# TAPIR - TDWG Access Protocol for Information Retrieval

## Network Builders' Guide

---

**TDWG Access Protocol for Information Retrieval (TAPIR)**

**Network Builders' Guide**

**Version:** 1.0 (First Draft)

**Date:** 1 August 2007

**Technical Writers**

- Charles Copp (eim [at] globalnet [dot] co [dot] uk)
- Renato De Giovanni (renato [at] cria [dot] org [dot] br)

**Reviewers**

- Lee Belbin (lee [at] tdwg [dot] org)
- Markus Döring (m [dot] doering [at] bgbm [dot] org)

**Status of this document**

Working draft under revision. This document may be updated, replaced or made obsolete by other documents at any time.

This document provides guidelines for building networks based on the TAPIR protocol. It is a product of the TDWG TAPIR task group, details of which can be found at http://www.tdwg.gbif.org/subgroups/tapir.

Comments about this document can be sent to the TAPIR mailing list: tdwg-tapir@lists.tdwg.org (subscription and archives are open to the public).

---

**Copyright Notice**

# 1. Biodiversity data sharing networks

With the adoption of a world-wide commitment to conserve and enhance biodiversity, the need for accessible, valid and comprehensive wildlife data has never been greater. The requirement to build environmental and biodiversity information into social and economic planning exists at all geographic scales and is increasingly reflected in legislation and public expectation. The need to model and understand the effects of global climate change, population rise, fuel supply and the mitigation of economic development is urgent.

To be able to make sound judgements and engage politicians and public in well informed debate there must be ready access to high quality data. There needs to be a constant supply of new and historical data from collections and publications to provide context. Huge amounts of new primary biodiversity data are being collected, and the digitisation of existing data from field surveys and museum collections is advancing in almost every country. However, the data resource remains fragmented, isolated by the software used to store it, and by the ability of providers to make their information available on-line in a suitable format and using compatible terminology. Widely distributed data can only be integrated and delivered to an increasing audience through a system of integrated data networks.

The development of data sharing networks for mobilising primary biodiversity data is being tackled in many ways. These networks operate from local to the global scale and include both generalised and thematic networks. National networks include the United Kingdom's National Biodiversity Network (NBN) and Mexico's CONABIO, both of whom have extensive international links. At the international level the two best known general biodiversity networks are those based on the use of the DiGIR and the BioCASe protocols. DiGIR, used primarily in North America, works with the DarwinCore exchange format, while BioCASe, primarily used in Europe, works with ABCD. Large scale thematic networks include the Ocean Biogeographic Information System (OBIS), speciesLink, FishNet2, HerpNet and the Mammal Networked Information System (MaNIS). The Global Biodiversity Information Facility (GBIF) has been steadily developing since 2004 and integrates data from many networks.

The next sections describe some of the strategies and the main service components that can be used to build networks for searching and retrieving data across multiple providers.

## 1.1. Service Oriented Architecture

The ubiquitous nature of the "Web" has allowed the development of a Service Oriented Architecture (SOA) approach in both software and business development. The SOA approach emphasises the building of networks based on so-called loose coupling between software agents. This essentially means that instead of users interacting with single monolithic applications that handle all aspects of data, data management, user interfaces and communications, the system is broken down into a number of processes and services (a service being a unit of work performed by a component to produce specific results for a user). Loose coupling refers to the convention that data and processing do not have to be bound together, nor do the components of the system need to know how each other work. Software agents use discovery services to locate data and processing services, and need to know only how to transmit messages based on a small set of standardised interfaces and external schemas. In a Web-based service network, requests for services and delivery of services are done through the medium of XML over HTTP, FTP or SMTP. The big advantage of the SOA approach is that it allows the development of extensible systems that can steadily add new providers and services using different implementations. The end user does not have to understand the structure of the system or where the data or services are located and each participant is only concerned with how their part of the system works. For the end user this may be a simple browser page.

## 1.2. Existing approaches

By its nature, a biodiversity data sharing network normally needs to link a heterogeneous array of data sources. Data providers may be distributed across a wide geographic area and physically located on hardware ranging from simple PCs to enterprise scale servers. Data may be managed in a wide range of database or other software applications, running on different operating systems. A simple directory service portal might store or access metadata about data providers and link users through local Web sites and services. The collation and marshalling of data from multiple sources to provide users either with raw data or Web services based on the collated data requires different strategies and software.

There are two main strategies that can be used: Build a distributed query system, or copy provider data to a central data cache and run queries against the cache.

The advantage of a distributed query system is that queries are passed on to the data providers, which spreads the processing load and gives providers a greater sense of control over their data. In distributed query systems, providers can decide whether they wish to answer any particular query, or maybe place limits on the numbers of returned rows. Individual providers may also have interfaces to various portals and offer their own Web services to partners or customers, all from the same data store. This ensures that provider records are consistently updated. By placing the query answering and processing load with the data providers, it is possible to develop rich networks that are open to many types of client and that offer a diversity of services.

The disadvantage of distributed systems is that they can become very large and the state of the system may change frequently. The site load (number of sites that may be queried) can vary as nodes (providers) are added, removed or simply go off-line. For the portals that marshal data resulting from queries cascaded to multiple providers, this can represent a significant problem. Provider sites need to be regularly polled to see if they are online. Ideally, an indexing system should be maintained to filter queries to the relevant providers. Query times may also be affected by time-outs and slow processing by providers.

Some operations are not even possible in a distributed scenario. For example, the ordering of the combined results of a federated query within the network can only be accomplished with caching infrastructure.

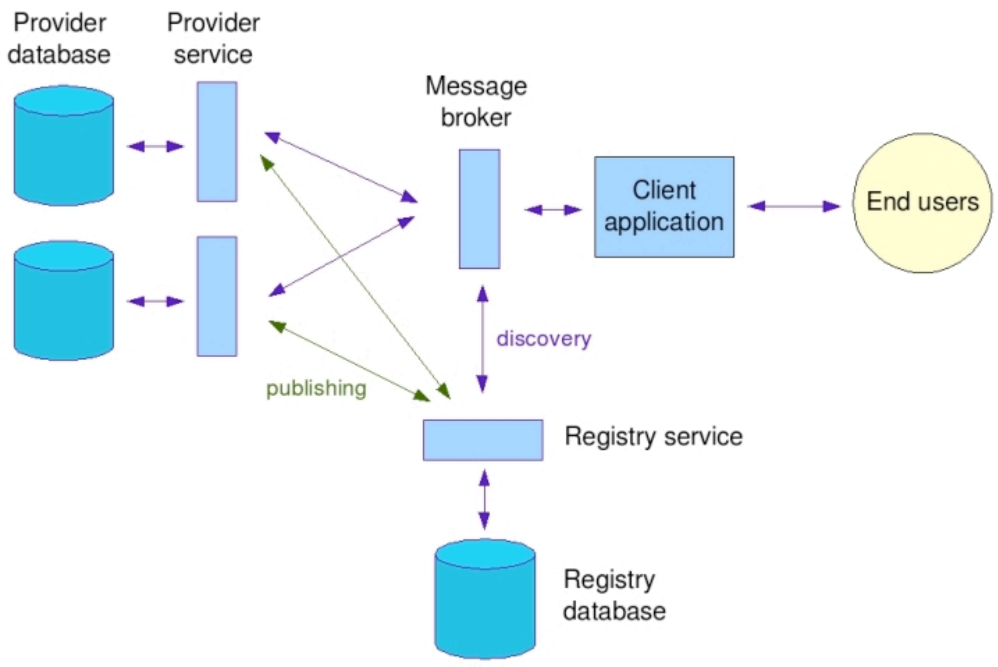

**Figure 1.** *Components of a distributed biodiversity data network. Although the various datastores and services are shown separately, some may be housed on the same site and several services may be performed*

*by one application.*

The alternative strategy is to harvest data from providers to create a central data cache against which queries can be run. Whereas data providers may have a great range of data structures and file types, the data cache can have a simple and unified structure that is optimised for retrieval. A cache significantly improves the service to clients. The cache may be updated in various ways. In the UK NBN Gateway system, data providers manage their own data on the cache, deciding when they will update it and what restrictions they wish to place on it (or even removing it completely). In systems such as that developed for BioCASE and for GBIF, harvester programs send queries to providers to automatically retrieve both metadata for indexes and actual data for the central cache.

