are responsible for translating DiGIR messages to the query language used by local databases and retrieving the requested data.

Other type of interactions and architectures are possible, including monitoring services, indexing services, and even portals on top of other portals.

Data providers can serve data from several "resources". A DiGIR resource is seen as a local data source that has mapped its structure against a single conceptual schema, although a conceptual schema can be an extension from another one. These schemas provide a uniform view for heterogeneous resources. It is also known that a DiGIR resource can map its data structure against more than one conceptual schema, thus allowing for the existence

[13] http://www.specifysoftware.org/Informatics/informaticsornis/
[14] http://splink.cria.org.br/
[15] http://iobis.org/
[16] http://www.gbif.net/
[17] http://digir.net/schema/conceptual/darwin/2003/1.0/darwin2.xsd

of completely independent and modularized schemas. Networks can for instance map their DiGIR resources against a conceptual schema which is common among other networks, but also map additional conceptual schemas which could be specific to their context.

The original scope of the DiGIR protocol schema[18] is to validate messages exchanged only with providers. Communication with resources can only be done through providers. In that case resources are referenced inside the messages (they don't have an access point such as the providers' URL). Messages exchanged between other components, like presentation layers and query dispatchers are not covered by DiGIR.

5.1.3 Technical details

General message format

DiGIR messages have three main sections inside a <request> or <response> root element. The <header> section is present in all messages, and it includes information about the software version which produced the message, the time stamp, the URL or IP address from where the message originated, the URL or IP address to where the message is destinated, and the type of request. Some types of request may have an additional attribute in the header destination element to reference a DiGIR resource.

After the <header> there may be a content section depending on the type of message. In requests it may contain filter conditions and specifications about the concepts to be returned. In responses it contains the structured content being returned.

Responses have an additional <diagnostics> section that carries information about possible errors, warnings, or simply some additional information such as the number of records that matched a query.

Since a DiGIR provider can be seen as a query dispatcher regarding its local resources, a provider should be able to accept multiple destination elements in the header related to more than one local resource. In this case, responses from resources are concatenated and enclosed in a <responseWrapper> root element.

The next pages cover all types of requests and responses with several examples of DiGIR messages.

[18] http://digir.net/schema/protocol/2003/1.0/digir.xsd

Request and response types

There are three different request and response types that could be used to exchange messages with a DiGIR service:

DiGIR metadata operation

Metadata requests ask for information about a provider service and its resources. It is also the default request when a provider's URL is accessed without any parameters. Metadata requests contain only the header section, and the destination element should be a provider's access point.

Sample DiGIR metadata request using an XML message:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>1.0.0</version>
    <sendTime>2004-07-20T08:51:50-0500</sendTime>
    <source>127.0.0.1</source>
    <destination>
     http://example.net/provider/DiGIR.php
    </destination>
    <type>metadata</type>
  </header>
</request>
```

Metadata responses include general information about the provider (only name and access point); about the entity who is hosting the service (name, web site address, contact information, and an optional abstract); and about each available resource. Resources also have some general information such as name, code (used to reference resources in requests); content information such as number of available records, timestamp of the last updated record, abstract, keywords, and record basis; technical information such as maximum number of records that can be retrieved, minimum number of characters to be used in <like> operations; and other elements related to how to make citations and if there are any restrictions to use retrieved data. An important part of resource metadata is what conceptual schemas have been mapped.

Sample DiGIR metadata response:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>$Revision: 1.96 $</version>
    <sendTime>2004-07-20T08:51:55-0500</sendTime>
    <source>http://example.net:80/provider/DiGIR.php</source>
    <destination>127.0.0.1</destination>
    <type>metadata</type>
  </header>
  <content>
    <metadata>
      <provider>
        <name>Example Biodiversity Center</name>
        <accessPoint>
         http://example.net:80/provider/DiGIR.php
        </accessPoint>
        <implementation>$Revision: 1.96 $</implementation>
        <host>
          <name>Some Hosting Institution</name>
          <code>SHI</code>
          <relatedInformation>
           http://example.net/
          </relatedInformation>
          <contact type="technical">
            <name>Person A</name>
            <title>Support specialist</title>
            <emailAddress>person_a@example.net</emailAddress>
            <phone>+11 22 333333</phone>
          </contact>
          <abstract>Default hosting institution to be used
           in examples.
          </abstract>
        </host>
        <resource>
          <name>Herbarium dataset</name>
          <code>HDS</code>
          <relatedInformation>
           http://example.net/herbarium
          </relatedInformation>
          <contact type="administrative">
            <name>Person B</name>
```

```
        <title>Curator</title>
        <emailAddress>person_b@example.net</emailAddress>
        <phone>+11 22 334444</phone>
      </contact>
      <contact type="technical">
        <name>Person C</name>
        <title>Systems Analyst</title>
        <emailAddress>person_c@example.net</emailAddress>
        <phone>+11 22 334445</phone>
      </contact>
      <abstract>
       Plant specimen dataset from some region.
      </abstract>
      <keywords>plant, specimen</keywords>
      <citation>Herbarium dataset DiGIR provider.
       Retrieved on (date accessed).
       http://example.net:80/provider/DiGIR.php
      </citation>
      <useRestrictions>Users may not distribute, modify,
       transmit, reuse, repost, transfer, or use any
       content or information from this service for
       commercial purposes without prior written
       permission.
      </useRestrictions>
      <conceptualSchema schemaLocation=
       "http://example.net/schema/darwin2.xsd">
       http://digir.net/schema/conceptual/darwin/2003/1.0
      </conceptualSchema>
      <recordIdentifier>HDS</recordIdentifier>
      <recordBasis>specimen</recordBasis>
      <numberOfRecords>7887</numberOfRecords>
      <dateLastUpdated>
       2004-05-06T13:56:00-0500
      </dateLastUpdated>
      <minQueryTermLength>3</minQueryTermLength>
      <maxSearchResponseRecords>
       100
      </maxSearchResponseRecords>
      <maxInventoryResponseRecords>
       1000
      </maxInventoryResponseRecords>
    </resource>
  </provider>
</metadata>
```

```
  </content>
  <diagnostics>
    <diagnostic code="STATUS_INTERVAL" severity="info">
     3600
    </diagnostic>
    <diagnostic code="STATUS_DATA" severity="info">
     1,0,0
    </diagnostic>
  </diagnostics>
</response>
```

DiGIR inventory operation

Inventory requests ask for distinct values of a specific concept. An optional filter can be used, and a concept from one of the mapped conceptual schemas must be specified. It accepts an additional element that can be used to turn on counting of the total number of distinct values.

Sample DiGIR inventory request:

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="http://digir.net/schema/protocol/2003/1.0"
         xmlns:darwin=
  "http://digir.net/schema/conceptual/darwin/2003/1.0">
  <header>
    <version>1.0.0</version>
    <sendTime>2004-07-20T09:15:30-0500</sendTime>
    <source>127.0.0.1</source>
    <destination resource="HDS">
     http://example.net/provider/DiGIR.php
    </destination>
    <type>inventory</type>
  </header>
  <inventory>
    <filter>
      <equals>
        <darwin:Collector>Thomas, J.</darwin:Collector>
      </equals>
    </filter>
    <darwin:Genus/>
    <count>true</count>
  </inventory>
</request>
```

Inventory responses include record elements enclosing each distinct value from the concept. A count of the number of occurrences of each distinct value is provided, and a total count for the number of distinct values is returned in the diagnostics section if this was requested. According to the protocol schema, paging is not possible in inventory requests, although some provider implementations do offer this feature.

Sample DiGIR inventory response:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>$Revision: 1.96 $</version>
    <sendTime>2004-07-20T09:15:32-0500</sendTime>
    <source resource="HDS">
     http://example.net:80/provider/DiGIR.php
    </source>
    <destination>127.0.0.1</destination>
    <type>inventory</type>
  </header>
  <content xmlns:darwin=
      "http://digir.net/schema/conceptual/darwin/2003/1.0"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <record>
      <darwin:Genus count="4">Agave</darwin:Genus>
    </record>
    <record>
      <darwin:Genus count="8">Passiflora</darwin:Genus>
    </record>
    <record>
      <darwin:Genus count="26">Rubus</darwin:Genus>
    </record>
    <record>
      <darwin:Genus count="3">Solanum</darwin:Genus>
    </record>
  </content>
  <diagnostics>
    <diagnostic code="STATUS_INTERVAL" severity="info">
     3600
    </diagnostic>
    <diagnostic code="STATUS_DATA" severity="info">
     1,0,1
    </diagnostic>
```

```
    <diagnostic code="MATCH_COUNT" severity="info">
     4
    </diagnostic>
    <diagnostic code="RECORD_COUNT" severity="info">
     4
    </diagnostic>
    <diagnostic code="END_OF_RECORDS" severity="info">
     true
    </diagnostic>
  </diagnostics>
</response>
```

DiGIR search operation

Search requests have a mandatory <filter> element and an optional <records> element to specify what should be returned and how records should be structured in results. The record structure is optional because some requests may simply ask for the total number of records that match a filter condition. But when they are needed, the whole structure can be directly specified, as in the next example, or it can be referenced through an optional "schemaLocation" attribute in the <structure> element. Search requests also accept an additional element that can be used to turn on counting of the total number of matched records. Paging is possible by using the attributes "limit" and "start".

Sample DiGIR search request:

```
<?xml version="1.0" encoding="UTF-8"?>
<request
  xmlns="http://digir.net/schema/protocol/2003/1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:darwin=
   "http://digir.net/schema/conceptual/darwin/2003/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <header>
    <version>1.0.0</version>
    <sendTime>2004-07-20T09:56:00-0500</sendTime>
    <source>127.0.0.1</source>
    <destination resource="HDS">
     http://example.net/provider/DiGIR.php
    </destination>
    <type>search</type>
  </header>
  <search>
```

```
    <filter>
      <and>
        <equals>
          <darwin:Collector>Thomas, J.</darwin:Collector>
        </equals>
        <equals>
          <darwin:Genus>Rubus</darwin:Genus>
        </equals>
      </and>
    </filter>
    <records limit="3" start="0">
      <structure>
        <xsd:element name="record"
                     minOccurs="0"
                     maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="darwin:InstitutionCode"/>
              <xsd:element ref="darwin:CollectionCode"/>
              <xsd:element ref="darwin:CatalogNumber"/>
              <xsd:element ref="darwin:ScientificName"/>
              <xsd:element ref="darwin:Latitude"/>
              <xsd:element ref="darwin:Longitude"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </structure>
    </records>
    <count>true</count>
  </search>
</request>
```

Search responses include the requested records that have matched the query. They are formatted according to the given record structure.

Sample DiGIR search response:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://digir.net/schema/protocol/2003/1.0">
  <header>
    <version>$Revision: 1.96 $</version>
    <sendTime>2004-07-20T09:56:03-0500</sendTime>
    <source resource="HDS">
     http://example.net:80/provider/DiGIR.php
```

```
  </source>
  <destination>127.0.0.1</destination>
  <type>search</type>
</header>
<content xmlns:darwin=
    "http://digir.net/schema/conceptual/darwin/2003/1.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <record>
    <darwin:InstitutionCode>INS</darwin:InstitutionCode>
    <darwin:CollectionCode>COL</darwin:CollectionCode>
    <darwin:CatalogNumber>27694</darwin:CatalogNumber>
    <darwin:ScientificName>
     Rubus brasiliensis
    </darwin:ScientificName>
    <darwin:Latitude>-23.6625</darwin:Latitude>
    <darwin:Longitude>-46.7738</darwin:Longitude>
  </record>
  <record>
    <darwin:InstitutionCode>INS</darwin:InstitutionCode>
    <darwin:CollectionCode>COL</darwin:CollectionCode>
    <darwin:CatalogNumber>27907</darwin:CatalogNumber>
    <darwin:ScientificName>
     Rubus brasiliensis
    </darwin:ScientificName>
    <darwin:Latitude>-23.412</darwin:Latitude>
    <darwin:Longitude>-46.7899</darwin:Longitude>
  </record>
  <record>
    <darwin:InstitutionCode>INS</darwin:InstitutionCode>
    <darwin:CollectionCode>COL</darwin:CollectionCode>
    <darwin:CatalogNumber>27908</darwin:CatalogNumber>
    <darwin:ScientificName>
     Rubus urticaefolius
    </darwin:ScientificName>
    <darwin:Latitude>-23.412</darwin:Latitude>
    <darwin:Longitude>-46.7899</darwin:Longitude>
  </record>
</content>
<diagnostics>
  <diagnostic code="STATUS_INTERVAL" severity="info">
   3600
  </diagnostic>
  <diagnostic code="STATUS_DATA" severity="info">
```

```
     1,1,1
    </diagnostic>
    <diagnostic code="MATCH_COUNT" severity="info">
     26
    </diagnostic>
    <diagnostic code="RECORD_COUNT" severity="info">
     3
    </diagnostic>
    <diagnostic code="END_OF_RECORDS" severity="info">
     false
    </diagnostic>
  </diagnostics>
</response>
```

When there are NULL values or unmapped concepts that are present in record structures, the corresponding elements in responses have the "xsi:nil" attribute set to "true".

DiGIR filter encoding

The DiGIR protocol has its own generic query mechanism defined through the filter element, and it does not assume anything either about how data is locally stored (Relational database, XML files, Object database, etc) or about its particular query language (SQL, XPath, etc). The task of translating between a DiGIR query and a local database query language is completely delegated to the DiGIR Provider implementation.

DiGIR filters can directly contain a single comparative expression like:

```
<filter>
  <equals>
    <darwin:ScientificName>
     Rubus brasiliensis
    </darwin:ScientificName>
  </equals>
</filter>
```

Comparison operators include <equals>, <notEquals>, <lessThan>, <lessThanOrEquals>, <greaterThan>, <greaterThanOrEquals>, and <like>. All of them are used to represent simple comparisons between a concept and a single value.

Comparisons with NULL values can be performed using the "xsi:nil" attribute, which is already part of the XML Schema definition language:

```
<filter>
  <equals>
    <darwin:Latitude xsi:nil="true"/>
  </equals>
</filter>
```

Another special comparison operator <in> can be used to represent comparisons between a single concept and a list of values:

```
<filter>
  <in>
    <list>
      <darwin:Genus>Physalis</darwin:Genus>
      <darwin:Genus>Rubus</darwin:Genus>
      <darwin:Genus>Byrsonima</darwin:Genus>
    </list>
  </in>
</filter>
```

Comparison expressions can be combined through logical operators. There are four logical operators available: <and>, <andNot>, <or>, <orNot>. All of them are defined as binary operators, therefore it is necessary to nest logical operations if they involve more than two comparisons:

```
<filter>
  <and>
    <equals>
      <darwin:ScientificName>
       Rubus brasiliensis
      </darwin:ScientificName>
    </equals>
    <or>
      <greaterThan>
        <darwin:YearIdentified>1980</darwin:YearIdentified>
      </greaterThan>
      <like>
        <darwin:IdentifiedBy>Koch%</darwin:IdentifiedBy>
      </like>
    </or>
  </and>
</filter>
```

It is interesting to mention that concepts from different conceptual schemas can also be combined in the same filter. The namespace prefix in concept element names is always used to identify their schema.

DiGIR record structure

DiGIR search requests can also specify which content elements should be returned and how they should be structured. This is done through a record structure specification. The DiGIR record structure uses a simplified subset of XML Schema definitions, where content elements are referenced by the XML "ref" attribute:

```
<structure>
  <xsd:element name="record"
               minOccurs="0"
               maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="darwin:InstitutionCode"/>
        <xsd:element ref="darwin:CollectionCode"/>
        <xsd:element ref="darwin:CatalogNumber"/>
        <xsd:element ref="darwin:ScientificName"/>
        <xsd:element ref="darwin:Latitude"/>
        <xsd:element ref="darwin:Longitude"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</structure>
```

The syntax of record structures is not specified by the DiGIR protocol schema. The <structure> element is actually defined there as an XML complex type which accepts any content, although the definition of a record element is mandatory.

The <record> element is the basis for looping, paging, and limiting records. With a relational database, the usual approach is to have an underlying "root table" defined during resource configuration whose records are bound to the <record> element.

Since content elements are being referenced, their cardinality should be taken from the corresponding element definitions in the conceptual schema. Another possibility using record structures is to request further grouping of specific elements, as shown in the next example.