One of the disadvantages of a centralised system is that queries are performed against a cache that is potentially not up-to-date. GBIF currently indexes up to 30 million records per day using 50 parallel processes. This gives an indication of the cost and complexity of updating the cache. Some data providers may also fear lack of recognition or loss of control of their data (e.g. by not being able to count hits on their own system) which may affect relationships with funding bodies. There is also the problem of managing a potentially very large cache that may require mirroring to other sites with concomitant need for synchronisation.

In a complex network, both central cache and distributed query strategies can be used, especially if queries are cascaded to various data aggregators as well as single data providers. Mixed architectures are also possible, like the approach taken by the speciesLink network, where several regional servers mirror data from local providers that cannot serve data directly to the network.

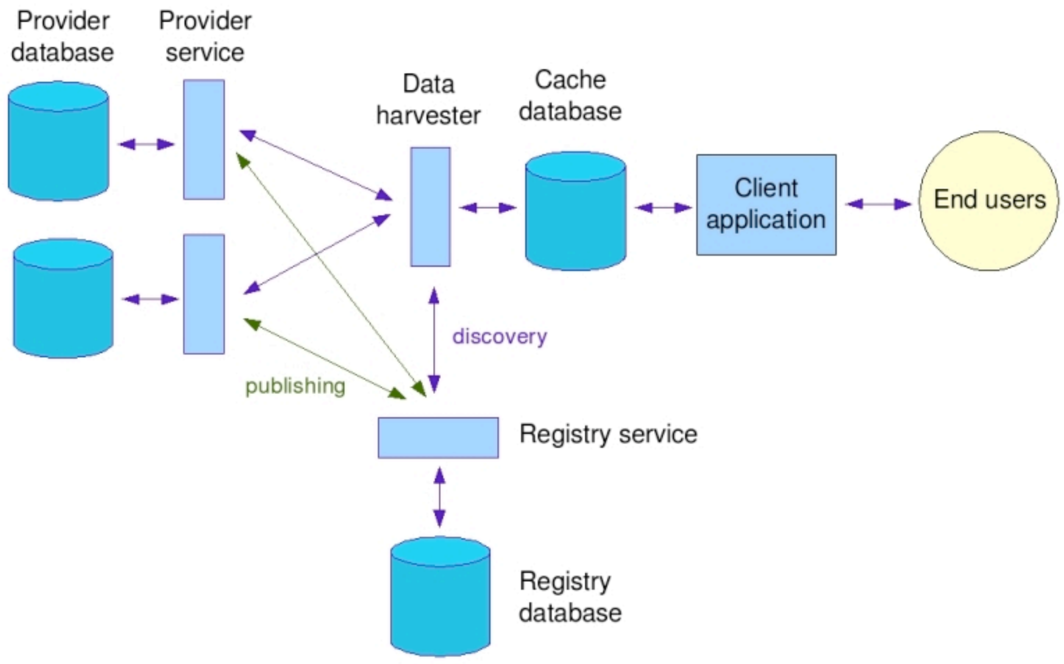

**Figure 2.** *Components of a cached biodiversity network.*

## 1.3. Main components

Establishing a Web-based biodiversity data network relies on establishing the basic framework of service providers, discovery services and software agents. The primary services are the data providers that respond to user queries, but many other services may be involved. These services may include a directory of participant providers, a data aggregating service, thesaurus and name services, or other specialised data processing services.

A functioning biodiversity data network delivering information from diverse sources to a range of different types of users usually requires the following components:

- User interfaces
- Message brokers and portals
- Data aggregators and harvesters
- Directory services
- Data provider software

Additional components and services can also be used in different parts of the network. Examples include taxonomic name servers, data transformation services, data validation services and mapping services. In this document the focus will be on the main components that are needed to provide search and retrieval functionalities.

### 1.3.1. User interfaces and Web portals

In a biodiversity data network scenario, there can be different places where people interact with specific parts of the system. This may range from configuration interfaces for some components to the main user interface - the one which is exposed to end users and provides the means for interacting with the whole network.

User interfaces include for example simple command-line interfaces, graphical user interfaces and Web-based interfaces. Biodiversity data networks usually want to reach a wide range of users, so the most common interface in this case is a Web-based interface through global or thematic Web portals.

A Web portal is a "hub" Website that provides an interface for users to reach or query a number of distributed sites, services and applications. Simple portals may behave as little more than directories to other sites but the portals used for establishing biodiversity data networks use various types of middleware, such as directory services, messaging and marshalling services to deliver collated data, and additional data services, such as mapping and data cleaning tools.

Portal Websites for biodiversity networks typically include many sections such as news, forums, document archives and directories. From the data network perspective, the key functionality is the query interface. The query interface is a critical part of the network because it is the means through which users discover what can be searched for and the means to construct queries. To do this successfully, the interface must make it simple for users to enter the correct search terms and also provide enough feedback and contextual information so that they are not continually frustrated by negative or misleading results.

As an example, biodiversity data is linked principally to the names of organisms but users may not know the current taxonomic names. Users and some of the data providers may use common names or different names and languages for the same thing. This raises issues of how to guide the user when selecting a name to enter (through the interface design) and how the many synonymous or potentially synonymous names are reconciled between databases, especially those with a significant amount of legacy data linked to old specimens. Effectively handling data disparity with a clean and intuitive user interface is a major challenge. In some of the simpler cases, user interfaces may offer the user nothing more than a blank text box in which they must type an exact taxonomic name. In other situations, such as the NBN Gateway, users can type any name or part of a name and the underlying thesaurus will attempt to interpret it using taxonomic and common names and synonyms. The user can make use of an alphabetical pick list to select a name, ensuring its correct spelling. Some sites, such as the Northern Ireland CEDAR/Habitas site use high quality photographs to aid species selection - a great benefit to skilled users as well as the general visitor.

Besides providing ways of interacting with end users, the user interface also needs to perform at least one of the following tasks:

- Communicate with a central database that aggregates data from all providers.
- Communicate directly with data provider services.
- Communicate with a message broker service.

### 1.3.2. Message brokers

Message brokers are a type of software called "middleware" and are a key element in enabling the integration of data in heterogeneous networks. Middleware is simply software that connects other software together, usually working as a hub between many applications or clients.

A message broker is basically a means of conveying a search request from a client software to one or more database applications. Typically, message brokers allow queries and data to be sent back and forth between applications in an asynchronous fashion. This means that messages may be received, possibly processed and then held in a queue for forwarding to recipient service end points (such as database wrappers).

Message brokers may provide a complete messaging system, the routing, and sometimes extra data conversion intelligence needed to pass sender messages in the appropriate format to the right recipients via an independent message protocol.

The BioCASE network uses a message broker, referred to as the Unitloader. The Unitloader software generates protocol documents and distributes them to data providers via threads. The Unitloader does not include its own registry and needs to be provided with a list of providers that are to be queried. In the DiGIR network, the message broker is known as the Portal Engine. This component can either be manually configured with a list of provider addresses or make use of a UDDI registry service to discover them.

### 1.3.3. Data aggregators and harvesters

Data harvesting is the process of gathering data from disseminated data sources into a single database that can be republished or directly searched. Harvesting may require the restructuring of data from the source data model to match a common union schema. For this purpose, data sometimes need to be "translated" into a single unified XML format.

Data harvesting can be related to several of the functions in a distributed biodiversity data network. One of the key roles is gathering metadata for the purposes of documenting and indexing the network. Metadata can be presented and stored in many ways. For instance, information related to an organisation and its data holdings can be placed in the META section of an organisation's Website homepage. This information can be harvested automatically by "Web-crawling" software or search programs working from a directory of participants. More sophisticated parsing programs may parse a whole Web page and its related pages to look for indexable keywords. A related means of harvesting metadata utilises RDF. The partners publish their metadata in RDF format at a URL which will be listed in a directory service (such as UDDI). Harvester software may then collect the RDF statements and store them as a metadata cache in a suitable database such as an RDF triple store. Metadata may also be held in a data provider's database and accessed by some specific query transmitted to the provider using HTTP. In all of these examples the role of the harvester is to automatically collect metadata and update a central cache.

As noted above, a central cache may also be utilised by a biodiversity data portal to create a "hub" cache database of actual field data to speed up responses. This process may be automatic: for example, trawling partner databases on a nightly basis to harvest new or changed records.

A data aggregator is also a program or service that collects information and resolves any contextual differences between various sources. Typically data aggregation services act by linking many data providers, performing transformations, and sometimes analysis services on the aggregated data. Results are then presented as report or service to the client. A typical example from the commercial world is the functioning of price comparison Web sites. These sites trawl numerous suppliers, rank the results, and provide a link through to suppliers to aid purchasing. A non-caching biodiversity query portal works as an aggregating service, gathering data from numerous providers. Data is normally transformed before it can be used to create a unified response. This may be handled by an aggregator or by wrapper programs attached to each database.

### 1.3.4. Directory services

Directory services are key components of biodiversity data networks. The directory holds contact information for organisations and services available to the network and metadata describing the services and the extent of data holdings. This information can be used by the portal query marshalling program to decide which data providers to forward queries to. For example, such directory services may avoid sending a query related to living ants to a purely botanical or palaeontological data provider.

The Directory for the network may be a bespoke database associated with the portal. This database may contain contact information and metadata that can be queried to provide users with details of data sets and sources without actually querying the data. The database can be set up so that participating data providers can manage their own metadata; otherwise metadata should either be harvested from participant Websites or queried from on-line databases.

An alternative to bespoke metadata databases is the use of UDDI (Universal Description Discovery and Integration), which is a platform independent XML-based Internet registry model. UDDI was originally conceived as a linking and discovery system for businesses, providing three levels of data known as white pages, yellow pages and green pages using a telephone directory analogy.

White pages provide the contact details and known contact points whilst the yellow pages categorise the available services using standard classifications or ontologies. The green pages provide technical details of the actual Web services provided by the businesses and describe their interfaces using WSDL (Web Services Definition Language). UDDI has had a mixed response from the competitive business world because a widely used registry allows a quick and easy way to switch services as well as discover them. UDDI does however have valuable potential for co-operative networks such as GBIF. UDDI uses XML and SOAP in its message system and provides a standard data model and API for Web service providers to link to. There are now numerous commercial UDDI registry services available but a network can establish its own service using available published standards and code.

The usage of directory services increases the complexity of the network by including a new component with a different protocol. But the advantage of using a public registry is that participants can register themselves and manage their own details independently of the query service. This is convenient for large, general content, dynamic networks. For smaller biodiversity networks, a UDDI system might not be convenient. In this case, a more specific registry database or a simple configuration file that is closely coupled to the network hub (portal software or cache database) may be easier to maintain and be more efficient. The choice depends on how the network is likely to grow and what central support is available for maintaining data and software.

### 1.3.5. Data provider and wrapper software

Biodiversity data are held by many thousands of individuals and organisations, each with their own specific needs and history of data management. There has been much effort in the last decade to develop common data models and applications suitable for managing biodiversity data. However, data resources remain scattered in many different formats. The aim of creating a biodiversity data network is to make the combined information from these various sources available as integrated, user-focused views through easy-to-use interfaces.

Access to the data held by data providers is achieved through the use of wrapper programs that extract data automatically from the target data store in a given format, in response to a given query. Target data stores can be relational databases, triple stores, XML database, etc. This process requires a number of steps. An incoming query must be parsed and analysed to convert it into a local query that will be understood by the underlying database management system. This process may include the mapping of concepts used in the incoming query to those used locally. The second stage involves building and running the local query. This stage may run against a single underlying datastore or it could be run against several. The results of the query must then be organised. This may include integration and transformation to a format specified by the original incoming query.

Database wrappers are used in some form by all database-driven Websites but these may be mostly very specific to the application and not readily transportable to other data sources. This situation lead to the development of special Data Provider Software for the emerging biodiversity networks. DiGIR and BioCASE networks make use of two best known Data Provider packages: DiGIR PHP Provider and BioCASe PyWrapper.

## 1.4. Message exchange protocols

The various components of the network must communicate with one another and with other Web-based networks. Communication is normally achieved using an XML-based message protocol, with HTTP as the transmission method.

The SOA model is structured on the concept of passing requests for services and then getting back responses using widely adopted message protocols. Sometimes the protocols are built on top of other protocols, like SOAP (Simple Object Access Protocol). SOAP is a generic, extensible XML-based messaging protocol and is used in wide number of applications that access distributed data and services. SOAP is very well supported and SOAP-wrappers are available for most programming languages. There are other protocols that claim advantages in some business applications (such as CORBA). There are also domain-specific protocols that use HTTP+XML such as DiGIR (Distributed Generic Information Retrieval Protocol) and BioCASe (The BioCASE Access Protocol).

The DiGIR and the BioCASe protocols are the two most widely used protocols for biodiversity data networks. The DiGIR Protocol references DarwinCore, an extensible list of concepts, as its default data schema. The BioCASe protocol uses a much larger, hierarchical schema called ABCD. Both DiGIR and BioCASe protocols are recognised and used by the GBIF indexing and caching software. For comparative details see: GBIF Data

Access and Database Interoperability: "A Unified protocol for search and retrieval of distributed data". September 2004. Markus Döring and Renato De Giovanni.

TAPIR (the TDWG Access Protocol for Information Retrieval) is a recent unified protocol that can replace or work alongside existing biodiversity protocols. TAPIR specifies a standardised, stateless, HTTP transmittable, XML-based request and response protocol for accessing structured data that may be stored on any number of distributed databases of varied physical and logical structure. TAPIR requests can be formulated as either XML documents, which may be transmitted using HTTP POST or encoded as Key-Value Pairs in HTTP GET messages. The key feature of TAPIR is that it acts as a framework for external schemas, query definitions and output models. This means that DarwinCore, ABCD and other schemas can be supported.

# 2. TAPIR networks

All architectures and components that were previously described can be used by TAPIR networks. TAPIR networks are defined by the use of the TAPIR protocol for communication with data providers. Interaction with any of the other components is not covered by the TAPIR protocol and should therefore be done in different ways.

The TAPIR Specification describes the message format for operations that can be invoked in both Key-Value-Pair (KVP) and XML requests. These operations are:

- **Metadata:** Default operation used to retrieve basic information about the service.
- **Capabilities:** Used to retrieve the essential settings to properly interact with the service.
- **Inventory:** Used to retrieve distinct values of one or more concepts.
- **Search:** Used for search and retrieval of data.
- **Ping:** Used for monitoring service availability.

These operations can be used harvesting metadata and data, performing searches and data mining. TAPIR provides the means for relaying data from providers to a client but TAPIR does not amalgamate data from multiple sources. This must be done by another component (e.g. a query marshalling service). Therefore, when a message broker is used in the network, it is necessary to define a separate protocol for the communication between client software and message brokers. Such a protocol would typically accept multiple final destinations in requests to the message broker, and then expect a collection or an aggregation of TAPIR responses in return.

## 2.1. Managing and discovering participants

TAPIR networks need a way of defining their participants (the data providers). This can be done by means of a directory service or with simple manual configuration files. When the number of participants is significant and diverse (different protocols and types of data), a registry service is usually recommended. Before setting up a new registry service, it is recommended to first check if there is a similar service already being used by the community, as the main advantage of a registry is to contain the maximum number of services from a particular community. This way there can be a single registration process for data providers and a single discovery interface for users and client software. It is also important to note that registry services usually offer the possibility of associating classifications with each data provider, therefore allowing the definition of thematic networks.

In registries like UDDI, it is necessary to register the corresponding type of service before adding a new data provider. Registration usually includes the service name and ideally the service interface (a machine-readable definition of the service). For Web Services the current standard is to use a WSDL file. TAPIR does not have an associated WSDL file due to the complexity and flexibility of the protocol. It is however possible to define a WSDL for TAPIR query templates.