```
<structure>
  <xsd:element name="record"
               minOccurs="0"
               maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="darwin:InstitutionCode"/>
        <xsd:element ref="darwin:CollectionCode"/>
        <xsd:element ref="darwin:CatalogNumber"/>
        <xsd:element ref="darwin:ScientificName"/>
        <xsd:element name="coordinates">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="darwin:Latitude"/>
              <xsd:element ref="darwin:Longitude"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</structure>
```

And finally, the same structures could be remotely referenced by using the "schemaLocation" attribute.

```
<structure schemaLocation=
           "http://example.net/searchStructure.xml"/>
```

One important point regarding the way record structures are defined and used is that results will always be either custom grouping elements or instances of concept elements. It is not possible to rename elements (elements using the XML "ref" attribute cannot override names of the referenced elements[19]) and it is not possible to return values as XML attributes (attribute declarations cannot reference elements, they can only reference global attribute declarations[20]). So, although the record structure provides some flexibility about how data should be returned, it is not flexible enough to produce a result which could be validated against external XML Schema representing content elements which are independent from the protocol.

[19] http://www.w3.org/TR/xmlschema-1/#section-Constraints-on-XML-Representations-of-Element-Declarations

[20] http://www.w3.org/TR/xmlschema-1/#Attribute_Declaration_details

Concerning the use of modularized conceptual schemas, elements from a record structure can perfectly reference concepts from different schemas inside the same structure. Conceptual schemas are always associated with the namespace prefix inside the "ref" attribute.

DiGIR conceptual binding

As shown in previous examples, content elements from conceptual schemas are either directly used in messages (as it happens in filter comparison expressions and in inventory requests) or referenced (as it happens in record structure specifications).

It was a protocol design decision that each content element from a conceptual schema should be derived from one of three abstract elements named <searchableData>, <returnableData>, or <searchableReturnableData>, which are defined in the main DiGIR protocol schema. Therefore, a concept could be searchable but not returnable, as it happens with the "BoundingBox" element from DarwinCore. There could also be returnable elements which are not searchable (maybe in the case of binary data types).

This approach ensures that a filter expression can only use searchable concepts, and that an inventory operation can only request values from returnable concepts. Record structures should in theory have a similar restriction, in the sense that content elements should only point to returnable concepts, but as mentioned before, this is not being validated by the protocol.

The original approach also considered the possibility of adding another inheritance layer to classify concepts according to their data types, but that would bring an additional complexity since XML Schema doesn't allow for multiple inheritances. In that case, concepts would be derived from "alphaSearchableData", or "numericSearchableData", and so on. And that would ensure that string operations would only be performed against concepts that are searchable strings. But this has been discarded.

The ways concepts are used in the protocol bring two consequences for conceptual schemas. The first one is that all concepts should be definitely bound to the protocol since they need to be classified as searchable and/or returnable. Binding is done through the XML Schema "substitutionGroup" technique, and therefore it is not possible to use a completely independent conceptual schema.

Another consequence is that the "substitutionGroup" technique requires that both the "head" element (abstract concept types in this case) and the member elements (the "real" concept elements) must always be declared in the global scope of the schema[21].

[21] http://www.w3.org/TR/xmlschema-1/#Element_Equivalence_Class

And due to the way DiGIR treats unmapped concepts and NULL values in results, all non-mandatory elements from a conceptual schema need to be declared as "nillable".

So, a DiGIR-compatible conceptual schema is always a flat list of elements which are derived from one of the three abstract data elements defined by the protocol. And although this approach brings some restrictions, it also enables additional validation regarding the use of concepts in the protocol.

DiGIR operation calls

Unfortunately there is no official specification about how messages should be exchanged using DiGIR. As a possible reference, the original provider implementation accepts requests in several ways:

- As a single GET or POST parameter called "request" or "doc" containing either the DiGIR XML request document or a URL pointing to a request document.
- A single GET or POST default parameter containing a URL pointing to a request document.
- A set of GET or POST parameters containing parts of a request document: "filter", "resource", "startrec", "maxrecs", "clientkey", "recordstruct", "sortstruct", and "countrecs".

5.1.4 Existing DiGIR software

At this moment, there are already several software supporting the DiGIR protocol. The implementations that have been developed as part of the original project are the most widely used. They are all freely available from the sourceforge site under a project called "digir", which includes:

- DiGIR PHP provider: Component responsible for connecting local data sources to DiGIR networks. It has been implemented in PHP and works in conjunction with a web server. Although considered stable and fully functional, it does not support requests destinated to more than one resource at the same time. Local data sources are restricted to Relational Databases. There are drivers to most vendors. Besides the files available from the sourceforge site, packages including web server and UDDI registration capabilities can be obtained from the GBIF site[22] for a number of different platforms. The PHP provider is also being distributed as part of the Specify[23] software.

[22] http://www.gbif.org/
[23] http://www.specifysoftware.org/

- DiGIR portal engine: Component responsible for encapsulating a DiGIR network of distributed data providers. It receives requests from user interfaces or search agents, distributes them to the corresponding data providers, and integrates the responses into a single document before sending back to the requestor. It was implemented in java and runs with Tomcat. Provider access points are either manually configured or taken from UDDI registries.
- DiGIR presentation layer: Component responsible for offering a user interface to query a DiGIR network. It communicates with a portal engine. It was also implemented in java and runs in another Tomcat instance.

Besides the original provider software, there are two other known provider implementations:

- GBIF data repository tool[24]: A package including Zope with a Python implementation of a DiGIR provider, and a MySQL database with pre-configured tables.
- Biota[25] provider: A functional java implementation of a provider which is currently being tested. It should be part of the next versions of the Biota distribution.

And besides the original portal engine, there is another portal implementation using PHP, known as the PHP DiGIR Portal[26]. It includes a user interface and it communicates with several providers using the socket-based script available from the original PHP provider implementation.

And finally, there are some additional libraries which are also related to DiGIR:

- GBIF portal library[27]: The GBIF data portal uses java libraries to directly communicate with providers.
- perl client library: The speciesLink search interface[28] uses a perl library to communicate with a portal engine. Only metadata and search messages are handled.

[24] http://circa.gbif.net/Public/irc/gbif/ict/library?l=/download_gbif_tools/gbif_repository
[25] http://viceroy.eeb.uconn.edu/biota
[26] http://digirportal.berkeley.edu/
[27] http://sourceforge.net/projects/gbif
[28] http://splink.cria.org.br/simple_search

- Globus DiGIR wrapper: The SEEK project[29] has also developed a web service on top of the Globus toolkit[30] to communicate with a portal engine.

5.2 BioCASe protocol

5.2.1 Brief history

The BioCASe protocol[31] was developed by the Biodiversity Informatics department of the Botanical Garden Botanical Museum Berlin[32] for the BioCASE project[33] in 2003 with similar goals as the DiGIR project. But instead of using Darwin Core as the default conceptual schema BioCASE aimed to implement a distributed and heterogenous network using ABCD[34] which is a rather complex and hierarchically structured xml standard for exchanging biological specimen or observation data. Initially it was intended to use the DiGIR protocol, but very soon it was discovered that DiGIR was not suitable for these needs, as its conceptual binding using XML substitution groups required for the conceptual schema:

- The insertion of protocol specific parts into the conceptual schema
- A flat list of global element declarations

The only possible solution would have been to adopt a modified ABCD schema which contained a list of all its hierarchical elements as global elements. At the TDWG meeting in Indaiatuba 2002 such a list with a couple of thousand elements was presented and was not considered acceptable. From the BioCASE point of view, the protocol should not influence conceptual schema design in any major way.

Based on the DiGIR protocol a new protocol with a different conceptual binding was created using simple XPaths[35] to refer to elements of the conceptual schema. This allowed for using any hierarchical schema, but has the disadvantage of losing strong XML based validation all the way back into the conceptual schema. Additionally some minor changes were made to the protocol with a new capabilities request type being the most prominent one. All software was made publicly available through the Mozilla public license.

[29] http://seek.ecoinformatics.org
[30] http://www.globus.org/toolkit/
[31] http://www.biocase.org/dev/protocol
[32] http://www.bgbm.org/BioDivInf
[33] http://www.biocase.org
[34] http://www.bgbm.org/TDWG/CODATA/Schema/
[35] http://www.w3.org/TR/xpath

5.2.2 Description

With the DiGIR protocol as its basis, BioCASe[36] is also an XML-based protocol over HTTP to retrieve data from independent and heterogeneous databases. To achieve a uniform virtual view from network participants, each BioCASe data source needs to choose one or more conceptual schemas (or data exchange standards) and then map some fields from a local database against (some of) the elements of those schemas.

Different conceptual schemas can be defined by each network, and therefore the protocol can be used by any community without regard to discipline. Conceptual schemas don't need to contain any protocol specific add-ons and apart from recursive structures any XML Schema structure including choices and repeating elements are supported. Each xml element or attribute is regarded as a concept that can be referred to via an XPath-based conceptual binding. Two examples of such schemas in current use are the Access to Biological Collection Data standard[37] and the BioCASE metaprofile for describing collections[38].

The BioCASE network consists of a central registry, a message broker service called the Unitloader[39] and providers running the BioCASE provider software[40].

The unitloader software is a java class generating protocol documents and distributing them via threads to the list of "datasource" services requested. It is not directly connected to a registry but needs a list of desired providers to be queried.

A provider is regarded as a host which runs the BioCASE provider software. This is a collection of software tools for querying local datasources, for graphically configuring datasources[41] and of course a number of wrappers that act as datasource services. Each datasource is restricted to a single database but may be configured for any number of accepted conceptual schemas. This way it is possible to work with ABCD and DarwinCore for example. Contrary to DiGIR there is no provider service giving access to all the datasources, but instead each individual datasource has its own access point URL.

[36] http://www.bgbm.org/biodivinf/Schema/protocol_1_3.xsd
[37] http://www.bgbm.org/TDWG/CODATA/Schema/ABCD-1.20.xsd
[38] http://www.bgbm.org/biodivinf/Schema/BioCASE-MetaProfile-123.xsd
[39] http://www.biocase.org/dev/unitloader
[40] http://www.biocase.org/dev/provider/
[41] http://www.biocase.org/dev/configtool

5.2.3 Technical details

General message format

A BioCASe message can be a request or a response - both made up of 3 major parts.

The "header" section gives some information about the origin and destination of the message like a list of all services that have been involved in the chain of the message flow together with a timestamp. The software versions involved in creating the message are also specified.

The "content" section only exists for responses and search or scan requests. It carries the requested response data in the format specified by the conceptual schema or specifies the request to be carried out by a datasource. It also contains attributes that hold status information for paging and record counts such as the overall matched records in the database and how many records were returned or "dropped". This is the number of records that matched the query but did not validate against the conceptual schema because of missing or erroneous mandatory data.

The "diagnostics" section is used for debugging information and to provide warnings or error messages.

Request and response types

Similar to DiGIR there are 3 different message types supported. But instead of a metadata operation there is a slightly different capabilities operation. The DiGIR inventory type is called "scan" (as it was called in its original proposal at the time the BioCASe protocol was established).

BioCASe capabilities operation

A capabilities request allows a client to get information about which concepts are mapped (defined) in a provider database. This request type returns a list of xpaths identifying all mapped concepts. There are no parameters involved in a capabilities request and this is also the default response for the provider software if no operation was specified.

Sample BioCASe capabilities request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<request xmlns="http://www.biocase.org/schemas/protocol/1.3">
  <header>
    <version software="unitloader">0.98</version>
    <sendTime>2003-09-25T17:02:45+02:00</sendTime>
    <source>198.14.7.54</source>
    <source>192.168.1.153</source>
    <type>capabilities</type>
  </header>
</request>
```