## 2.2. General guidelines

Three important aspects must be considered before setting up a TAPIR network:

- The protocol itself is independent from the semantics of the network

  The approach taken by TAPIR and its predecessors (DiGIR and BioCASe) to handle the heterogeneity of multiple data providers is to require the definition of one or more federated schemas (conceptual schemas) representing the semantics of the network. When all participants of a network adopt the same federated schemas, it is possible to formulate queries and output models based on them. This approach makes the multiple providers look like a single uniform database. Software used by data providers will translate the queries into to the local query language and the local database structure. To do this, part of the configuration process of provider software is to map local databases against one or more conceptual schemas. This means that TAPIR is a generic protocol and cannot be used if there are no conceptual schemas previously defined.

- The semantics of the network can be independent of output models

  Defining the semantics (conceptual schemas) of a network may not be enough if this definition does not also serve as an output model definition. Output models are the types of output that will be returned by data providers. When a conceptual schema coincides with an output model, it is known as a canonical model. This can happen when the conceptual schema is defined using an XML Schema, which is also the language used to define response structures. In this case the conceptual schema can be used as the basis for an output model. However, it is important to know that it is possible to define different output models based on the same conceptual schema. Part of the task of setting up a TAPIR network involves the definition of output models and sometimes query templates based on these models.

- There are different levels of provider implementation

  There are different levels (or categories) of TAPIR providers in order to simplify the implementation of provider software. In general, networks that require greater flexibility will need to use more complex software. The fact that there are different levels of provider implementation means that there can be interoperability issues. Therefore, setting up a TAPIR network also requires defining the minimum protocol capabilities that need to be supported by all participants.

The basic requirements for setting up a TAPIR network therefore include the definition of at least one conceptual schema, one output model and the minimum capabilities for participant providers.

## 2.3. Conceptual schemas

Conceptual schemas can be seen as data models with a formal definition of classes (entities), their characteristics (attributes) and how classes relate to each other (relationships). In biodiversity networks, the things of interest include wildlife and earth science specimens, observations of these specimens, and their associated measurements, descriptions and identifications. Locations, people and documents are also part of this context.

There can be multiple uses of conceptual schemas, such as in database design, class modelling and data exchange. For TAPIR, conceptual schemas represent a data abstraction layer that allows multiple heterogeneous data providers to be seen as if they were part of a single large database.

Concepts are elements that belong to a conceptual schema. In theory, concepts can be of any kind - an entity, an attribute or a relationship - but this version of the TAPIR protocol only considers concepts that are related to attributes. This means that the use of other types of concepts within TAPIR will be considered experimental.

Conceptual schemas are the only means to define the semantics of a TAPIR network. Therefore, while TAPIR is not bound to any particular conceptual schema, at least one schema is required. When a TAPIR data provider advertises in the capabilities response that it knows certain concepts, this usually means that each concept corresponds to a field in the local database. This is the reason for generic TAPIR data provider software to include a "mapping" process between one or more conceptual schemas and the local database. Provider software may also allow concepts to be mapped in different ways, such as using a concatenation of fields, predefined values, etc.

### 2.3.1. Existing conceptual schemas

Since TAPIR derived from two protocols that were used by specimen data networks, the most well-known conceptual schemas for TAPIR come from these networks. Both ABCD and DarwinCore have been created with the purpose of providing a general data exchange format for biological collections. Although both are defined as XML Schemas, each follows a different approach.

ABCD officially started in 2000 but has its roots in data modelling activities from the early 1990s. ABCD has a comprehensive and highly structured XML Schema. ABCD tries to cover all possible aspects that can be related to biological records and it makes use of strict controlled vocabularies. Since ABCD was purely conceived as a data exchange standard through XML Schema, it only defines the syntax of valid ABCD documents. There are no formal definitions of entities and how they relate to each other, although XML Schema complex types are used to define the major entities (such as identification and taxon name). When used both as a conceptual schema and as a TAPIR response structure, ABCD can be seen as a canonical schema. ABCD concept identifiers are defined according to the TAPIR recommendation for XML Schemas as the concatenation of the target namespace with the local xpath to instance nodes. Only nodes related with content are being considered.

So the element "/DataSets/DataSet/Units/Unit/InstitutionID", which is part of ABCD version 2.06, becomes a concept identified by:

http://www.tdwg.org/schemas/abcd/2.06/DataSets/DataSet/Units/Unit/InstitutionID

DarwinCore was originally created to be used by the DiGIR protocol. In fact, the first versions of DarwinCore were bound to the DiGIR protocol through XML Schema substitution groups. The recent version of DarwinCore removed this dependency and keeps the same approach of defining a short list of concepts in a DublinCore style. DarwinCore focuses only on the basic content about the geographic occurrence of organisms and the physical existence of biotic specimens in collections. By design, DarwinCore is also minimally restrictive of information content so that it can also be used by data quality tools. The DarwinCore XML Schema defines a flat list of global elements that represent attributes of biological records. In terms of XML Schema, DarwinCore cannot be directly used as a data exchange structure since only one root element is allowed in XML instances. This means that it is necessary to define another XML Schema (known as an application schema) referencing DarwinCore elements from one or more grouping elements.

DarwinCore concept identifiers are also defined as the target namespace concatenated with the element name. So the concept "GlobalUniqueIdentfier", which is part of DarwinCore 1.4, becomes:

http://rs.tdwg.org/dwc/dwcore/GlobalUniqueIdentifier

Both ABCD and DarwinCore have extension mechanisms. ABCD can be extended by defining extra elements inside specific extensible slots located in the ABCD structure. In DarwinCore, it is possible to define additional schemas to cover new concepts following the same DarwinCore format. The new version of DarwinCore is comprised of a core schema and two extensions: the curatorial extension and the geospatial extension.

The TDWG architecture group has defined an ontology for the biodiversity domain. This ontology is being developed in OWL and includes formal definitions of classes and properties (which can represent attributes or relationships between classes). The TDWG ontology should soon include concepts related to all TDWG standards.

### 2.3.2. Creating new conceptual schemas

Before creating a new conceptual schema, it is highly recommended to check if the community is already using a particular standard for the same purpose. Adopting an independent conceptual schema may cause interoperability issues with related networks. TAPIR providers can only understand queries that reference known concepts (as advertised by capabilities responses), so a TAPIR client cannot send the same request to TAPIR providers that understand different conceptual schemas.

If an existing conceptual schema does not satisfy the needs of the network, it may be better to extend the schema rather than creating a new one. Adopting a standard and extending it means that there can be full interoperability at least at the standard level among all networks that are using it.

The TAPIR specification does not define any particular format for conceptual schemas. TAPIR only expects that a list of concept identifiers can be extracted from a conceptual schema so that they can be referenced by TAPIR messages. Although DarwinCore and ABCD are defined with XML Schemas, other formats such as RDF Schema, OWL and XMI could be used to create conceptual schemas. If there is no suitable conceptual schema being used by the community, choosing the format of a new conceptual schema may depend on other needs not directly related to TAPIR.

If the network just needs a single structure to be returned by TAPIR searches in plain XML, the conceptual schema could be defined in XML Schema. In this case, the XML Schema will serve both as a conceptual schema and a response structure, as with BioCASE-ABCD. However,recursive elements should be avoided when using XML Schemas, since there is no standard procedure in TAPIR to derive concept identifiers using recursion.

If the network requires multiple response structures, and especially if these will be conformant with the Semantic Web (for instance using RDF), conceptual schemas using RDF Schema or OWL are recommended.

Regardless the format of conceptual schemas, if there is the possibility for the network to require that providers' responses be used by Semantic Web applications, it is recommended to add a slash or a hash in the end of the conceptual schema namespace. The reason is that Semantic Web tools will often concatenate the namespace with local names, and this will generate predicate URIs that can cause problems in situations like RDF/XML round-tripping. If these predicate URIs will also be used as TAPIR concept identifiers, then it is preferable to have the slash as a separator, since hashes have a special meaning in the query term of URLs.

The TAPIR specification recommends concept identifiers to be:

1. Globally unique, to avoid any conflicts with other concepts;
2. Permanently resolvable by means of standard Internet protocols, so that more information about the concept can always be retrieved;
3. Free from characters that are not allowed in the query term of URLs, so that it is not necessary to escape characters when concepts need to referenced in KVP requests.

When the conceptual schema is defined by an XML Schema, TAPIR recommends that concept identifiers are generated by concatenating the target namespace with the xpath of instance nodes, as exemplified in the previous subsection about existing conceptual schemas.

When the conceptual schema is defined with RDF Schema or OWL, as with the TDWG Ontology, then the process of defining concept identifiers is still under discussion and there may be more than one alternative (see: http://wiki.tdwg.org/twiki/bin/view/TAPIR/TapirRdf).

Concepts can also be defined in a simple way if they are only used by TAPIR. In this case, concepts can be defined through Concept Name Server configuration files.

### 2.3.3. Concept Name Servers

The fastest way to define a list of concepts to be used by a TAPIR network is using a Concept Name Server (CNS) configuration file. The CNS file is a simple text file with specific sections defined by names in square brackets followed by a list of key value pairs. The section called "concept_source" includes information about a conceptual schema. The section "aliases" includes the respective concept identifiers that belong to the conceptual schema along with their aliases. The following example shows a simple and fictitious conceptual schema to be used by a network that wishes to search and retrieve data about species red lists.

```
[concept_source]

label      = Red List Schema v1.0
namespace  =  http://example.net/redlist/
alias      = redlist
location    =  http://example.net/redlist_1_0.txt

[aliases]

DataProviderCode  =  http://example.net/redlist/DataSourceCode
ScientificName     =  http://example.net/redlist/ScientificName
RedListCategory    =  http://example.net/redlist/RedListCategory
Country             =  http://example.net/redlist/Country
AssessmentDate     =  http://example.net/redlist/AssessmentDate
Assessor            =  http://example.net/redlist/Assessor
```

Inside [concept_source] there is a label for the conceptual schema, a namespace (in this example following the recommendation of ending with a slash), an alias, and finally the location of the conceptual schema. The location will usually point to an ".xsd" file, or sometimes a ".rdfs" or ".owl" file. However, since TAPIR does not mandate any particular format, a simple ".txt" file was used here. This text file could just list the concepts identifiers and associated information.

In the last example, all concept identifiers follow the same pattern in accordance with the TAPIR recommendations. They are globally unique (in this case through HTTP URIs), permanently resolvable through standard Web protocols (the URLs could resolve to more documentation about each concept) and without having any special character.

CNS files are also used to advertise concepts from multiple schemas, serving as a kind of network configuration file. Even if conceptual schemas are defined in a specific format, it is recommended to also represent the concepts using a CNS file. The reason is that most generic data provider software will be able to extract concepts and conceptual schemas from CNS files, while the same may not be true for other formats.

## 2.4. Output models and response structures

TAPIR search operations directly or indirectly expect an output model as a parameter. Output models are defined in XML and their definition can either be embedded in TAPIR requests or referenced as external XML documents. The purpose of output models is to indicate how the search response should be structured and to allow paging and counting, so it includes:

- An XML Schema defining the response structure.
- A pointer to an indexing element in the response structure that will serve as a basis for counting and paging.
- A mapping between data nodes in the response structure and the associated concepts that provide meaning.

Output models therefore depend on one or more conceptual schemas to indicate the semantics of data nodes in the response structure.

Response structures can also be embedded in output models or can be referenced as external XML Schema documents. Keeping response structures as external documents can be convenient because:

- They can always be directly edited by any XML Schema editor software such as XML Spy, <oXygen/>, Stylus Studio and xmlDraft.
- If the data providers are using similar but not identical conceptual schemas, it is possible to have different output models, all pointing to the same external response structure, but each one mapping the data nodes to different (equivalent) concepts. Although search requests will be slightly different for these providers because of the different output model parameters, this approach can help getting back the same type of response from the different providers.

The following example shows a possible response structure that could be used to return results about species red list data:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://example.net/redlist"
              elementFormDefault="qualified">
   <xs:element name="redlist">
     <xs:complexType>
       <xs:sequence>
           <xs:element name="source" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                  <xs:element name="taxon" maxOccurs="unbounded">
                   <xs:complexType>
                     <xs:sequence>
                         <xs:element name="name" type="xs:string"/>
                         <xs:element name="category" type="xs:string"/>
                         <xs:element name="country" type="xs:string"/>
                          <xs:element name="assessor" type="xs:string" minOccurs="0"/>
                         <xs:element name="date" type="xs:date" minOccurs="0"/>
                     </xs:sequence>
                   </xs:complexType>
                </xs:element>
              </xs:sequence>
                <xs:attribute name="code" type="xs:string" use="required"/>
            </xs:complexType>
          </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

As it can be seen from the example, a TAPIR response structure consists of an XML Schema. All rules for creating an XML Schema are defined by the W3C XML Schema specification. Although any kind of XML Schema could be used, it is recommended to limit the use of XML Schema features to a specific set of constructs. The TAPIR specification refers to this set of constructs as the "basic schema language". The reason for this recommendation is that while some data providers may hard code the way that search results will be produced to conform to the specified response structure, others will try to dynamically parse the XML Schema and automatically produce the results. Since the XML Schema language is quite complex, algorithms to generate XML instances based on XML Schema can be complex. To simplify this task, the "basic schema language" as defined by the TAPIR specification includes the following constructs of XML Schema: targetNamespace definition, element definition (including minOccurs and maxOccurs), attribute definitions (including attribute "use"), local definitions of complexType and simpleType, sequences, and the "all" definition. By working with simple response structures, a greater number of data provider software can be used.

If the response structure includes more than one global element, TAPIR providers will always render the first one. In the example there is only one global element, so the root element in XML instances will always be

<redlist>. This element may have multiple <source> subelements, which in turn may have multiple <taxon> elements. The <source> element has a mandatory attribute called "code", and the <taxon> element has five subelements: name, category, country, assessor and date. The XML Schema attributes "minOccurs" and "maxOccurs" define cardinality, so the elements <assessor> and <date> are optional (the default values for "minOccurs" and "maxOccurs" is 1).

It is important to know that generic TAPIR provider software may not always produce XML instances that are valid according to the response structure specified in the request. This situation may occur when the response structure includes constructs that are not supported by the software. For instance, the previous example includes an XML Schema data type that is not part of the basic schema language: "xs:date". Therefore, a generic data provider software that only understands the basic schema language will not check if the content of each <date> element is valid according to the "xs:date" data type. In these cases, and when the network will work with a fixed set of response structures, data providers may want to check that their data conforms to the expected data types when configuring provider software.

Since response structures only define the syntax of valid XML instances, it is necessary to indicate the semantics of data nodes in a machine-readable way. This is done through an output model, more specifically in the <mapping> section. Assuming that the previous response structure was made available in the address **http://example.net/redlist/redlist.xsd**, it could be referenced by an output model and combined with the same concepts defined in the previous section in this way:

```
<?xml  version="1.0"  encoding="UTF-8"?>
<outputModel  xmlns="http://rs.tdwg.org/tapir/1.0"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                    http://rs.tdwg.org/tapir/1.0/tapir.xsd
                                    http://www.w3.org/2001/XMLSchema
                                    http://www.w3.org/2001/XMLSchema.xsd">

   <structure  location="http://example.net/redlist/redlist.xsd"/>

   <indexingElement  path="/redlist/source/taxon"/>

  <mapping>
     <node  path="/redlist/source/@code">
        <concept  id="http://example.net/redlist/DataSourceCode"  required="true"/>
    </node>
     <node  path="/redlist/source/taxon/name">
        <concept  id="http://example.net/redlist/ScientificName"  required="true"/>
    </node>
     <node  path="/redlist/source/taxon/category">
        <concept  id="http://example.net/redlist/RedListCategory"  required="true"/>
    </node>
     <node  path="/redlist/source/taxon/country">
        <concept  id="http://example.net/redlist/Country"  required="true"/>
    </node>
     <node  path="/redlist/source/taxon/assessor">
        <concept  id="http://example.net/redlist/Assessor"/>
    </node>
     <node  path="/redlist/source/taxon/date">
        <concept  id="http://example.net/redlist/AssessmentDate"/>
    </node>
  </mapping>

</outputModel>
```

The previous output model also defines that the element <taxon> will be the indexing element, so all paging and counting parameters will be related to this element.