Sample BioCASe capabilities response:

```
<?xml version='1.0' encoding='UTF-8'?>
<response xmlns='http://www.biocase.org/schemas/protocol/1.3'>
  <header>
    <version software='Python Interpreter'>
     2.3 (\#46, Jul 29 2003, 18:54:32)
     [MSC v.1200 32 bit (Intel)]
    </version>
    <version software='Wrapper'>1.4.0 alpha</version>
    <version software='OS'>nt</version>
    <sendTime>2003-09-25T17:02:46+02:00</sendTime>
    <source>192.168.1.12</source>
    <destination>192.168.1.153</destination>
    <destination>198.14.7.54</destination>
    <type>capabilities</type>
  </header>
  <content>
    <capabilities>
      <SupportedSchemas request='true'
       namespace='http://www.tdwg.org/schemas/abcd/1.2'
       response='true'>
       <Concept>
        /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/DateSupplied
       </Concept>
       <Concept>
        /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Description
       </Concept>
```

```
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/CopyrighDeclaration
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/IPRDeclaration
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/LegalOwner/URLs/URL
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/RightsURL
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/SpecificRestrictions
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Rights/TermsOfUse
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Statements/Acknowledgement
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Statements/Disclaimer
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Statements/LogoURL
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Statements/StatementURL
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Supplier/Addresses/Address
        </Concept>
```

```
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Supplier/Person/PersonName
        </Concept>
        <Concept>
         /DataSets/DataSet/DatasetDerivations/
DatasetDerivation/Supplier/URLs/URL
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceExpiryDate
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceInstitutionCode
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceLastUpdatedDate
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceName
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceNumberOfRecords
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceVersion
        </Concept>
        <Concept>
         /DataSets/DataSet/OriginalSource/
SourceWebAddress
        </Concept>
        <Concept>
         /DataSets/DataSet/Units/Unit/
CollectorsFieldNumber
        </Concept>
        <Concept>
         /DataSets/DataSet/Units/Unit/Gathering/
GatheringAgents/GatheringAgent/Person/PersonName
        </Concept>
```

```
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/
GatheringDateTime/DateText
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/
GatheringDateTime/ISODateTimeBegin
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Altitude/MeasurementAtomized/MeasurementLowerValue
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Altitude/MeasurementAtomized/MeasurementScale
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Aspect/AspectText
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
BiotopeData/BiotopeText
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Country/CountryName
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Country/ISO2Letter
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
LocalityText
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
NamedAreas/NamedArea/NamedAreaName
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Slope/MeasurementAtomized/MeasurementLowerValue
       </Concept>
```

```
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/GatheringSite/
Slope/MeasurementAtomized/MeasurementScale
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Gathering/
Project/ProjectTitle
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/Identifier/IdentifierPersonName
        /PersonName
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/AuthorString
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/HigherTaxa/HigherTaxon
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/NameAuthorYearString
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/ScientificNameAtomized/
Botanical/FirstEpithet
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
/Identification/TaxonIdentified/ScientificNameAtomized/
Botanical/Genus
       </Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/ScientificNameAtomized/
Botanical/Rank</Concept>
       <Concept>
        /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/ScientificNameAtomized/
Botanical/SecondEpithet
       </Concept>
```

```
        <Concept>
         /DataSets/DataSet/Units/Unit/UnitID
        </Concept>
        <Concept>
         /DataSets/DataSet/Units/Unit/UnitStateDomain/
SpecimenUnit/UnitPreparation/PreparationType
        </Concept>
       </SupportedSchemas>
       <SupportedSchemas request='true'
          namespace='http://www.namespacetbd.org/darwin2'
          response='true'>
         <Concept>/RecordSet/Record/CatalogNumber</Concept>
         <Concept>/RecordSet/Record/CollectionCode</Concept>
         <Concept>/RecordSet/Record/Collector</Concept>
         <Concept>/RecordSet/Record/Country</Concept>
         <Concept>/RecordSet/Record/DateLastModified</Concept>
         <Concept>/RecordSet/Record/Family</Concept>
         <Concept>/RecordSet/Record/FieldNumber</Concept>
         <Concept>/RecordSet/Record/Genus</Concept>
         <Concept>/RecordSet/Record/IdentifiedBy</Concept>
         <Concept>/RecordSet/Record/InstitutionCode</Concept>
         <Concept>/RecordSet/Record/ScientificName</Concept>
         <Concept>/RecordSet/Record/Subspecies</Concept>
       </SupportedSchemas>
     </capabilities>
  </content>
  <diagnostics>
    <diagnostic>OK</diagnostic>
  </diagnostics>
</response>
```

The previous example shows the <source> element occurring twice in the header, with the original source being the first source element of the sequence. The response lists the mapped concepts for 2 configured conceptual schemas, DarwinCore and ABCD. The schemas are identified by their namespace only.

BioCASe scan operation

A scan request concentrates on one concept referenced by an xpath to the element of the respective conceptual schema. It is essentially a select distinct in SQL and returns all unique values for this concept. In contrast to the DiGIR inventory it is not possible to specify a filter for a scan.

Here is an abbreviated scan request example asking for distinct values of botanical genera:

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="http://www.biocase.org/schemas/protocol/1.3">
  <header>
    <sendTime>2003-09-25T17:02:45+02:00</sendTime>
    <source>198.14.7.54</source>
    <type>scan</type>
  </header>
  <scan>
    <requestFormat>
     http://www.tdwg.org/schemas/abcd/1.2
    </requestFormat>
    <concept>
     /DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/ScientificNameAtomized/
Botanical/Genus
    </concept>
  </scan>
</request>
```

Scan responses deliver each sorted distinct value in a <value> element and provide the total number of values in the content attribute "recordCount":

```
<?xml version='1.0' encoding='UTF-8'?>
<response
   xmlns='http://www.biocase.org/schemas/protocol/1.3'
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xsi:schemaLocation=
    'http://www.biocase.org/schemas/protocol/1.3
  http://www.bgbm.org/biodivinf/schema/protocol_1_3.xsd'>
  <header>
    <version software='Python Interpreter'>
     2.3 (\#46, Jul 29 2003, 18:54:32)
     [MSC v.1200 32 bit (Intel)]
    </version>
    <version software='PyWrapper'>0.98a</version>
    <version software='DB module'>
     MS SQL Server module v0.91, using mxODBC 2.0.1
    </version>
    <version software='OS'>nt</version>
    <sendTime>2003-09-25T17:02:46+02:00</sendTime>
```

```
    <source>192.168.1.12</source>
    <destination>198.14.7.54</destination>
    <type>scan</type>
  </header>
  <content recordDropped='0'
           recordStart='0'
           recordCount='188'>
    <scan>
      <value>Acantholimon</value>
      <value>Achillea</value>
      <value>Aethionema</value>
      <value>Ajuga</value>
      <value>Alchemilla</value>
      <value>Alkanna</value>
      <value>Allium</value>
      <value>Alopecurus</value>
      <value>Alyssum</value>
       ...
      <value>Viola</value>
      <value>Xeranthemum</value>
      <value>Ziziphora</value>
    </scan>
  </content>
  <diagnostics>
    <diagnostic>OK</diagnostic>
  </diagnostics>
</response>
```

BioCASe search operation

The BioCASe protocol requires a filter to be specified in a search request as well as the conceptual schema to be used for responding ("responseFormat"). It is possible to use concepts taken from a different schema than the response schema for creating the query filter which is specified by the "requestFormat" namespace. To only count matching records while not returning any other data the optional element "count" should be set to true. The search is stateful. Paging is supported by using the "start" and "limit" attributes which default to 0.

Sample BioCASe search request:

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="http://www.biocase.org/schemas/protocol/1.3">
  <header>
```

```
    <version software="unitloader">0.98</version>
    <sendTime>2003-09-25T17:02:45+02:00</sendTime>
    <source>198.14.7.54</source>
    <source>192.168.1.153</source>
    <type>search</type>
  </header>
  <search>
    <requestFormat>
     http://www.tdwg.org/schemas/abcd/1.2
    </requestFormat>
    <responseFormat start="0" limit="2">
     http://www.tdwg.org/schemas/abcd/1.2
    </responseFormat>
    <filter>
      <like path= "/DataSets/DataSet/Units/Unit/
/Identifications/Identification/TaxonIdentified/
NameAuthorYearString">
       Ast*
      </like>
    </filter>
    <count>false</count>
  </search>
</request>
```

The response using the ABCD schema is as follows:

```
<?xml version='1.0' encoding='latin1'?>
<response xmlns=
   'http://www.biocase.org/schemas/protocol/1.3'>
  <header>
    <version software='Python Interpreter'>
     2.3 (\#46, Jul 29 2003, 18:54:32)
     [MSC v.1200 32 bit (Intel)]
    </version>
    <version software='Wrapper'>1.4.0 alpha</version>
    <version software='DB module'>
     MS SQL Server module v0.91, using mxODBC 2.0.1
    </version>
    <version software='OS'>nt</version>
    <sendTime>2003-09-25T17:02:47+02:00</sendTime>
    <source>192.168.1.12</source>
    <destination>192.168.1.153</destination>
    <destination>198.14.7.54</destination>
```

```
  <type>search</type>
</header>
<content recordDropped='0'
         recordCount='2'
         recordStart='0'
         totalSearchHits='122'>
  <DataSets xmlns='http://www.tdwg.org/schemas/abcd/1.2'>
    <DataSet>
      <OriginalSource>
        <SourceInstitutionCode>B</SourceInstitutionCode>
        <SourceName>PonTaurus DB</SourceName>
        <SourceLastUpdatedDate>
         2001-03-01
        </SourceLastUpdatedDate>
        <SourceNumberOfRecords>3068</SourceNumberOfRecords>
      </OriginalSource>
      <DatasetDerivations>
        <DatasetDerivation>
          <DateSupplied>2003-08-11</DateSupplied>
          <Supplier>
            <Organisation>
              <OrganisationName>
               Botanic Garden and
               Botanical Museum Berlin-Dahlem
              </OrganisationName>
            </Organisation>
            <Addresses>
              <Address>
                Königin Luise Str. 6-8, D-14191
                Berlin, Germany
              </Address>
            </Addresses>
            <EmailAddresses>
              <EmailAddress>
               m.doering@bgbm.org
              </EmailAddress>
            </EmailAddresses>
            <URLs>
              <URL>http://www.bgbm.org</URL>
            </URLs>
          </Supplier>
          <Rights>
            <LegalOwner>
              <Person>
```

```
          <PersonName>Markus Döring</PersonName>
        </Person>
        <Addresses>
          <Address>
           Königin Luise Str. 6-8, D-14191
           Berlin, Germany
          </Address>
        </Addresses>
        <EmailAddresses>
          <EmailAddress>
           m.doering@bgbm.org
          </EmailAddress>
        </EmailAddresses>
      </LegalOwner>
    </Rights>
  </DatasetDerivation>
</DatasetDerivations>
<Units>
  <Unit>
    <UnitID>1008_1</UnitID>
    <Identifications>
      <Identification>
        <TaxonIdentified>
          <HigherTaxa>
            <HigherTaxon>Fabaceae</HigherTaxon>
          </HigherTaxa>
          <NameAuthorYearString>
           Astragalus haussknechtii Bunge
          </NameAuthorYearString>
          <AuthorString>Bunge</AuthorString>
          <ScientificNameAtomized>
            <Botanical>
              <Genus>Astragalus</Genus>
              <FirstEpithet>
               haussknechtii
              </FirstEpithet>
            </Botanical>
          </ScientificNameAtomized>
        </TaxonIdentified>
        <Identifier>
          <IdentifierPersonName>
            <PersonName>Markus Döring</PersonName>
          </IdentifierPersonName>
        </Identifier>
```

```
  </Identification>
</Identifications>
<UnitStateDomain>
  <SpecimenUnit>
    <UnitPreparation>
      <PreparationType>
       dried and pressed
      </PreparationType>
    </UnitPreparation>
  </SpecimenUnit>
</UnitStateDomain>
<Gathering>
  <GatheringDateTime>
    <DateText>30-07-1999</DateText>
    <ISODateTimeBegin>
     1999-07-30
    </ISODateTimeBegin>
  </GatheringDateTime>
  <GatheringAgents>
    <GatheringAgent>
      <Person>
        <PersonName>Markus Döring</PersonName>
      </Person>
    </GatheringAgent>
  </GatheringAgents>
  <Project>
    <ProjectTitle>PonTaurus 1999</ProjectTitle>
  </Project>
  <GatheringSite>
    <LocalityText>
     upper Arpalik (Maden) Deresi, slopes along
     the road from Darbogaz to tarn Karagöl.
     N37°27´47´´ O34°36´82´´
    </LocalityText>
    <Country>
      <CountryName>Turkey</CountryName>
      <ISO2Letter>TR</ISO2Letter>
    </Country>
    <NamedAreas>
      <NamedArea>
        <NamedAreaName>Nigde</NamedAreaName>
      </NamedArea>
    </NamedAreas>
    <Altitude>
```

```
        <MeasurementAtomized>
          <MeasurementScale>Meter</MeasurementScale>
          <MeasurementLowerValue>
           2080
          </MeasurementLowerValue>
        </MeasurementAtomized>
      </Altitude>
      <BiotopeData>
        <BiotopeText>
         Grazed swards and open
         thorn-cushion communities.
        </BiotopeText>
      </BiotopeData>
      <Aspect>
        <AspectText>N</AspectText>
      </Aspect>
      <Slope>
        <MeasurementAtomized>
          <MeasurementScale>
           decimal degree
          </MeasurementScale>
          <MeasurementLowerValue>
           27
          </MeasurementLowerValue>
        </MeasurementAtomized>
      </Slope>
    </GatheringSite>
  </Gathering>
  <CollectorsFieldNumber>
   1008
  </CollectorsFieldNumber>
</Unit>
<Unit>
  <UnitID>1008_2</UnitID>
  <Identifications>
    <Identification>
      <TaxonIdentified>
        <HigherTaxa>
          <HigherTaxon>Fabaceae</HigherTaxon>
        </HigherTaxa>
        <NameAuthorYearString>
         Astragalus haussknechtii Bunge
        </NameAuthorYearString>
        <AuthorString>Bunge</AuthorString>
```

```
      <ScientificNameAtomized>
        <Botanical>
          <Genus>Astragalus</Genus>
          <FirstEpithet>
           haussknechtii
          </FirstEpithet>
        </Botanical>
      </ScientificNameAtomized>
    </TaxonIdentified>
    <Identifier>
      <IdentifierPersonName>
        <PersonName>Parolly</PersonName>
      </IdentifierPersonName>
    </Identifier>
  </Identification>
</Identifications>
<UnitStateDomain>
  <SpecimenUnit>
    <UnitPreparation>
      <PreparationType>
       dried and pressed
      </PreparationType>
    </UnitPreparation>
  </SpecimenUnit>
</UnitStateDomain>
<Gathering>
  <GatheringDateTime>
    <DateText>30-07-1999</DateText>
    <ISODateTimeBegin>
     1999-07-30
    </ISODateTimeBegin>
  </GatheringDateTime>
  <GatheringAgents>
    <GatheringAgent>
      <Person>
        <PersonName>Markus Döring</PersonName>
      </Person>
    </GatheringAgent>
  </GatheringAgents>
  <Project>
    <ProjectTitle>PonTaurus 1999</ProjectTitle>
  </Project>
  <GatheringSite>
    <LocalityText>
```

```
      upper Arpalik (Maden) Deresi, slopes along
      the road from Darbogaz to tarn Karagöl.
      N37°27´47´´ O34°36´82´´
    </LocalityText>
    <Country>
      <CountryName>Turkey</CountryName>
      <ISO2Letter>TR</ISO2Letter>
    </Country>
    <NamedAreas>
      <NamedArea>
        <NamedAreaName>Nigde</NamedAreaName>
      </NamedArea>
    </NamedAreas>
    <Altitude>
      <MeasurementAtomized>
        <MeasurementScale>Meter</MeasurementScale>
        <MeasurementLowerValue>
         2080
        </MeasurementLowerValue>
      </MeasurementAtomized>
    </Altitude>
    <BiotopeData>
      <BiotopeText>
       Grazed swards and open
       thorn-cushion communities.
      </BiotopeText>
    </BiotopeData>
    <Aspect>
      <AspectText>N</AspectText>
    </Aspect>
    <Slope>
      <MeasurementAtomized>
        <MeasurementScale>
         decimal degree
        </MeasurementScale>
        <MeasurementLowerValue>
         27
        </MeasurementLowerValue>
      </MeasurementAtomized>
    </Slope>
  </GatheringSite>
</Gathering>
<CollectorsFieldNumber>
 1008
```

```
            </CollectorsFieldNumber>
          </Unit>
        </Units>
      </DataSet>
    </DataSets>
  </content>
  <diagnostics>
    <diagnostic>OK</diagnostic>
  </diagnostics>
</response>
```

BioCASe filter encoding

The <filter> wraps the where clause of a SQL statement. It is a nested structure using different combinations of logical and comparison operators specified as XML tags. The following operators are supported:

- Binary comparison operators: equals, notEquals, lessThan, lessThanOrEquals, greaterThan, greaterThanOrEquals, like
- Unary comparison operators: isNull, isNotNull
- Comparison operators for multiple arguments: in
- Unary logical operators: not
- Binary logical operators: and, or

Here is a more complex filter example illustrating several conditions based on ABCD:

```
<filter>
 <and>
   <like path="/DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/NameAuthorYearString">
      Abies*
   </like>
   <or>
     <like path="/DataSets/DataSet/Units/Unit/Identifications/
Identification/TaxonIdentified/HigherTaxa/HigherTaxon">
      Pinace*
     </like>
     <and>
       <like path="/DataSets/DataSet/Units/Unit/Gathering/
```

```
GatheringSite/Country/CountryName">
        *Russia*
       </like>
       <greaterThan path="/DataSets/DataSet/Units/Unit/
Gathering/GatheringDateTime/ISODateTimeBegin">
         2002-04
       </greaterThan>
     </and>
   </or>
 </and>
</filter>
```

It is not possible to compare 2 concepts with each other. The IN operator comes in a special form and takes a list of values for a concept to be compared to:

```
<filter>
  <in path="/DataSets/DataSet/Units/Unit/
Identifications/Identification/TaxonIdentified/HigherTaxa/
HigherTaxon">
    <value>Pinaceae</value>
    <value>Pinophyta</value>
    <value>Pinophytina</value>
  </in>
</filter>
```

BioCASe conceptual binding

As mentioned before, the conceptual binding (in other words, how concepts from conceptual schemas are referenced from messages) is done by using a simple XPath alike expression. So a protocol message cannot be validated with a simple XML parser but needs a separate program analyzing the message and comparing it to the conceptual schema definitions.

XPaths serve as an identifier for a concept and are not really interpreted as being an XPath. They don't carry any more meaning than serving as an ID for a single concept within a conceptual schema which is identified by its namespace. Therefore only absolute paths using the path separator '/' in combination with the child:: axis are valid. Attributes are denoted by using a final '[@=name]' after the attribute holding element path.

This technique allows working with any non-recursive potentially nested schema.

BioCASe operation calls

There is no existing specification about which CGI parameters are to be used for passing messages. It is recommended to use the parameter 'query' though.

5.2.4 Existing BioCASe software

Software implementing the BioCASe protocol are currently being developed mainly by the BioCASE project[42] itself:

- PyWrapper[43] being the core of the provider software parsing requests and assembling responses. It is written entirely in Python.
- Provider software package including the local querytool[44]. Changes of the PyWrapper will also result in necessary changes for the configtool[45].
- Java based unitloader classes[46] acting as a message broker. They are generating protocol messages via their simple API, sending threaded queries in parallel to several datasources and pooling their responses. They also implement alternative emailing services when required to get complete results with a large timeout.
- The BioCASE collection metadata network, especially the java based metaloader software[47] responsible for harvesting a network of metaprofile data providers.
- The Austrian GBIF node is currently developing another message broker and indexing system for their national network.

[42] http://www.biocase.org/dev
[43] http://www.biocase.org/dev/wrapper/
[44] http://www.biocase.org/dev/provider/
[45] http://www.biocase.org/dev/configtool
[46] http://www.biocase.org/dev/unitloader
[47] http://www.biocase.org/dev/metaloader

6 Other standards and technologies

During the integration process, other standards and technologies were investigated. In particular, XQuery for being a powerful and generic query language that could possibly replace both protocols; OGC standards due to the similar problems that are being addressed by some of their geospatial and location based services; and finally SOAP for its increasing popularity as a means to develop web services.

6.1 XQuery

XQuery is part of a new generation of query languages and it aims at being far more generic than the existing options (SQL for instance). Statements can be expressed in order to act across different data sources at the same time, including structured and semi-structured documents, relational databases, object repositories, or even web services[48]. XQuery could be used not only on search and update operations, but also on transformation and re-structuring of data being retrieved.

In a network of distributed and heterogeneous databases, a possible way to use XQuery would be to write statements based on the data structure of a conceptual schema. Since each data provider would have mapped its local database against the same conceptual schema, those statements could be translated according to the local data structure by wrapper software. Due to the richness and complexity of XQuery, it would be too costly to write complete parsers for it. So instead of having such a parser in the wrapper, the wrapper could pass on the translated queries to a local XML server or middleware software capable of dealing with XQuery statements to be processed.

There could be an interface for advanced users to write their own XQuery statements, but also other simplified interfaces which should be able to automatically generate at least the most used and required types of XQuery statements.

However, one of the consequences of XQuery's flexibility is that it remains unclear how data providers could restrict the amount of data to be returned in a single response. This issue would also affect paging capabilities. Both features (limiting the number of records and paging) are present and considered important in the DiGIR protocol.

The current XQuery specification version, although considered stable, is still in the stage of a working draft, and there are only a few implementations of

[48] http://www.w3.org/TR/xquery-use-cases/

it. Presently, most part of the available XML servers, and XML middleware tools are proprietary software[49], and not all of them support the complete XQuery specification. Using commercial software would certainly be too expensive for networks with so many data providers. And the costs should not only include licenses, but equipment and training as well.

XQuery parsers have been mostly implemented in native XML databases, some of them are free and open source[50]. But this would require frequent data exporting for almost all production databases, and performance could also be a problem when querying larger data sets. Moreover, most database administrators are still unfamiliar with XML database products.

On the other hand, only a few commercial relational databases have released prototype XQuery interfaces[51], which is certainly not enough to cover the diverse infrastructure of existing data providers.

Still, one specific tool has been identified as a potential candidate to be used in a scenario similar to the one previously suggested. The XQuark project[52] is working on a suite of free and open source tools (XQuark Bridge and XQuark Fusion) that seem to implement the whole XQuery specification in a distributed environment. Both tools are platform independent (java), and each local data source needs to map its data structure against a conceptual schema through an object-relational mapping technique. Although the number of supported databases is still limited and the web services interface is not available yet, it would definitely be interesting to conduct a more detailed study to assess the viability of using these tools. With positive conclusions from such a study, the protocol could be substituted by XQuery, and almost all existing software could be replaced by the aforementioned tools or by similar ones.

6.2 OGC standards

The Open Geospatial Consortium (OGC) is an international organisation which is developing standards for geospatial and location based services. At least two of the OGC standards are of particular interest to the protocols being discussed here. The Web Feature Service (WFS)[53] and the Common Catalogue Query Language (CQL)[54].

Web Feature Services expose a set of features (that could be seen as generic

[49] http://www.rpbourret.com/xml/ProdsMiddleware.htm
[50] http://www.rpbourret.com/xml/ProdsNative.htm
[51] http://www.oreillynet.com/lpt/wlg/4991
[52] http://xquark.objectweb.org/index.html
[53] http://www.opengis.org/docs/02-058.pdf
[54] http://www.opengis.org/docs/02-059.pdf

objects) allowing discovery, query, and transformation operations like insert, delete, and update.

Each service can handle one or more feature types (types of objects) and each feature type has an associated XML Schema defining its structure. Feature types and their structure can be discovered through "DescribeFeatureType" requests.

Each feature type is free to have its own custom structure, although the associated XML element definition needs to inherit from a generic GML[55] feature element using the "substitutionGroup" mechanism. And the associated XML type needs to extend a GML "AbstractFeatureType".

Among its properties (which can also be complex) features typically have a geometry-valued property based on one of the GML simple geometry types. Simple geometries are either expressed by coordinates in two dimensions or by a curve delineation which is subject to linear interpolation. OGC has defined many spatial operators that can be used against geographic features: "Equals", "Disjoint", "Touches", "Within", "Overlaps", "Crosses", "Intersects", "Contains", "DWithin", and "Beyond".

Since collecting and observational events fall in the category of geographic features, they could benefit from all these operators. It would definitely be interesting to add similar spatial operators in the new protocol.

Another aspect of features is that each one needs a unique identifier to make database operations possible. Locally unique identifiers are sufficient for this purpose. However, combined with the service scope represented by the service URL, locally unique identifiers can easily become globally unique identifiers. The WFS specification correctly points out the usefulness of being able to directly reference feature instances through a global unique identifier which would include the service URL. So, besides being an identifier it would also be an address, and a default representation of the feature could be easily served by accessing that address. Although recommended, this functionality is left as an implementation specific decision, and it is not covered by the WFS specification. In our case, it would also be interesting to be able to directly reference specimen or other biodiversity objects using a URL address. A useful attribute in this case would be the object version, which is also an optional feature attribute defined in the WFS specification.

Features (or collection of features) can be retrieved through "GetFeature" operations, which accept a filter specification quite similar to DiGIR and BioCASe filters, except for the spatial operators. The OGC filter encoding definition, including comparison, logical and spatial operators, is addressed by the CQL specification, and it includes the possibility of referencing fea-

[55] http://www.opengis.org/docs/02-023r4.pdf

tures through their identifiers.

The comparison operators defined in the filter encoding specification are: "PropertyIsEqualTo", "PropertyIsNotEqualTo", "PropertyIsLessThan", "PropertyIsGreaterThan", "PropertyIsLessThanOrEqualTo", "PropertyIsGreaterThanOrEqualTo", "PropertyIsLike", "PropertyIsNull", and "PropertyIsBetween". And the logical operators are "and", "or", and "not".

The default representation of a feature is its complete structure, but a "GetFeature" request can ask for specific properties. Feature properties are referenced through a simplified XPath expression, both in filters and in "GetFeature" requests for a partial feature structure. The service can also decide about mandatory properties that should always be included in responses regardless of being asked in requests.

The WFS specification covers additional methods which are not essential at this moment for the new protocol being suggested, including feature locking ("GetFeatureWithLock", "LockFeature") and transactions ("Transaction" with insert, update and delete operations). A transaction may include several operations at the same time.

However, according to the WFS specification, a service may not implement all methods. A complete description of the service can be retrieved using a "GetCapabilities" request. A response for such request includes general information about the service, request types supported, the list of available feature types (and operations allowed on each one), and a section about filter encoding capabilities (listing the available functions and operators).

Standards being defined in the biodiversity area, such as ABCD and DarwinCore, could possibly be seen as global feature type definitions. However, in the case of ABCD, the most logical candidate for a feature would be the "unit" element. This would exclude all metadata elements defined above it. In the case of DarwinCore, its elements could be feature properties from a generic "biological record" element. Regardless of these changes, in both cases, the schemas could not be directly used, since a feature type definition should be an element derived from another GML element, and its type should extend an abstract type also defined in GML. As it happens with DiGIR/DarwinCore, WFS binds content definition to the protocol.

Comparing WFS to the protocols being considered in this report, there is one request type which is defined in both DiGIR and BioCASe (named "inventory" and "scan", respectively) that is not addressed by the WFS specification. This request type gives us the possibility to retrieve distinct values of mapped "concepts" (or feature properties in the WFS jargon).

It also doesn't seem to be possible to get a customized view of features, by restructuring their properties, as it can happen now in DiGIR through the record structure specification in search requests.

Considering extensibility, feature properties can be extended by changing the feature type definition, either by including new elements or by extending their XML types. Feature property names may be namespace qualified.

Regarding specific Structured Descriptive Data[56] (SDD) needs, when retrieving data from WFS it doesn't seem to be possible to output "normalized" features referencing terminology or dictionary elements. And it is unclear if the capability of returning instances from different feature types would be enough to represent relationships between current SDD elements. Such capability is normally associated to a request explicitly including multiple queries involving different feature types.

One important information about all OGC standards, including WFS, is that the specifications are being changed[57] to be completely based on SOAP and WSDL to benefit from the growing number of existing SOAP tools and wrappers. Therefore, before seriously considering a migration to the WFS specification it would be wiser to wait for its next version.

6.3 SOAP

SOAP is a generic XML-based protocol for exchanging information in a distributed environment. It defines an extensible messaging framework, including the possibility of making remote method calls, and it also offers a simplified way to encode parameters and return values. Messages can actually be exchanged over several underlying protocols (HTTP, TCP, SMTP) and all its design tries to abstract anything that could be specific to any platform or programming language.

It must be clear that SOAP is only an additional protocol layer over which the existing protocols could be based on. Just by using SOAP would not reconcile the differences between DiGIR and BioCASe. However, an integrated protocol could be based on SOAP or could be independent from it.

At the present moment, a considerable number of web services have been implemented using SOAP, and there are many SOAP wrappers available for most part of the programming languages. This situation makes SOAP almost a natural choice when implementing a web service.