There are advantages in defining output models as external documents such as in the previous example. The main advantage is that external documents can be referenced by KVP requests. Assuming that the previous output model was made available in the address **http://example.net/redlist/model.xml**, it could be referenced by a KVP request in this way:

```
http://example.net/myprovider?op=search&model=http://example.net/redlist/model.xml&
filter=http://example.net/redlist/RedListCategory+equals+"Endangered"+
and+http://example.net/redlist/Country+equals+"Brazil"
```

This example searches for endangered taxa in Brazil, and the result is formatted according to the specified output model. If all providers assign the alias "redlist" to the output model, the aliases "RedListCategory" and "Country" to their respective concepts, and if the abbreviated parameter names are used, the same requests could become:

```
http://example.net/myprovider?op=s&m=redlist&f=RedListCategory+equals+"Endangered"+and+Country+equals+"Brazil"
```

A second advantage of using external output models is related to generic provider software. After parsing output models, they are usually represented as objects which can be serialized and cached for future use. When requests reference output models by location or alias, generic provider software can load cached representations instead of having to parse the output model again, therefore optimizing performance.

However, there may be reasons for embedding output models and response structures in TAPIR XML requests instead of using KVP. For instance, a client application may allow users to choose what fields to return. In this case, output models need to be dynamically built and requests will have to be in XML. The next example shows a TAPIR search request in XML with an inline output model and a simple (tabular) response structure that could be dynamically created by a client application.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="http://rs.tdwg.org/tapir/1.0"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                http://rs.tdwg.org/tapir/1.0/tapir.xsd
                                http://www.w3.org/2001/XMLSchema
                                http://www.w3.org/2001/XMLSchema.xsd">
   <header>
      <source sendtime="2005-11-11T12:23:56.023+01:00"/>
   </header>
   <search>
     <outputModel>
       <structure>
           <xs:schema targetNamespace="http://example.net/redlist"
                      elementFormDefault="qualified">
            <xs:element name="redlist">
              <xs:complexType>
                <xs:sequence>
                    <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                          <xs:element name="source"   type="xs:string"/>
                          <xs:element name="taxon"    type="xs:string"/>
                          <xs:element name="category" type="xs:string"/>
                          <xs:element name="country"  type="xs:string"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
           </xs:element>
         </xs:schema>
       </structure>
        <indexingElement path="/redlist/item"/>
       <mapping>
          <node path="/redlist/item/source">
             <concept id="http://example.net/redlist/DataSourceCode"/>
         </node>
          <node path="/redlist/item/taxon">
             <concept id="http://example.net/redlist/ScientificName"/>
         </node>
          <node path="/redlist/item/category">
             <concept id="http://example.net/redlist/RedListCategory"/>
         </node>
          <node path="/redlist/item/country">
             <concept id="http://example.net/redlist/Country"/>
         </node>
       </mapping>
     </outputModel>
     <filter>
       <and>
         <equals>
             <concept id="http://example.net/redlist/RedListCategory"/>
            <literal value="Endangered"/>
         </equals>
         <equals>
             <concept id="http://example.net/redlist/Country"/>
            <literal value="Brazil"/>
         </equals>
       </and>
     </filter>
   </search>
</request>
```

## 2.5. Query templates

TAPIR networks can optionally define query templates if they wish to avoid dealing with the complexity of TAPIR filters both on the client and the server side. The main purpose of query templates is to allow networks to create one or more types of search or inventory requests with pre-defined filters. Since TAPIR filters can also be parameterised, query templates can be formulated as KVP requests where additional parameters are passed to specify the values to be used in filter conditions. Query templates can therefore be useful in networks that can work with a restricted set of search and inventory requests based on specific filter criteria.

The search request that was given as an example in end of the last section could be transformed into the following query template:

```
<?xml version="1.0" encoding="UTF-8"?>
<searchTemplate xmlns="http://rs.tdwg.org/tapir/1.0"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                       http://rs.tdwg.org/tapir/1.0/tapir.xsd">

   <label>Get list of taxa based on red list category and country</label>

   <documentation>Search taxa by red list category and country. Result is
   ordered by taxon name. Parameter names are "category" and "country".
```

```
    </documentation>

    <externalOutputModel  location="http://example.net/redlist/model.xml"/>

   <filter>
     <and>
       <equals>
           <concept  id="http://example.net/redlist/RedListCategory"/>
          <parameter  name="category"/>
       </equals>
       <equals>
           <concept  id="http://example.net/redlist/Country"/>
          <parameter  name="country"/>
       </equals>
     </and>
   </filter>

   <orderBy>
      <concept  id="http://example.net/redlist/ScientificName"/>
   </orderBy>

</searchTemplate>
```

Assuming that this template was made available in the address **http://example.net/redlist/basicsearch.xml**, the same request used in the last section could be formulated as:

```
http://example.net/myprovider?op=search&template=http://example.net/redlist/basicsearch.xml&category=Endangerec
```

Using abbreviated TAPIR parameter names and assuming all providers assign the "basicsearch" alias for this template, the same request would become:

```
http://example.net/myprovider?op=s&t=basicsearch&category=Endangered&country=Brazil
```

Query templates can therefore be seen as custom APIs to interact with TAPIR providers. It is even possible to associate WSDL files for each query template in a capabilities response.

As with response structures and output models, query templates can either be embedded in XML requests or referenced as external documents as in the previous example. The advantages of using external documents are the same as mentioned in section about output models: the possibility of using them in KVP requests and the perspective of getting better performance when using generic provider software. Although it is possible to embed entire query templates in XML requests, this alternative was made available mainly to keep the protocol symmetric across the different request encodings.

Query templates can also be used to unify access to data providers with different but matching conceptual schemas, such as DarwinCore and ABCD. In this case, two query templates could provide the same API but reference different output models and different concepts in the filter. If providers assign the same alias to these different but equivalent query templates, then all requests could be exactly the same across all providers in the network.

When a TAPIR network is based on query templates, the ways of interacting with providers need to be carefully considered. For instance, the search template that was given as an example would not be sufficient if data providers would also need to be indexed by data aggregators. Data aggregators need a simple way to scan all records from data providers. In this case, inventory templates could help since they can be used to show all distinct values for any concept. The following inventory template does not include a filter or parameters, so it would return all distinct values of taxa that are available from the provider's database:

```
<?xml  version="1.0"  encoding="UTF-8"?>
<inventoryTemplate  xmlns="http://rs.tdwg.org/tapir/1.0"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                          http://rs.tdwg.org/tapir/1.0/tapir.xsd">

    <label>Taxa  inventory</label>

   <documentation>Returns  a  list  of  all  distinct  scientific  names
   that  exist  in  the  database.  No  parameters  or  filter  conditions
    are  present.</documentation>

   <concepts>
      <concept  id="http://example.net/redlist/ScientificName"/>
   </concepts>
</inventoryTemplate>
```

Using the abbreviated TAPIR parameter names and if all providers assign the same alias "nameinventory" for this template, the same request could become:

```
http://example.net/myprovider?op=i&t=nameinventory
```

Data aggregators would then be able to loop over all taxa available from the provider. For each taxon, a request would be sent based on a new search template that could accept as parameters a scientific name and a date or timestamp. The search would retrieve only the records that were changed since the last harvesting process, and return all concepts for each matching record. When data aggregation is envisaged, it is always recommended to have a concept representing the timestamp when records were last updated and a concept for a global unique identifier.

## 2.6. Service profiles

There are different levels of the TAPIR protocol that can be implemented by data provider software. While it may be difficult to implement the entire specification, partial implementations can be useful and easy to implement. TAPIR networks can be established with just a single type of provider software using predefined messages and filters. TAPIR networks can also be highly heterogeneous with some providers able to supply a wide range of complex requests. The extent to which a provider software implements the protocol is expressed by the capabilities response.

Three main levels of service profile were envisaged for data providers:

- **TAPIR Lite:** Supports only KVP request encoding (no need to parse XML requests) and supports only query templates for both inventory and search operations. In this case, there is no need for providers to parse response structures, output models or filters. Based on the template passed as a parameter, providers can "manually" produce responses that comply with the respective output model, without the need to dynamically parse and interpret it.

  The next example shows a capabilities response with a minimum set of protocol features that would typically be supported by TAPIR Lite providers.