Actually, SOAP and web service are not synonyms. According to W3C's

[56] http://160.45.63.11/Projects/TDWG-SDD/

[57] http://www.opengis.org/press/?page=pressrelease&view=20040318OWS2PR

definition, a "web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols"[58].

Clearly, SOAP is not a requirement to develop a system that satisfies this definition, although it perfectly fits into the web service idea. The current implementations of both DiGIR and BioCASe could already be seen as web services, except for the fact that they still lack a formal interface description. In this context, interfaces are usually described through WSDL[59], which at the moment includes bindings for SOAP messages, HTTP GET and POST verbs, and MIME format. Specific bindings for services that exchange XML messages without using SOAP can be described with WSDL through the mimeXml element[60] from the MIME binding specification.

Nevertheless, there are many similarities between some aspects of the protocols being discussed here and SOAP-based protocols. Both have a similar message structure, with a header and a content (or body) parts. This generic design is usually seen as a disadvantage of using SOAP since it makes it difficult, if not impossible, to validate through XML Schema the content inside the body part. Due to the similar approach, DiGIR and BioCASe have the same restriction at least on search response messages, but the other types of messages can be completely validated through XML Schema. By using SOAP, the content of all types of messages would have this restriction.

From the perspective of protocol specification, moving to SOAP would be relatively easy. The header elements could move to the SOAP header, the content elements could move to the SOAP body part, and the standard fault strategy of SOAP could be used in place of the diagnostics elements.

From an implementation perspective, developers could take advantage of the large amount of existing SOAP wrappers. Depending on the SOAP binding style being used, the wrapper could take complete care of XML parsing, data serialization and deserialization, thus allowing developers to directly use objects and variables in the way they are used to when programming. Generally, this would make both servers and clients easier to develop, although it is unclear if the wrappers would properly deal with more complex data structures such as filter and record structure parameters. But these facilities are mostly related to one specific SOAP binding style, the RPC/encoded. There are four possible binding styles[61] for SOAP. The use of SOAP encoding is actually being discouraged due to interoperability problems between

[58] http://www.w3.org/TR/2002/WD-ws-gloss-20021114/
[59] http://www.w3.org/TR/wsdl.html
[60] http://www.w3.org/TR/wsdl#_mime:mimeXml
[61] http://www-106.ibm.com/developerworks/webservices/library/ws-whichwsdl/

different wrappers. And the RPC binding style is also being discouraged due to scalability problems[62]. So, in case of using SOAP, the recommendation is to use the binding called document/literal to avoid scalability and interoperability issues, and this would mean almost the same amount of work being done now on the developer side to parse and produce messages.

It also remains unclear how performance could be affected by introducing SOAP wrappers in the existing architecture, since part of the message parsing task would move from the current software to the SOAP wrappers. In particular, wrappers for interpreted languages could probably slow down performance, although in the case of PHP the newest version (5.0.0) already includes a built-in SOAP module. This would be better assessed through a more specific case study.

In theory, it could be possible to embed SOAP functionalities in the existing software without using a wrapper, but this would mean an additional protocol level to worry about and to keep up with new specifications. This cost could be significant over the time. So, in case of using SOAP, the recommendation is to use SOAP wrappers. And this would probably cause a considerable impact on code.

It is interesting to mention that depending on the way that SOAP wrappers are used, client or server code tend to be bound to that specific wrapper implementation and also to the web service SOAP binding itself. Therefore, switching between different SOAP wrappers, or between different versions of the same SOAP wrapper, or between different web service bindings could be a non-trivial and time consuming task. There are already more generic wrappers[63] that provide binding independent access to web services by directly using the logical description of the service from the associated WSDL files. In case of using SOAP, the recommendation for developers is to use these generic wrappers, or at least to use a layer between their code and the SOAP wrapper.

Deciding between using SOAP and keeping an application specific XML protocol is not a simple task in this case. Both approaches have advantages and disadvantages. A SOAP-based protocol could benefit from all tools and extensions that are being developed for the SOAP community, for instance security strategies[64] and message brokering tools. Developers would have a large set of wrapping tools available. On the other hand, there are still several interoperability problems between different SOAP wrappers[65], and it is also possible that sometimes SOAP-related issues consume most part of

[62] http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/
[63] http://www-106.ibm.com/developerworks/webservices/library/ws-wsif.html
[64] http://www.nwfusion.com/news/tech/2002/1216techupdate.html
[65] http://www.xmethods.com/soapbuilders/interop.html

the development efforts[66]. Considering mainly the risks related to significant impact on code and performance, the recommendation is to keep the protocol independent from SOAP. If the advantages of using SOAP become clearer in the future, newer versions of the protocol could be based on it.

However, not basing the protocol on SOAP does not prevent to have a SOAP service on top of a message broker. This would be an interesting case study, since message brokers are certainly the kind of service most used by the community. A SOAP service on top of it could perhaps ease the development of user interfaces and also the integration of message brokers with networks that are trying to connect distributed web services of different types.

[66] http://www.artima.com/webservices/articles/whysoapP.html

7 Integration proposal

7.1 Overall strategy

The proposed protocol aims at eliminating the current differences between DiGIR and BioCASe, but also including new features, and incorporating some ideas from other standards. Impact on existing software and networks was also considered, so that the new protocol could be implemented in a relatively short term.

When reviewing other technologies and standards, XQuery has been considered a potential query language to be adopted in the future due to its remarkable flexibility and power. While still being at the stage of a working draft, it is expected to soon become a standard. By using a well-known and globally accepted query language, one could benefit from tools, documentation, and knowledge produced by other communities. But there would certainly be a dependency on XQuery parsers (which are still limited in number and in compliance with the specification). Using XQuery in a distributed and heterogeneous scenario would also require additional research in order to find a way of abstracting local data models. A consistent proposal using XQuery would deserve a deeper study and working prototypes before being suggested.

OGC specifications, namely WFS and CQL, have interesting similarities if compared to both DiGIR and BioCASe. The CQL filtering specification is actually more powerful than the filters being used by DiGIR and BioCASe, and therefore provided new ideas to the suggested protocol. And the WFS specification uses a different paradigm when it defines "features" as the object of queries, which is somehow similar to DiGIR retrieving records, but different from BioCASe retrieving documents. Although OGC specifications have been a source for new ideas, they also lack some of the existing functionalities of the protocols being integrated (such as inventory operations). Regardless these missing functionalities, suggesting the adoption of only one OGC specification would not be coherent since most part of the specifications have some connection - it would probably be necessary to adopt all of them, including the extensive GML specification. Considering that OGC specifications are being changed to adhere to the SOAP protocol, any serious proposal to adopt them could first wait for their new versions.

Using SOAP as a generic protocol layer has been considered as well. It would not add any functionality to the current protocols, but implementations could benefit from the existing SOAP wrappers, code generators, and also from easier interface definitions (WSDL includes a SOAP binding). Similarities between the SOAP message structure and the general message

format used by DiGIR and BioCASe (kept by the suggested protocol) would certainly facilitate a migration to SOAP. However, a consistent proposal to adopt SOAP as a top level protocol layer would deserve working prototypes and thorough testing due to the significant number of existing performance and interoperability issues between SOAP implementations.

The protocol that is being proposed kept the main message structure and integrated the operations already present in DiGIR and BioCASe.

7.2 Service types

Typical architectures of networks performing search and retrieval operations across distributed databases involve at least three possible different services:

- Message broker service: Responsible for distributing messages to other services and pooling their responses.
- Provider service: A specialized version of a message broker, since messages are only distributed to local resources. It is usually a software installation with many configurable underlying data sources.
- Datasource service: Situated in the end of any request process, it is the service responsible for translating a request into a local query language and retrieving data from a specific data repository. It is usually associated to a single database, or to a subset of it.

All services have many similarities that apparently suggest a common protocol. Even so, the services are clearly not the same, and using the same protocol schema to validate messages from all services would probably make the schema more complicated and less restrictive (allowing for some unreal combinations of elements). A single specification targeted to all services would likely lack the desirable clarity.

Differences become more evident if we think that a message broker service could potentially be configured to access other message brokers, encapsulating whole networks in a cascading way. In this scenario, there could be many ways of reaching a datasource. Attention should be given to avoid endless loops, and special additional requests might be desirable to retrieve tree representations of networks in order to optimize queries.

The integration proposal presented here concentrates only in the protocol used to communicate directly with datasources, which has been considered the most critical part in the integration process.

7.3 Access points

One of the main differences between DiGIR and BioCASe networks is that datasource services don't exist in DiGIR networks. DiGIR resources are only addressed by the protocol through a specific attribute inside the header. On the other hand, BioCASe does not have provider services - it addresses datasources directly through their access points.

An access point is a URL used to communicate with a service. To optimize service discovery through repositories like UDDI it is desirable for each datasource to have its own access point, especially when providers have several underlying datasources related to different themes. Discovery mechanisms, message broker configurations, and even client software benefit from the possibility of directly communicating and referencing specific datasources. In the proposed protocol, each datasource must have its own access point.

7.4 Conceptual binding

One of the main differences between DiGIR and BioCASe is related to how concepts are referenced. DiGIR chose to define concepts as extensions of abstract elements defined by the protocol, and this is done via substitution groups. Although this approach offered more validation possibilities to control how concepts are used, it also limited conceptual schemas to be a flat list of global elements bound to protocol definitions.

This was the main reason for BioCASE to develop its own protocol to be able to reference any element from external and hierarchical schemas. BioCASe references concepts through a simple absolute XPath expression using only the child axis in its abbreviated form (/).

The main requirements for the new protocol concerning conceptual binding were:

- To be able to reference concepts from schemas completely external and independent from the protocol.
- To specify conceptual schemas when referencing concepts, therefore allowing datasources to use multiple conceptual schemas.
- To allow conceptual schemas to be defined in different formats.

The suggested protocol thus extends BioCASe's conceptual binding by allowing any kind of reference to external concept definitions, and by enabling clear distinction between concepts from different schemas. The proposed conceptual binding has the following properties:

- Concepts are referenced through an XML attribute called "path", and are completely external to the protocol.
- Paths can be seen as identifiers for concepts.
- Concept paths must be prefixed by the namespace prefix of the respective conceptual schema (except when they are listed in capabilities responses).

After the conceptual schema prefix, concept identifiers can use almost any string. When conceptual schemas are defined as pure XML Schemas, the suggested format to identify concepts is to use a simple XPath expression pointing the corresponding elements in instance documents.

The following XML Schema could be seen as a simple conceptual schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="dataset">
    <complexType>
      <sequence>
        <element name="record"
                 minOccurs="0"
                 maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="scientificName" type="string"/>
              <element name="basisOfRecord" type="string"/>
            </sequence>
            <attribute name="catalogNumber"
                       type="string"
                       use="required"/>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

In this case, concepts could be referenced within the protocol as:

```
<concept path="cs:dataset/record/scientificName"/>
```

or

```
<concept path="cs:dataset/record/@catalogNumber"/>
```

Where "cs" is a namespace prefix associated to the conceptual schema and previously declared in the document like:

```
xmlns:cs="http://example.net/schemas/cs/1.0"
```

7.5 Filter encoding

The differences between DIGIR and BioCASe regarding filter encoding include:

- While DiGIR uses combined logical operators to express negation (<andNot>, <orNot>) BioCASe uses a unary <not> operator.
- While DiGIR uses "xsi:nil" attributes to express NULL values, BioCASe has specific comparison operators (<isNull>, <isNotNull>).
- While DiGIR defines values as content of concept elements, BioCASe defines values as direct content of comparison operator elements.

Analysing these differences, the suggested protocol considered a better approach to have a unary logical operator to express negation. In practical terms, it allows for a filter to begin with a negation condition without any preceding binary logical operator (which is not possible in DiGIR). As a consequence, all combined operators were dropped. The <isNull> operator was considered more explicit than "xsi:nil", and better suited the chosen conceptual binding technique. Finally, instead of defining values as content of operator elements or concept elements, it was considered a better solution to have a new element <literal> defining values as one attribute.

The suggested protocol has more changes and also incorporates new features to provide additional functionality, including:

- Possibility of using more than two conditions inside binary logical elements (in this case, the position of binary logical elements represent only the boundaries for a sequence of conditions linked through the respective logical operator).
- Possibility of comparing two concepts in the same condition (an example could be a typical data cleaning request searching for records where minimum altitude is greater than maximum altitude).

- Possibility of using parameters (instructing the service to look for values in the environment variables).
- Possibility of using arithmetic operators.

Other enhancements considered important, such as spatial operators and functions, were left to future versions since they would demand additional study.

A summary with examples about the new filter encoding follows.

7.5.1 Expressions

Expressions, new to both DiGIR and BioCASe protocols, are the basic atoms of a filter. They are solely used to build comparison statements, and a valid expression may consist of a literal, a concept, a parameter or an arithmetic operation.

A literal represents a fixed value and takes the form of:

```
<literal value="42"/>
```

A concept uses the proposed conceptual binding to reference a concept defined in a conceptual schema (which in turn is mapped to local data repositories):

```
<concept path="cs:dataset/record/scientificName"/>
```

Parameters are similar to literals but instead of having a fixed value they take it from external sources such as the HTTP POST and GET environment variables. The name of the parameter within the POST/GET variables is given by the parameter definition.

```
<parameter name="sname"/>
```

The previous parameter expression would evaluate to "Ficus" for the following sample URL:

```
http://example.net/a.cgi?sname=Ficus
```

The 4 basic arithmetic operations <add>, <sub>, <mul>, are all binary and take 2 further expressions as their arguments. The first argument is treated as the leftmost one. The arithmetic expression 13+7 would be encoded as:

```
<add>
  <literal value="13"/>
  <literal value="7"/>
</add>
```

7.5.2 Comparison operators

The proposed operators can be classified according to the number of arguments they take:

- Unary operators: <isNull>
- Binary operators: <equals>, <greaterThan>, <lessThan>, <greaterThanOrEquals>, <lessThanOrEquals>, <like>
- Unbounded operators: <in>

All operators take a concept as their first argument. Unary operators take no additional arguments. Binary operators can take an expression as the second argument. Unbounded operators take one or more literal expressions as additional arguments. A simple comparison looks like:

```
<equals>
  <concept path="cs:dataset/record/scientificName"/>
  <literal value="Stipa"/>
</equals>
```

7.5.3 Logical operators

The integration proposal reduced the logical operators to the 3 basic ones: <and>, <or>, <not>. The unary <not> operator takes a single argument which can be a comparison or a logical operator. The other two operators <and> and <or> were changed from their binary form to unbounded and can therefore take any number of arguments, but at least two. A more elaborated expression could be:

```
<and>
  <like>
    <concept path="cs:dataset/record/scientificName"/>
    <literal value="Stipa%"/>
  </like>
  <equals>
    <concept path="cs:dataset/record/basisOfRecord"/>
    <literal value="observation"/>
  </equals>
  <not>
    <in>
      <concept path="cs:dataset/record/@catalogNumber"/>
      <values>
        <literal value="101"/>
        <literal value="102"/>
      </values>
    </in>
  </not>
</and>
```

7.6 Response structure and view

In spite of being able to return structured responses in search operations, both DiGIR and BioCASe have substantial differences in how results are specified and generated.

DiGIR depends on a "record structure" parameter in order to generate search responses. A DiGIR record structure uses a schema (using a subset of the XML Schema language) to represent what concepts should be returned and how they should be structured in responses. At the same time, each DiGIR resource needs to define during configuration a "root table" which will serve as a basis for looping and generating "record" elements in responses. For this reason, record structures must contain the definition of a <record> element which is made of a complex type containing a sequence of either elements referencing concepts or further complex types. Since concepts are referenced through the XML Schema "ref" attribute, values can only be returned as content from XML elements with the same name as the corresponding concept.

On the other hand BioCASe generates responses based on the structure of conceptual schemas used by datasources. Each conceptual schema is converted into another XML file as a basis for mapping and generating responses. Consequently, responses always follow the same structure of conceptual schemas, but they are not bound to records from a single underlying

"root table".

The suggested protocol tried to reconcile both approaches by extending the idea of "record structures" using a more generic alternative of "response structures". A response structure is also a schema in XML Schema language, specifying how concepts should be returned, but without any fixed record binding. Response structures are actually part of a view definition which also comprises a mapping section and an indexing element definition that should be used as a reference for counting and paging.

A view could look like:

```
<view>
  <structure>
    <schema
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://example.net/schemas/sp1">
      <xs:element name="dataset">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="specimen"
                        minOccurs="0"
                        maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="acnum" type="xs:string"/>
                  <xs:element name="sname" type="xs:string"/>
                </xs:sequence>
                <xs:attribute name="lastupdate"
                              type="xs:dateTime"
                              use="optional"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </schema>
  </structure>
  <indexingElement path="/dataset/specimen"/>
  <mapping xmlns:s="http://example.net/cs/1.0">
    <nodes>
      <node path="/dataset/specimen/acnum"/>
      <concept path="s:CatalogNumber"/>
    </nodes>
```

```
    <nodes>
      <node path="/dataset/specimen/sname"/>
      <concept path="s:ScientificName"/>
    </nodes>
    <nodes>
      <node path="/dataset/specimen/@lastupdate"/>
      <concept path="s:DateLastUpdated"/>
    </nodes>
  </mapping>
</view>
```

The first <element> definition inside a <schema> should always be considered the root element when producing responses, so a possible result according to the given structure could be (excluding external protocol elements):

```
<dataset xmlns="http://example.net/schemas/sp1">
  <specimen lastupdate="2002-05-15T13:20:00Z">
    <acnum>423</acnum>
    <sname>Thylacinus cynocephalus</sname>
  </specimen>
  <specimen lastupdate="2001-02-19T16:04:01Z">
    <acnum>567</acnum>
    <sname>Sarcophilus Harrisii</sname>
  </specimen>
</dataset>
```

The mapping section uses a simple one-to-one mapping that could be extended in future versions of the protocol. It maps nodes (content elements or attributes) from the response structure to concepts from conceptual schemas used by the datasource.

Since the XML Schema language is vast and complex, wrappers are not expected to support the whole specification. Only a restricted subset of the language is required, basically including target namespaces, elements (with cardinality), attributes, sequences, and local declarations of complex and simple types (with simple and complex content). Additional aspects of the XML Schema language can be supported by wrappers and advertised in capabilities responses.

The containment relationship between complex elements and its sub elements or attributes must be translated to the datasource backend specific query language in order to generate responses. In the case of relational databases, one or more SQL statements could be used taking into account

datasource's configuration settings regarding tables, joins, and mappings with the conceptual schemas.

Resulting structures can arrange concepts in many ways, as elements' content or attributes' values, all accepting custom names, and the result can be validated against an external XML Schema.

By separating the notion of response structures from conceptual schemas, not only different views can be taken from multiple conceptual schemas, but also conceptual schemas are left to be defined in different ways, perhaps not using XML Schema.

7.7 General message format

The general message format has been kept. A root element, <request> or <response>, should contain a <header> section followed by another section with the operation name. The only difference is that both previous protocols defined the operation using a <type> element inside the header, and then used a generic <content> element following the header. Therefore, it allowed for inconsistencies between the specified type and the actual content elements. By replacing the general <content> element by the operation name, validity becomes stricter and the schema more readable.

In requests the operation element contains possible parameters to be passed:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <search>
    <!-- search operation specific parameters -->
  </search>
</request>
```

In responses, the operation element contains possible results followed by a <diagnostics> section containing any relevant additional information, as well as warnings or errors:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
```

```
  </header>
  <search>
    <!-- search operation result -->
  </search>
  <diagnostics>
    <!-- diagnostics information -->
  </diagnostics>
</response>
```

7.7.1 Header

Headers were also kept essentially the same, but some additional changes are being suggested. Besides removing the <type> element the main changes proposed are:

- The attribute "resource" used by DiGIR networks has been removed from the destination element, since datasources are now referenced by their access points.
- <source> elements are allowed to have multiple occurrences to track the addresses used by all services (stages) involved in a communication process. Intermediary services must include their address in the end of the list. By keeping track of the process, implementations of cascading message brokers will have a means to avoid endless loops.
- The access point is now an attribute of <source> instead of its content, and a timestamp "sendtime" has also been incorporated as an attribute.
- Inside <source> there's now an optional <software> element to indicate name and version of the software used to process the message.
- A destination element is not necessary anymore, since the protocol covers only datasource services which are directly reached by their access points.

A header from a request that has been dispatched by a message broker is shown in the next example.

```
<header>
  <source accesspoint="11.12.13.14"
          sendtime="2001-12-17T09:30:47-05:00">
    <software name="generic client software"/>
  </source>
  <source accesspoint="15.16.17.18"
          sendtime="2001-12-17T09:30:49-05:00">
    <software name="generic portal" version="0.95"/>
  </source>
</header>
```

A header from a response to the previous request could be:

```
<header>
  <source
    accesspoint="http://example.net/datasource/a.cgi"
    sendtime="2001-12-17T09:30:50-05:00">
    <software name="generic wrapper" version="1.1.4"/>
  </source>
  <destination accesspoint="11.12.13.14"/>
</header>
```

7.7.2 Diagnostics

The <diagnostics> section present in responses is almost the same as it was before, containing one or more <diagnostic> elements, as in the next example:

```
<diagnostics>
  <diagnostic code="PRG_MISSING_LIBRARY" type="error">
   Could not find module MBSTRING
  </diagnostic>
</diagnostics>
```

The former "severity" was renamed to "type" and contains as before the following categories: debug, info, warn, error, fatal

However, from a client software perspective, unification of diagnostic codes was also considered important. The complete list of codes should be present in a final specification of this protocol.

7.8 Request operations and response types

The proposed operations for the protocol include:

- **Metadata:** Containing basic information to describe the service.
- **Inventory:** Used to retrieve distinct values from a list of concepts.
- **Search:** Main operation used to retrieve data from local data sources in a customizable response structure.
- **View:** Used to get pre-defined views from local data.
- **Capabilities:** Containing essential settings and technical information about the service functionality.
- **Ping:** Used for monitoring purposes to check availability of services.

7.8.1 Metadata operation

The main idea of a metadata operation is to retrieve some basic description about a service, including things like label, abstract, keywords, related entities and contacts, etc.

The existing DiGIR metadata operation served as the basis for this proposal but it has been changed in several ways in order to address a number of known issues. Main changes are:

- Inclusion of the datasource access point (could be seen as redundant considering only direct communications with datasources, but it is actually essential considering that the response may go to a client software that doesn't know the service's address).
- Inclusion of "lang" attribute in content elements that could be served in many languages, and change of their cardinality to "unbounded".
- Removal of <implementation> element which was duplicated (header element already has this information in the <software> element).
- Move of <minQueryTermLength> to the capabilities response.
- Removal of elements <maxSearchResponseRecords> and <maxInventoryResponseRecords> which were substituted to <maxElementLevels> and <maxElementRepetitions> in the capabilities response. <maxElementLevels> can be used to restrict the

maximum number of nested elements in response structures acting on the XML depth dimension. <maxElementRepetitions> can be used to restrict the maximum number of repetitions of each element being produced by search responses, acting on the XML breadth dimension.

- Removal of elements <recordBasis> and <recordIdentifier> which only make sense to specific datasources.
- Change of <useRestrictions> to a generic <rights> element.
- Renaming of previous resource <name> to <label>.
- Inclusion of available local views in a list.
- Change of <dateLastUpdated> and <numberOfRecords> elements to optional attributes from views (since a datasource can now actually serve representations of different things).

Another issue worth mentioning is that, even if the datasource could also be reached through a provider service, there's no explicit relationship shown in the metadata response. A datasource is seen as a completely independent service.

The term "provider" has caused a lot of confusion in the past due to all possible meanings: provider service, provider software, original data provider, and provider institution. In the proposed protocol there is no explicit relationship with a provider service in datasources metadata responses, but it is possible to give clear credits to a provider institution. Actually a datasource service can now give credits to as many entities (organisations, institutions, companies, etc) it wishes.

Entities can also play more than one role at the same time, and roles are defined by the networks. The list of contacts associated to entities is the same as used by DiGIR. Another interesting point is that a same entity can be referenced by more than one service. For those cases, there is now an <identifier> element that can be used as a global unique identifier to produce lists of entities without repetitions across many services.

A metadata request requires no parameters:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <metadata/>
</request>
```

A metadata response looks like:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <metadata>
    <label lang="en">Plant specimen database</label>
    <label lang="pt_BR">
     Banco de dados de Espécimes de Plantas
    </label>
    <accesspoint>
     http://example.net/datasource/a.cgi
    </accesspoint>
    <abstract lang="en">
      This resource contains specimen records
      of plants from all over the world
    </abstract>
    <keywords lang="en">plant, specimen</keywords>
    <citation lang="en">
     Plant specimen database. Retrieved on (date
     accessed). http://example.net/datasource/a.cgi
    </citation>
    <rights lang="en">Users may not distribute, modify,
     transmit, reuse, repost, transfer, or use any
     content or information from this service for
     commercial purposes without prior written permission.
    </rights>
    <conceptualSchemas>
      <conceptualSchema namespace=
        "http://example.net/schemas/specimen/1.0"/>
    </conceptualSchemas>
    <views>
      <view name="specimen"
            dateLastUpdated="2004-08-01T20:00:00-03"
            numberOfRecords="100"/>
    </views>
    <relatedEntities>
      <entity lang="en">
        <identifier>
         http://example.net/organisation/mydata.xml
        </identifier>
```

```
        <name>Biodiversity Informatics Institute</name>
        <name lang="pt_BR">
         Instituto de Informática para Biodiversidade
        </name>
        <acronym>BII</acronym>
        <logoURL>http://example.net/mylogo.gif</logoURL>
        <role>provider</role>
        <role>host</role>
        <description>
         Organisation created for this example
        </description>
        <relatedInformation>
         http://example.net/organisation/
        </relatedInformation>
        <contact type="administrative">
          <name>Person A</name>
          <title>Curator</title>
          <email>person_a@example.net</email>
          <phone>+111 11 111111</phone>
        </contact>
        <contact type="technical">
          <name>Person B</name>
          <title>Sysadmin</title>
          <email>person_b@example.net</email>
          <phone>+111 11 222222</phone>
        </contact>
      </entity>
    </relatedEntities>
  </metadata>
</response>
```

As shown in the last example, the "lang" attribute can also be declared in elements <metadata>, <entity> or <contact> as a default setting to all language aware elements which are below them.

7.8.2 Inventory operation

Inventory operations, which are used to retrieve distinct values from concepts, are present in both DiGIR and BioCASe with a few minor differences. Apart from the different naming ("inventory" in DiGIR, and "scan" in BioCASe), DiGIR accepts filters in requests and has two different counting procedures: the total number of distinct values and the number of occurrences of each distinct value. All these features are present in the new protocol.

However, both DiGIR and BioCASe are unable to request distinct values from a combination of concepts, which should now be possible.

The main features of inventory operations in the suggested protocol are:

- An optional filter can be used in requests.
- One or more concepts can be used as parameters (when more than one concept is specified, distinct combinations of their values are returned).
- Concepts can be part of different conceptual schemas.
- An optional "count" parameter can be used to request the total number of distinct records and the number of occurrences of each record in the local datasource repository.
- Paging can be done using the parameters "start" and "limit" ("start" represents the index of the first record to be returned and begins with "0", while "limit" represents the number of records to be returned).
- Responses include a <summary> element where "start" indicates the index of the first record returned, "next" indicates the index of the next record that could be retrieved using a subsequent request, "totalReturned" indicates the number of records returned, "totalMatched" indicates the total number of records that matched the query (not necessarily returned)

An inventory request looks like:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <inventory count="true"
             start="0"
             limit="2"
             xmlns:ns1="http://example.net/ns1"
             xmlns:ns2="http://example.net/ns2">
    <concepts>
      <concept path="ns1:record/taxon/fullname"/>
      <concept path="ns2:gazeteer/location/isocountry"/>
    </concepts>
    <filter>
      <!-- filter specific elements -->
```

```
    </filter>
  </inventory>
</request>
```

An inventory response for this request could be:

```
<?xml version="1.0" encoding="utf-8" ?>
<response>
  <header>
    <!-- header specific elements -->
  </header>
  <inventory>
    <record count="13">
      <value>Abies alba Mill.</value>
      <value>DE</value>
    </record>
    <record count="45">
      <value>Quercus robur L.</value>
      <value>DE</value>
    </record>
    <summary start="0"
             next="2"
             totalReturned="2"
             totalMatched="117"/>
  </inventory>
  <diagnostics>
    <!-- diagnostics information -->
  </diagnostics>
</response>
```

The order of values inside <record> must correspond to the order of concepts specified in the request.

The "totalMatched" and "count" attributes in responses are only present if the request asked for counting.

If a request asks for counting but specifies "limit" zero, than no records are returned, but a "totalMatched" is present in responses.

7.8.3 Search operation

Search operations simply put together two of the main ideas already discussed: views and filters. Additional parameters can be used to request par-

tial views and to perform paging and counting.

The main features of search operations are:

- Search requests can specify how responses should structure results by means of view definitions.
- View definitions can be completely declared in requests, or they can be remotely located and referenced, or a default local view can be used if none is specified.
- Partial views can be requested by specifying nodes from the view.
- Filters are now optional.
- Paging and counting is based on the <indexingElement> from the view section.

Supposing that the view used as an example in section "Response structure and view" could be available from "`http://example.net/viewlib/specimen.vwd`", then a search request asking only for the element "/dataset/specimen/sname" could look like:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <search count="true" start="0" limit="2">
    <view
     location="http://example.net/viewlib/specimen.vwd"/>
    <partial>
      <node path="/dataset/specimen/sname"/>
    </partial>
    <filter xmlns:s="http://example.net/cs/1.0">
      <equals>
        <concept path="s:CatalogNumber"/>
        <literal value="423"/>
      </equals>
    </filter>
  </search>
  <diagnostics>
    <!-- diagnostics information -->
  </diagnostics>
</request>
```

Concerning requests:

- If no <view> section is specified, than the default local view should be used (and if there are no local views, than an error should be raised).
- If no <partial> section with specific nodes is specified, than all view structure should be returned.
- If no <filter> is present then no filtering is used.