```
<?xml  version="1.0"  encoding="UTF-8"?>
<response  xmlns="http://rs.tdwg.org/tapir/1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                 http://rs.tdwg.org/tapir/1.0/tapir.xsd">
   <header>
      <source  accesspoint="http://example.net/tapir.cgi"
               sendtime="2005-11-11T12:23:56.023+01:00">
         <software  name="myTapirProvider"  version="1.0"/>
     </source>
   </header>
   <capabilities>
     <operations>
       <ping/>
       <metadata/>
       <capabilities/>
       <inventory>
         <templates>
              <template  location="http://example.net/redlist/nameinventory.xml"
                      alias="nameinventory"/>
         </templates>
       </inventory>
       <search>
         <templates>
              <template  location="http://example.net/redlist/basicsearch.xml"
                      alias="basicsearch"/>
         </templates>
       </search>
     </operations>
     <requests>
       <encoding>
         <kvp/>
       </encoding>
       <globalParameters>
          <logOnly>denied</logOnly>
       </globalParameters>
       <filter/>
     </requests>
     <concepts>
        <schema  namespace="http://example.net/redlist/"
                 location="http://example.net/redlist_1_0.txt"
               alias="redlist">
           <mappedConcept  id="http://example.net/redlist/DataSourceCode"
                         alias="DataProviderCode"/>
           <mappedConcept  id="http://example.net/redlist/ScientificName"
                         alias="ScientificName"/>
           <mappedConcept  id="http://example.net/redlist/RedListCategory"
                         alias="RedListCategory"/>
           <mappedConcept  id="http://example.net/redlist/Country"
                         alias="Country"/>
           <mappedConcept  id="http://example.net/redlist/AssessmentDate"
                         alias="AssessmentDate"/>
           <mappedConcept  id="http://example.net/redlist/Assessor"
                         alias="Assessor"/>
       </schema>
     </concepts>
     <variables/>
     <settings>
         <maxElementRepetitions>100</maxElementRepetitions>
     </settings>
   </capabilities>
</response>
```

- **TAPIR Intermediate:** Supports a fixed range of output models, so it does not need to dynamically parse an output model and its reponse structure, but it does need to be able to parse query templates. While search operations are only based on known output models, features like partial node selection, "order by" and filter parameters should be supported. Inventory operations can reference any mapped concept, so the "order by" and filter parameters should also be supported for inventories. Requests can be encoded in XML. The advantage for implementation is that providers do not need to parse the XML Schema of response structures.

  The next example shows a capabilities response with a set of protocol features that would typically be supported by TAPIR Intermediate providers.

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="http://rs.tdwg.org/tapir/1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                 http://rs.tdwg.org/tapir/1.0/tapir.xsd">
  <header>
     <source accesspoint="http://example.net/tapir.cgi"
             sendtime="2005-11-11T12:23:56.023+01:00">
       <software name="myTapirProvider" version="2.0"/>
    </source>
  </header>
  <capabilities>
    <operations>
      <ping/>
      <metadata/>
      <capabilities/>
      <inventory>
        <anyConcepts/>
      </inventory>
      <search>
        <outputModels>
          <knownOutputModels>
              <outputModel location="http://example.net/redlist/model.xml"
                           alias="redlist"/>
          </knownOutputModels>
        </outputModels>
      </search>
    </operations>
    <requests>
      <encoding>
        <kvp/>
        <xml/>
      </encoding>
      <globalParameters>
         <logOnly>denied</logOnly>
      </globalParameters>
      <filter>
          <encoding>
            <expression>
              <concept/>
              <literal/>
              <parameter/>
              <variable/>
              <arithmetic/>
            </expression>
            <booleanOperators>
              <logical>
                <not/>
                <and/>
                <or/>
              </logical>
              <comparative>
                  <equals caseSensitive="false"/>
                 <greaterThan/>
                 <greaterThanOrEquals/>
                 <lessThan/>
                 <lessThanOrEquals/>
                <in/>
                <isNull/>
                  <like caseSensitive="false"/>
              </comparative>
            </booleanOperators>
          </encoding>
      </filter>
    </requests>
    <concepts>
       <schema namespace="http://example.net/redlist/"
                 location="http://example.net/redlist_1_0.txt"
               alias="redlist">
          <mappedConcept id="http://example.net/redlist/DataSourceCode"
                         alias="DataProviderCode"/>
          <mappedConcept id="http://example.net/redlist/ScientificName"
                         alias="ScientificName"/>
          <mappedConcept id="http://example.net/redlist/RedListCategory"
                         alias="RedListCategory"/>
          <mappedConcept id="http://example.net/redlist/Country"
                         alias="Country"/>
          <mappedConcept id="http://example.net/redlist/AssessmentDate"
                         alias="AssessmentDate"/>
          <mappedConcept id="http://example.net/redlist/Assessor"
                         alias="Assessor"/>
      </schema>
    </concepts>
    <variables/>
    <settings>
        <maxElementRepetitions>100</maxElementRepetitions>
    </settings>
  </capabilities>
</response>
```

- **TAPIR Full:** Providers should be able to parse and respond to any query template with any output model based on known (mapped) concepts. Response structures can be parsed to the extent of known

XML Schema constructs. Providers must also support inventory on any concepts.

The next example shows a capabilities response with a set of protocol features that would typically be supported by TAPIR Full providers with some extra functionality such as accepting "log-only" requests and the "xslt" parameter:

```
<?xml  version="1.0"  encoding="UTF-8"?>
<response  xmlns="http://rs.tdwg.org/tapir/1.0"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://rs.tdwg.org/tapir/1.0
                                   http://rs.tdwg.org/tapir/1.0/tapir.xsd">
   <header>
      <source  accesspoint="http://example.net/tapir.cgi"
                sendtime="2005-11-11T12:23:56.023+01:00">
        <software  name="myTapirProvider"  version="1.0"/>
     </source>
   </header>
   <capabilities>
     <operations>
       <ping/>
       <metadata/>
       <capabilities/>
       <inventory>
          <anyConcepts/>
       </inventory>
       <search>
          <outputModels>
            <anyOutputModels>
              <responseStructure>
                </basicSchemaLanguage>
              </responseStructure>
            </anyOutputModels>
          </outputModels>
       </search>
     </operations>
     <requests>
       <encoding>
         <kvp/>
         <xml/>
       </encoding>
       <globalParameters>
          <logOnly>accepted</logOnly>
         <xslt/>
       </globalParemeters>
       <filter>
           <encoding>
             <expression>
               <concept/>
               <literal/>
               <parameter/>
               <variable/>
               <arithmetic/>
             </expression>
             <booleanOperators>
               <logical>
                 <not/>
                 <and/>
                 <or/>
               </logical>
               <comparative>
                   <equals  caseSensitive="false"/>
                  <greaterThan/>
                   <greaterThanOrEquals/>
                  <lessThan/>
                  <lessThanOrEquals/>
                 <in/>
                  <isNull/>
                   <like  caseSensitive="false"/>
               </comparative>
             </booleanOperators>
           </encoding>
       </filter>
     </requests>
     <concepts>
         <schema  namespace="http://example.net/redlist/"
                   location="http://example.net/redlist_1_0.txt"
                 alias="redlist">
           <mappedConcept  id="http://example.net/redlist/DataSourceCode"
                          alias="DataProviderCode"/>
           <mappedConcept  id="http://example.net/redlist/ScientificName"
                          alias="ScientificName"/>
           <mappedConcept  id="http://example.net/redlist/RedListCategory"
                          alias="RedListCategory"/>
           <mappedConcept  id="http://example.net/redlist/Country"
                          alias="Country"/>
           <mappedConcept  id="http://example.net/redlist/AssessmentDate"
                          alias="AssessmentDate"/>
           <mappedConcept  id="http://example.net/redlist/Assessor"
                          alias="Assessor"/>
       </schema>
     </concepts>
     <variables/>
     <settings>
         <maxElementRepetitions>100</maxElementRepetitions>
         <maxElementLevels>20</maxElementLevels>
     </settings>
   </capabilities>
</response>
```

All three levels of implementation can coexist within the same TAPIR network and still achieve interoperability with a common set of features supported by all participants. Therefore one of the requirements for setting up a TAPIR network is to define the minimum capabilities level for all providers.