A search response for this request could be:

```
<?xml version="1.0" encoding="utf-8" ?>
<response>
  <header>
    <!-- header specific elements -->
  </header>
  <search>
    <dataset xmlns="http://example.net/schemas/sp1">
      <specimen>
        <acnum>423</acnum>
        <sname>Thylacinus cynocephalus</sname>
      </specimen>
    </dataset>
    <summary start="0"
             totalReturned="1"
             totalMatched="1"/>
  </search>
  <diagnostics>
    <!-- diagnostics information -->
  </diagnostics>
</response>
```

Notice that element <acnum> was not requested, but since it is a mandatory element according to the view definition then it should be always returned. But the optional attribute "lastupdate" was not returned.

Even being able to specify in search requests how many elements (using the <indexingElement> tag) should be returned, responses may be limited by a service setting called "maxElementRepetitions". This setting acts on the breadth dimension the XML result (considering only the XML content inside the <search> element). It means that no element originated from the same "xsd:element" definition can be repeated more times than the

maximum number of allowed repetitions. This setting gives data providers some control to avoid server overload and to avoid complete data dumping requests, and at the same time it can be seen as a paging limit.

7.8.4 View operation

This operation (not present in DiGIR and BioCASe) came up as a natural consequence after having defined all features of the suggested protocol. It actually does not offer more functionality than what could already be achieved through regular search operations. But it offers a simpler way of performing searches, leveraging direct access to original data providers and opening new possibilities.

Since search responses are essentially the result of a combination of a view definition and a filter, direct access to local data can be considerably facilitated through local definitions of parameterized views. The idea is that datasources may be able to define one or more XML views from its local data.

These local views can be designed at will by data providers considering the conceptual schemas being used, or they can simply reference remote pre-defined views. Each local view has a local unique name that can be referenced by search and view operations. One of the local views should be set as being the default.

As already seen in the "Filter encoding" section, comparison expressions inside filters can use a literal value or a parameterized value to be taken from environment variables (HTTP GET or HTTP POST parameters). This entire infrastructure is enough to provide simple direct access to local data.

A local view could be defined and listed in the capabilities response as:

```
<views default="specimen">
  <view name="specimen"
        xmlns:s="http://example.net/cs/1.0">
    <structure>
      <schema
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://example.net/schemas/sp1">
        <xs:element name="dataset">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="specimen"
                          minOccurs="0"
```

```
                    maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="acnum"
                                type="xs:string"/>
                    <xs:element name="sname"
                                type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </schema>
    </structure>
    <indexingElement path="/dataset/specimen"/>
    <mapping>
      <nodes>
        <node path="/dataset/specimen/acnum"/>
        <concept path="s:CatalogNumber"/>
      </nodes>
      <nodes>
        <node path="/dataset/specimen/sname"/>
        <concept path="s:ScientificName"/>
      </nodes>
    </mapping>
    <filter>
      <equals>
        <concept path="s:CatalogNumber"/>
        <parameter name="id"/>
      </equals>
    </filter>
  </view>
</views>
```

The view operation is the only one which is only available through HTTP GET or HTTP POST invocation without using an XML message as input. A simple call to the previously defined view would be:

```
http://example.net/a.cgi?operation=view&name=specimen&id=123
```

The response would be an XML representation of part of the local data. In the given example, the parameterized filter takes the value of the "id"

parameter and uses it in a comparison with the local mapping of the concept "CatalogNumber". If "CatalogNumber" maps to a field with unique values, then the response would be an XML representation of a specimen record, which could be even bookmarked or referenced:

```
<?xml version="1.0" encoding="utf-8" ?>
<dataset>
  <specimen>
    <acnum>123</acnum>
    <sname>Rubus rosaefolius</sname>
  </specimen>
</dataset>
```

Responses from view operations are the only ones that do not include by default specific protocol elements (like header and diagnostics). They contain just the XML representation given by the view structure definition. However, it is possible to request the presence of protocol elements by using the parameter called "verbose" (in this case, the response will follow the same structure of search responses).

Views can also perform paging when there are multiple "objects" (indexing elements) involved. Paging can be done using the "start" and "limit" parameters.

Since views are only defined against concepts from conceptual schemas, they can be used by any datasource which uses the same conceptual schemas and which have mapped the necessary concepts. Libraries of views are also possible, and could be used during service configuration.

One interesting aspect of the view given as an example, is that the respective calling string could serve as a global unique identifier for an underlying object. It is therefore a special kind of identifier which also happens to be an address, from where the latest version of the object can always be directly retrieved.

7.8.5 Capabilities operation

This operation was originally conceived in BioCASe to indicate the conceptual schemas used by datasources, as well as all mapped concepts from each schema. Although it can also be seen as metadata about services, it has been decided that two operations were more convenient, one with basic descriptions about the service (metadata) and another with technical information (capabilities). Some of the DiGIR metadata elements were moved to the

capabilities operation described here. So the idea of a capabilities operation is to retrieve settings, available functionalities, and technical information about services.

A capabilities request requires no parameters:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <capabilities/>
</request>
```

A capabilities response includes several different sections:

```
<?xml version="1.0" encoding="utf-8" ?>
<response>
  <header>
    <!-- header specific elements -->
  </header>
  <capabilities>
    <schemas>
      <!-- conceptual schemas being used -->
    </schemas>
    <views>
      <!-- possible XML views from local data -->
    </views>
    <settings>
      <!-- general service settings -->
    </settings>
    <operators>
      <!-- supported operators in filters -->
    </operators>
    <structure>
      <!-- supported set of response structure language -->
    </structure>
  </capabilities>
</response>
```

Conceptual schemas and mapped concepts

This section should contain a list of conceptual schemas being used by the service, with two mandatory attributes: namespace and location. Each con-

ceptual schema has a list of mapped concepts, each one with two optional attributes:

- Searchable: indicating if the concept can be used inside filter expressions (defaults to true).
- Mandatory: indicating if the concept should be present in all search responses (defaults to false).

This section could look like:

```
<schemas>
  <conceptualSchema
    namespace="http://example.net/schemas/specimen/1.0"
    location="http://example.net/schemas/specimen/1.0/sp.xsd">
    <concept path="scientificName"/>
    <concept path="catalogNumber"/>
    <concept path="basis" searchable="false"/>
    <concept path="ipr" mandatory="true"/>
  </conceptualSchema>
</schemas>
```

Views

This section simply lists all local views that are available to be used by search and view requests. Each view has a unique name, and one of them should be selected as a default. Views are an optional feature for data providers.

```
<views default="specimen">
  <view name="specimen">
    <!-- view definition -->
  </view>
  <view name="collector">
    <!-- view definition -->
  </view>
</views>
```

General settings

There are at least five possible settings of interest to clients. All of them giving control to service providers to avoid server overload caused by requests for excessive amount of data:

- minQueryTermLength: The minimum length of a wildcarded string used in "like" expressions.
- maxElementRepetitions: The maximum number of repetitions allowed for any repeatable elements in responses (related only to the content section). It can also be used as a reference for paging.
- maxElementLevels: The maximum number of levels allowed for responses (related only to the content section).
- maxResponseTags: The maximum number of tags that can be returned in responses (related only to the content section).
- maxResponseSize: The maximum size in kilobytes allowed to be returned in responses.

This section could look like:

```
<settings>
  <minQueryTermLength>3</minQueryTermLength>
  <maxElementRepetitions>200</maxElementRepetitions>
  <maxElementLevels>30</maxElementLevels>
</settings>
```

Supported operators

This section is used to indicate which operators can be used by filter encodings. The operators are divided into three categories:

- logical: used to indicate capability of dealing with "and", "or" and "not" operators.
- comparative: can be used to indicate support to "in", "isNull", "like" and "basicComparativeOperators" (=,<,<=,>,>=).
- basic arithmetic operators: used to indicate support to addition, subtraction, multiplication and division.

This section could look like:

```
<operators>
  <logical/>
  <comparative>
    <basicComparativeOperators/>
```

```
    <in/>
    <isNull/>
    <like/>
  </comparative>
  <basicArithmeticOperators/>
</operators>
```

Supported response structure

This section is used to indicate which subset of the XML Schema language is supported by the service in order to interpret response structures. A minimal subset is required for all wrapper implementations, and it is represented by the tag <basicSchemaLanguage>. Additional features of the XML Schema language may be optionally supported by wrappers.

This section could look like:

```
<structure>
  <basicSchemaLanguage/>
  <choice/>
</structure>
```

Other features include <group>, <import>, <references>, <extension>, <restriction>, and <substitutiongroup>.

7.8.6 Ping operation

This new operation has been introduced to enable better monitoring of services, without needing to use a metadata request (which usually needs to connect to some local database). Requests and responses are quite simple and can be used to check reachability, response time, and also to check if wrapper software is properly installed.

Sample ping request:

```
<?xml version="1.0" encoding="utf-8" ?>
<request xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <ping/>
</request>
```

Sample ping response:

```
<?xml version="1.0" encoding="utf-8" ?>
<response xmlns="http://example.net/protocol/1.0">
  <header>
    <!-- header specific elements -->
  </header>
  <pong/>
  <diagnostics>
    <!-- diagnostics information -->
  </diagnostics>
</response>
```

7.9 Operation calls

Concerning operations that can use an XML document as input, it has been agreed that communication with the service will be done through a single HTTP POST or HTTP GET parameter called "request" which must contain either the XML message according to the specified protocol or an URL pointing to the XML message. These operations include: "metadata", "capabilities", "inventory", "search", and "ping".

Operations that do not require complex parameters, like "metadata", "capabilities" and "ping" can also be invoked by passing an HTTP POST or an HTTP GET parameter called "operation" whose value should be the name of the operation.

View operations do not use an XML message in requests, so they are only invoked by HTTP POST or HTTP GET parameters including: "operation" (always equals to "view"), "name" (name of the view), "verbose" (optionally requests additional protocol elements), "start" (index of first object to be retrieved), and "limit" (total number of objects to be retrieved).

The default operation when a service is invoked without any parameters is a "metadata" operation.

7.10 Known and possible limitations

Since all efforts were concentrated in the integration process, the use of this protocol by other data exchange schemas that are being proposed as standards to the biodiversity community was not properly studied. Schemas like SDD and Taxonomic Concept Transfer Schema (TCS)[67] require features

[67] http://www.soc.napier.ac.uk/tdwg/index.php

that may or may not be addressed by this protocol when producing search responses, such as:

- Returning recursive elements.
- Returning inter-related elements (when an element references another in a different part of the structure)

Because the suggested protocol accepts flexible response structures, it might be possible to use some combination of mapping technique and response structure definition that could produce the desired results to the aforementioned schemas. However, additional research is required to better investigate these issues.

Since great part of biodiversity data has some geographic property, it would be highly desirable to support spatial operators in the protocol, as provided by the OGC CQL specification. However, the suggested protocol does not include any spatial operators in its current version.

The proposed protocol also does not enable the definition and use of custom functions inside filters. This could be an interesting feature to improve even more filtering capabilities.

When developing software to be compliant with the proposed protocol, the greatest challenge will probably be to dynamically generate responses to a requested custom structure with its own indexing element definition. When analyzing BioCASe experiences with dynamic response structures this seems to be feasible. Although the BioCASe protocol did not allow custom response structures, the internal way of storing them is very similar and has proven to work successfully.

By using a similar implementation approach, the dynamic generation of SQL could be based on an acyclic connected graph representation of the database structure with tables' aliases as vertices and foreign/primary key relationships as edges. For a given response structure, a rooted tree representation of the graph would need to be extracted to be able to uniquely identify a path to the requested tables that could be used to generate dynamic SQL joins.

Hierarchical and valid XML could be created with only one problem being identified regarding repeatable elements. If the relational database structure is different from the "relational" XML structure, the algorithm used for XML generation in some cases may have problems identifying the correct XML node that represents the "many" side of a relationship. Any repeatable node needs to be created in the XML result for each table record joined to the parental record related to preceding XML nodes. Problems could arise when

there are cascading repeatable elements without any attributes or any other mapped sub elements in the top of the chain. In this case, an algorithm may not know in which node the repetition should happen. As the BioCASe system has fixed and locally configured structures used for responses, it was possible to use local provider configurations to solve this. However, some ideas were proposed to overcome this problem in a custom response structure scenario, including the possibility to directly map repeatable elements and therefore providing additional references to the algorithms. These ideas can possibly eliminate the problem in most but maybe not all situations. So there could be some specific types of structures that will not be supported by datasource services depending on the algorithm used and on the mapping provided.

In case of denormalized databases an automatic normalization of the generated XML is not desired, so redundant data in the responded XML could therefore not be avoided. However, if denormalization is the result of using the same value of a concept for the whole datasource (such as global metadata concepts like collection name, institution name) it is possible to overcome the problem by allowing fixed values when mapping conceptual schemas. In this case, wrappers could easily avoid redundant information being returned.

7.11 Impact on software

All changes and additions proposed will certainly impact every software that needs to deal with DiGIR or BioCASe protocols. All types of services will be affected.

The least affected software are probably message brokers, which will need to stamp their IP address in headers and maybe handle the different error codes. In particular, DiGIR message brokers will need to address datasources by access points (affecting configuration and message routing).

Client software from both sides will need to adapt themselves to changes in filter and response structure specification. However, response parsers will not have significant changes, except if they want to make use of new functionalities like inventories accepting multiple concepts, or the new view operation. But they will need to consider some changes made in element names.

For the moment, DiGIR provider services will not be officially covered by the protocol. Their configuration and installation will need to be changed to also serve as a set of datasource services.

DiGIR providers (when also seen as datasource services) and BioCASe wrappers will be the most affected software. Accepting and parsing the new

filter and response structure specifications, producing results according to any valid given structure, and taking into account many other changes, although feasible in a short term, will demand considerable work. However, new features like the view operation, and inventory operations with multiple concepts will not produce much impact.

Configurators, which are usually integrated with wrappers, will need to adapt themselves to the proposed service metadata and to additional configuration parameters available from the new protocol.

Considering all these consequences, networks must carefully decide on a strategy to upgrade all their software components.

7.12 Migration strategy

When choosing software to be changed, priority should be clearly given to DiGIR providers and BioCASe wrappers, which should all become datasources compliant with the new protocol. Recommendation when making changes to these components is to keep as much as possible compatibility with previous protocols, so that they can be gradually substituted without experiencing significant service downtime.

Backwards compatibility can be achieved by including an additional verification on top of these components to detect the protocol being used by requests, and then addressing each message from different protocols by different libraries or procedures. This should ideally be done keeping the same access point, in case of installed BioCASe wrappers, so that other client software or message brokers from different networks accessing the same service could be upgraded at different times. Another possibility for BioCASe wrappers could be to install and configure new datasources as separate and parallel services, but at some point it would require updates in UDDI registries for networks that use them.

DiGIR providers when acting on behalf of several datasources will necessarily have new access points for each current resource. Thematic DiGIR networks could gradually install and configure separate message brokers and client software, and then redirect their official addresses when all datasources have been upgraded.

Tools to automate updates on configuration files will definitely be needed when installing new versions of datasources. If feasible, the best solution would be to allow the configuration parsing software (configurators or wrapper) to be able to read old and new versions of the configuration files and write out new versions when asked to. This reduces the number of independent software components needed.

Another alternative is to develop proxy services that could serve as message translators between different protocols. However it may not be an easy task, since new features from the proposed protocol cannot be translated to the existing ones. And any features from existing protocols which are not present in the new one could make it difficult to perform translations in the opposite way, although this should not be the case here (only a few deprecated metadata elements have been removed).

7.13 Considerations about conceptual schemas

The suggested protocol clearly separates conceptual schemas from possible views of mapped data. This fact provides great flexibility when conceiving conceptual schemas, which could be defined in several ways, not necessarily using the XML Schema language but at least using XML to benefit from the same parsing technology being used. Modularization and extension of conceptual schemas could be achieved by different means since datasources accept multiple conceptual schemas that are not bound to the protocol.

The advantage of modularizing conceptual schemas is that they can be reused by different networks that have only a partial intersection between their conceptual domains. When modularized, changes in one conceptual schema will not affect other parts of the conceptual domain. But from a data exchange perspective it doesn't matter if concepts are in the same schema or not, since they can be freely combined and aggregated by different views.

However, it is still an open issue how to best represent and modularize conceptual schemas - this was not part of this work.

The great flexibility provided by the protocol makes it possible for schemas like DarwinCore to choose between keeping its current approach, as a simple list of concept definitions, or not. Proper concept grouping and cardinalities when exchanging data can be achieved by XML views. And it is also possible for XML Schemas like ABCD to be modularized when seen as a conceptual schema.

It is important to observe that if different conceptual schemas define the same concept using a different format, it may be difficult to achieve full interoperability between data providers even by using the same protocol. A classical example is related to how dates can be represented. Conceptual schemas that separate dates in three concepts (day, month, and year) like DarwinCore does, need to use more protocol features to represent a condition like: date >= '2002-05-19'. The associated filter encoding to represent this condition could be:

```
<or>
  <greaterThan>
    <concept path="c:year"/>
    <literal value="2002"/>
  </greaterThan>
  <and>
    <equals>
      <concept path="c:year"/>
      <literal value="2002"/>
    </equals>
    <or>
      <greaterThan>
        <concept path="c:month"/>
        <literal value="05"/>
      </greaterThan>
      <and>
        <equals>
          <concept path="c:month"/>
          <literal value="05"/>
        </equals>
        <greaterThanOrEquals>
          <concept path="c:day"/>
          <literal value="19"/>
        </greaterThanOrEquals>
      </and>
    </or>
  </and>
</or>
```

While a conceptual schema defining dates using a single concept, like ABCD, would represent the same condition as:

```
<greaterThanOrEquals>
  <concept path="c:ISODateTime"/>
  <literal value="2002-05-19"/>
</greaterThanOrEquals>
```

This example illustrates how differences between conceptual schema definitions may lead to interoperability issues even using the same protocol.

The meaning of concepts should be as clear as possible when dealing with different conceptual schemas in order to avoid more problems. Another example could be a conceptual schema defining a concept called "ScientificName" as being the latest identification accepted for a specimen. Another

conceptual schema could use the same name for a concept but with a different meaning, such as any identification of the specimen, not only the latest one. A response structure assuming a one-to-one relationship between specimen and identification would certainly get wrong results from the later concept definition.

Sometimes cardinality can be embedded in a concept definition, such as a "collector" concept that may contain several collector names concatenated in a single string, whereas other conceptual schemas may have multiple occurrences of a single collector concept. Response structures must be aware of these differences to avoid more problems.

So the next step to achieve full integration and interoperability between DiGIR and BioCASe networks resides on an agreement at the conceptual schema level, by using at least a fully compatible set of concepts; or even better, by adopting a common set of concepts which could be derived from a concept library or an ontology for biodiversity data.

8 Additional recommendations

During the whole integration process, several additional topics that are interrelated to the protocol being suggested have also been discussed. They vary from a list of recommended specifications and desirable tools to specific research areas that could be interesting to investigate. A summary about these discussions is presented in the next paragraphs.

8.1 Specifications

As soon as an agreement is reached about the common protocol, one of the first things necessary is to produce a formal specification describing all aspects of the new protocol and including all necessary implementation details. It would also be highly desirable to produce additional specifications to get full advantage from the web services technology and to cover interoperation between other components that are part of the existing networks.

8.1.1 WSDL service specification

To be fully compliant with the web services definition it is necessary to have an XML description of the service interface and bindings. This is usually done through a WSDL document which formally specifies all possible interactions with the service. In theory, such a description could automate interaction between different services and could help on more sophisticated scientific workflows. A number of tools already exist which can take a WSDL document and automatically generate code in specific programming languages to communicate with the service. A WSDL document describing the suggested protocol could therefore provide additional means to facilitate development of network components as well as other tools.

8.1.2 Provider and message broker services

The protocol being proposed could be used almost in its complete form by other types of services that are typically part of the networks querying distributed data, namely providers and message brokers. These services are not officially covered by this work and they would likely deserve at least one additional specification.

Having a set of specifications covering all services used by typical components from distributed query architectures would certainly facilitate the

development of additional tools. Moreover, by using the same protocol these tools could potentially be shared by all networks.

8.2 Other suggested tools

When upgrading the current code base to adjust it to the new protocol, and also when migrating the existing networks to use new software releases, some additional tools could be of great value.

8.2.1 Service validators

During the process of updating existing datasource software or producing new implementations conforming to a new protocol, it would be highly desirable to have an external tool capable of automatically checking protocol conformance and thus validating or not the implementation. XML validation is only a first step to verify correct functionality of a datasource service. There are many possible constructions for filters and response structures that could produce wrong responses even inside valid XML documents. These situations will demand careful attention and more elaborate tests.

8.2.2 Automatic updates

It could also be worthwhile to check if there is any tool that could be used to enable automatic updates on data provider software, or even if it would be feasible to develop such a tool. In this case, networks could potentially schedule the upgrade of all data providers' software in a short period of time. The same tool could be used not only during drastic network migration periods, but also during regular software updates that happen frequently.

8.2.3 Proxy services

Although translation between messages from the suggested protocol and messages from the existing ones may not be fully possible, proxy services could help on the gradual migration of networks. It would be necessary to carry out an initial study to assess how far and in which cases a translation service could successfully intermediate communication between the different protocols. Depending on the conclusions, development and use of proxy services could play an important role when upgrading the networks.

8.3 Additional researches

Besides being a source for new ideas, some of the technologies considered during the integration process deserve more detailed investigation. Conclusions from such studies could drive future versions of the protocol to different directions.

8.3.1 XQuark prototype

As already mentioned in the XQuery section, a specific tool has been identified as a potential alternative for the protocol. To better assess its possible advantages and limitations compared to what has been proposed here and considering biodiversity networks' needs, it would be necessary to develop a prototype. The following items would require special attention:

- Evaluate the possibility of having interfaces capable of generating XQuery statements.
- Verify limitations and capabilities of the mapping language being used.
- Verify how conceptual schemas are defined.
- Evaluate the mapping interface.
- Estimate the amount of work to extend the number of supported database drivers.
- Conduct performance tests with large databases.
- Check if there is a way for data providers to limit the amount of data returned per request.

8.3.2 SOAP prototype

To better evaluate advantages and limitations of basing the entire networks on the SOAP technology, it would also be necessary to develop a prototype service. As already mentioned, this service could be an additional layer on top of a message broker or on top of a comprehensive data provider. The following items would require special attention:

- Verify interoperability issues, especially concerning complex data types like filters and response structures.
- Conduct performance tests in comparison to existing services.

Regardless the results, frequent studies about such tools and technologies would certainly bring new ideas and contribute to the improvement of existing networks.

8.3.3 Ontologies

Another wide research area concerns how to better represent conceptual schemas. Some of the conceptual schemas being used now are limited to property definitions with their corresponding data types. Richer representations including classes and relationships would certainly offer more possibilities, especially for networks that need to represent more complex situations, such as species interactions.

A number of ontology languages are available with different levels of robustness and expressivity. Many have been created to enable the Semantic Web idea and are based on XML. Richer conceptual representations combined with richer mappings will be sooner or later necessary to biodiversity networks.

9 Final comments

For several reasons, the adoption of a common protocol by DiGIR and BioCASe networks can be considered a priority:

- The number of different implementations that rely on DiGIR and BioCASe protocols has grown considerably in recent periods, making the upgrade of existing code base already difficult. Implementations tend to be bound to one of the existing protocols since the effort to produce software compliant with both is usually unaffordable. However, it is highly desirable for all software implementations to be fully interoperable. By using interoperable server software, data providers will be able to get more visibility. On the other hand, fully interoperable client software will be able to access a larger number of data providers. It is expected that any software implementation will prefer to use a common protocol, and therefore new implementations being planned could save considerable development time if a common protocol is adopted by the community.
- New thematic networks are also being planned and implemented. As soon as there are available tools compliant with a common protocol they could be immediately used by those networks, avoiding subsequent upgrades on installed software.
- The number of data providers, which is already considerable, keeps growing on a regular basis. The later a common protocol is supported by tools and adopted by networks, the more difficult it will be to migrate them in the future.

The protocol being proposed could be supported by existing software on a relatively short term, so that networks could already begin a migration process. Although many features were included, most of them don't need to be implemented from the beginning. Greater levels of functionality can be gradually achieved without breaking conformance with the protocol.

However, further discussions with the community are still necessary to reach a common agreement as to the protocol to be adopted. Whether accepted in its current form or not, this work has been done with the hope to serve as another step towards complete integration of biodiversity data services.

10 References

ABCD: Access to Biological Collection Data. Task Group web site:
http://bgbm3.bgbm.fu-berlin.de/TDWG/CODATA/default.htm

BioCASE: Biological Collection Access Service for Europe. Web site:
http://www.biocase.org/
Protocol XML Schema:
http://www.bgbm.org/biodivinf/Schema/protocol_1_3.xsd

CQL: Open Geospatial Consortium, Common Catalogue Query Language, "Filter Encoding Implementation Specification", Panagiotis A. Vretanos, version 1.0.0, Adopted Specification, 19 September 2001.
http://www.opengis.org/docs/02-059.pdf

DarwinCore: A set of data element definitions designed to search primary biodiversity data. Task Group web site:
http://darwincore.calacademy.org

DiGIR: Distributed Generic Information Retrieval. Web site:
http://digir.net
Protocol XML Schema:
http://digir.net/schema/protocol/2003/1.0/digir.xsd

GML: Open Geospatial Consortium, "Geography Markup Language Implementation Specification", Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, Arliss Whiteside, version 3.00, 29 January 2003.
http://www.opengis.org/docs/02-023r4.pdf

SDD: Structure of Descriptive Data. Task Group web site:
http://160.45.63.11/Projects/TDWG-SDD/

Semantic Web: World Wide Web Consortium, "Semantic Web Activity".
http://www.w3.org/2001/sw/

SOAP: World Wide Web Consortium, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, 24 June 2003.
http://www.w3.org/TR/soap12-part1/

TCS: Taxonomic Concept Transfer Schema. Task Group web site:
http://www.soc.napier.ac.uk/tdwg/index.php

UDDI: OASIS Standards Consortium, "Universal Description, Discovery, and Integration". Web site:
http://www.uddi.org/

WFS: Open Geospatial Consortium, "Web Feature Service Implementation Specification", Panagiotis A. Vretanos, version 1.0.0, Adopted Specification,

19 September 2002.
http://www.opengis.org/docs/02-058.pdf

WSDL: World Wide Web Consortium, "Web Services Description Language (WSDL) 1.1", W3C Note 15 March 2001.
http://www.w3.org/TR/wsdl.html

XML: World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation, 04 February 2004.
http://www.w3c.org/TR/REC-xml

XML Schema: World Wide Web Consortium, "XML Schema specification (Part0: Primer, Part1: Structures, and Part2: Datatypes)", W3C Recommendation, May 2001.
http://www.w3.org/XML/Activity.html#schema-wg

XPath: World Wide Web Consortium, "XML Path Language (XPath) Version 1.0", W3C Recommendation, 16 November 1999.
http://www.w3.org/TR/xpath

XQuery: World Wide Web Consortium, "XQuery: A Query Language for XML", W3C Working Draft, 23 July 2004.
http://www.w3c.org/TR/xquery/