For example, if a TAPIR network wants to allow TAPIR Lite providers to participate, then the minimum requirements must include the set of query templates and conceptual schemas that need to be supported. The easiest way to do this is to create the conceptual schema, response structures, output models, query templates and then produce a CNS configuration file for all of them. Participants of the network can be instructed to support all definitions in the CNS configuration file, including the specific aliases if necessary. The next example shows such a CNS configuration file.

```
[concept_source]

label        = Red List Schema v1.0
namespace  =  http://example.net/redlist/
alias        = redlist
location     =  http://example.net/redlist_1_0.txt

[aliases]

DataProviderCode  =  http://example.net/redlist/DataSourceCode
ScientificName      =  http://example.net/redlist/ScientificName
RedListCategory    =  http://example.net/redlist/RedListCategory
Country               =  http://example.net/redlist/Country
AssessmentDate      =  http://example.net/redlist/AssessmentDate
Assessor              =  http://example.net/redlist/Assessor

[models]

redlist  =  http://example.net/redlist/model.xml

[templates]

basicsearch      =  http://example.net/redlist/basicsearch.xml
nameinventory  =  http://example.net/redlist/nameinventory.xml
```

In this example, TAPIR Lite providers will only use the concept source and the templates. TAPIR Intermediate providers will also need the models. TAPIR Full providers will only need the concept source, but in supporting any output models and query templates, TAPIR Full providers may well be forced to declare the templates in capabilities responses.

If a TAPIR network wants more complex functionality, such as having clients that dynamically build response structures based on user's input, then TAPIR Lite providers will not be able to participate. The minimum requirements of such network will be high, as providers will need to know how to parse XML Schemas.

More complex networks could require providers to support XML Schema parsing beyond the "basic schema language". Another example of a more advanced network could be one that providers need to support custom filter operators.

It would be useful for TAPIR networks to have an automated way to periodically check that all participants support the necessary features.

### 2.7. Using a central cache

When a TAPIR network is designed to work with a central cache, there is no standard database structure to store harvested data as TAPIR can work with completely customized conceptual schemas and output models. In this situation, the database structure should be defined according to the conceptual schemas involved and to the specific "data usage" expected from the network. For caching, one can also consider creating specific output models that contain only the subset of information that is stored in the central database. This increases performance of the network and makes the caching more transparent to providers.

It should also be possible to set up a single TAPIR service over a cached database in case the network wants to make use of existing TAPIR client software to interact with the central database or perhaps to offer more complex TAPIR features on top of the central database.

## 3. Conclusions

The greatest advantage of TAPIR-based networks is that there are no special requirements for the structures of the underlying databases. TAPIR networks do not require that data providers use the same data models, as long as their data can be mapped to an abstraction layer defined by means of one or more conceptual schemas.

TAPIR is flexible enough to allow different levels of implementation, so that networks can choose a minimum level of functionality required by all participants.

Successfully setting up a TAPIR network involves:

- Defining an architecture with all necessary components.

- Following the specific guidelines related to TAPIR and presented in this document.
- Developing new software or using existing applications for each task.

After following these steps, TAPIR networks are encouraged to produce a brief document containing all technical requirements for data providers to participate.

## 4. Appendix

### 4.1. Term Definitions

| | |
|---|---|
| ABCD | Access to Biological Collection Data. See http://www.bgbm.org/TDWG/CODATA/Schema/default.htm |
| Access Point | The URL (Web address) of a Web Service. |
| BioCASe | The BioCASE Access Protocol. See http://www.biocase.org/products/protocols/index.shtml |
| BioCASE | Biological Collection Access Services. See http://www.biocase.org |
| CNS | Concept Name Server. A service to get information about existing conceptual schemas and their concepts. |
| CONABIO | Comisión nacional para el conocimiento y uso de la biodiversidad. See http://www.conabio.gob.mx/ |
| Concept | Definition of a property, class or relationship. |
| Conceptual schema | A formal definition of concepts. It can also be seen as a data model or ontology. |
| CORBA | Common Object Request Broker Architecture. See http://www.corba.org/ |
| DarwinCore | Data format for exchanging information about geographic occurrence of organisms and the physical existence of biotic specimens in collections. See http://www.tdwg.org/activities/darwincore/ |
| Data source | The term used in the BioCASE project for an Access Point |
| Dublin Core | Dublin Core Metadata Initiative. See http://dublincore.org |
| DiGIR | Distributed Generic Information Retrieval. See http://www.digir.net |
| Federation schema | A Conceptual Schema adopted by a federated network |
| FishNet2 | Distributed information network linking museum databases of fish collections.. See http://www.fishnet2.net/ |
| GBIF | Global Biodiversity Information facility. See http://www.gbif.org |
| GET | HTTP communication method where form data is encoded as parameters in an extension to a URL. The GET method is principally used to transmit requests for data to a Web server e.g. a simple database search |
| Habitas | The web site of the Ulster Museum Sciences Division. See http://www.habitas.org.uk |
| HerpNET | Global network of herpetological collections data. See http://www.herpnet.org/ |
| HTML | Hypertext Markup Language. a subset of Standard Generalised Markup Language (SGML), used for authoring pages for the World Wide Web. |
| HTTP | Hypertext Transfer protocol, the commonly used protocol for transmitting requests and documents between applications on the World Wide Web |
| KVP | Key-Value Pair. One of the possible encodings for TAPIR requests. |
| OMG | Object Management Group. See http://www.omg.org/ |
| MaNIS | Mammal Networked Information System. See http://manisnet.org/ |
| NBN | United Kingdom's National Biodiversity Network. See http://www.nbn.org.uk/ |
| normative | Referring to a standard or set of norms that are understood to be correct. A normative document is one which describes how things should be and why they are like that. |
| OBIS | Ocean Biogeographic Information System. See http://www.iobis.org/ |
| OWL | Web Ontology language. See http://www.w3.org/2004/OWL/ |
| POST | POST is an HTTP communication method that can include any kind of data or command. The data are encoded separately and do not form part of the URL as in a GET message so this method is better for complex, sensitive, lengthy or non-ascii data. |

| | |
|---|---|
| protocol | An agreed format for transmitting data between two or more devices |
| Provider | Originally defined as an organisation hosting either a DiGIR or a BioCASe service but in the TAPIR context, an organisation hosting a TAPIR access point, which may point to several data sources. |
| Provider software | Software running on a providers Web server that includes a database wrapper or a query portal. |
| RDF | Resource Description Framework. See http://www.w3.org/RDF/ |
| RDF Schema | Language for declaring basic class and types for describing the terms used in RDF. See http://www.w3.org/TR/rdf-schema/ |
| SOAP | Simple Object Access Protocol, an XML-based messaging protocol used for invoking Web services and exchanging structured data. |
| speciesLink | Brazilian distributed information network of specimen and observation data. See http://splink.cria.org.br |
| TAPIR | TDWG Access Protocol for Information Retrieval. |
| TDWG | Taxonomic Databases Working Group |
| Triple store | A system for storing and managing RDF data. |
| UDDI | Universal Description, Discovery and Integration. UDDI is a specification for maintaining standardised directories of information about Web services. |
| URL | Uniform Resource Locator. The address of a resource on the Internet |
| URI | Uniform Resource Identifier. A formatted string that serves as an identifier for a resource, typically but not exclusively, on the Internet. URIs are used in HTML hyperlinks. |
| W3C | World Wide Web consortium. See http://www.w3c.org |
| Web Service | A service based on Internet Protocols, such as HTTP, SMTP or FTP, and also based on XML. |
| wrapper | Software that allows standardised queries to be run against an underlying database of arbitrary structure. |
| WSDL | Web Services Description Language. An XML format for describing Web Services as a set of end points operating on messages containing either document-oriented or procedure-oriented information. WSDL is the language used by UDDI. |
| XMI | XML Metadata Interchange (XMI) is an OMG standard for exchanging metadata information via XML. See http://www.omg.org/technology/documents/formal/xmi.htm |
| XML | Extensible Markup Language developed by the W3C. a means of tagging data for transmission, validation and manipulation. http://www.w3.org/XML & http://www.w3.org/TR/REC-xml |
| XML Schema | A formal definition of the required and optional structure and content of XML formatted documents within its domain. See http://www.w3.org/XML/Schema |
| XPath | Defines a way of locating and processing items in XML documents by using an addressing syntax based on the path through the documents logical tree structure. See http://www3.org/TR/xpath |