



Technical Specifications for a European Question Data Bank

Final Version May 2009

Arofan Gregory / Pascal Heus / Chris Nelson
Metadata Technology

Jostein Ryssevik
Ideas2evidence



Abbreviations	4
Executive Summary	5
Background and Overview	6
The Question Bank as a Distributed Application	6
Existing CESSDA Metadata Resources	7
The Shared Metadata Model	7
Maintenance and Ownership	7
Technology Considerations	8
Integration and Deployment Strategies	9
Summary	10
Question Banks – Purpose and Functionality	11
Content and Structure	11
Functionality	12
Architecture	13
Overview	13
Metadata Model	13
Repository	14
Registry	17
Question Data Bank (QDB)	18
3CDB	19
New Content from QDB and 3CDB	19
QDB Specific Notes	20
System Administration and Security	21
Other Services	21
Summary	21
Registry	23
Scope	23
Conceptual Model	23
Using the Registry	25
Registry Interfaces	25
Core Services	31
Question Bank Conceptual Model	33
Scope	33
Location of Objects	33
Question Scheme – Core Model	35
Question Scheme – Embedded Constructs	36
Question Scheme – Associated Constructs	39
Requirements and Use Cases	49
CESSDA Requirements	49
CESSDA Tender Use Cases	51
Other CESSDA Use Cases	53
Metadata Specifications	67
Social science / Statistics	67
Technology	68
Technologies	71
Development	71
Server side	73
Web Front-End	76
Databases	77
Other Products	79
Implementation Issues	81
Metadata	81
Repositories	82
Registry	82
QDB	83
3CDB	83
Infrastructure	84
Conclusions	85



Abbreviations

3CDB	Concept Classification Conversions Data Bank
CESSDA	Council of European Social Science Data Archives
DDI.....	Data Documentation Initiative
HASSET.....	Humanities and Social Science Electronic Thesaurus
QDB	Question Data Bank
SDMX.....	Statistical Data and Metadata Exchange Standard
SOA	Service-oriented architecture
XML.....	eXtensible Markup Language



Executive Summary

This document describes the architecture for the European Question Bank, to be shared by all of the CESSDA data archives to provide information about questionnaires and questions to their users, both for re-use and in support of research. This question bank is made up of resources residing in several locations, in the databases of the many different archives. As such, it represents a coordinated system for navigating and accessing these resources from a single location on the network, but without centralized storage of the questions, questionnaires, and other metadata.

This type of application, while not simple to create, is well-supported by modern “web services” technologies. For this reason, this architecture is very much a service-oriented architecture – one which leverages the current technologies for distributed computing.

One aspect of the question bank is the requirement it has for access to other metadata resources. A question bank is not useful in isolation – it relies on knowledge of the concepts, classifications/code lists, variables and other information about the datasets which the questionnaires were used to collect. For this reason, a question bank should not be designed in isolation, but should form one major part of a larger architecture, which could also be used to support other shared applications such as the 3CDB.

This document addresses both this overall architectural framework, and the specific functionality of the question bank. The first section introduces some of the technology themes which are significant in understanding why this architectural approach – web services – was used. The next section describes the functionality and purpose of a question bank. Following that, the overall architectural framework is described, with further detail on the question bank architecture. The architecture is explained in detail with models of the various functions, resources, and components.

The business use of the question bank is then examined, with a discussion of various use cases. These are tied back to the detailed architecture models provided in the earlier technical sections. There is also a discussion of the various metadata models which can be used, and a discussion of the technology selection. Finally, implementation strategies are covered – the deployment of this type of an application involves many different organizations, and must be carefully planned in order to succeed. This section addresses these issues.



Background and Overview

The CESSDA Question Bank is a part of a larger envisioned system which will allow sharing and reuse of many of the resources held by the CESSDA archives. This document describes both the specific functionality of a shared CESSDA question bank, and the position this application occupies in relation to other components that may be deployed as part of the overall networked system. To provide a solid architectural foundation for the question bank, some specific technology choices have been made. To understand these, it is important to be familiar with the technology landscape within which the question bank will exist.

The Question Bank as a Distributed Application

The advent of the Internet has given rise to a strong interest in “distributed computing” – the sharing of applications and resources around the nodes on a network. This technology approach is very common today within enterprises – the existence of computer networks leads naturally to the idea that applications and data may themselves exist outside the local computer environment, but be available for local use.

The implementation of this concept presents many challenges, however – coordination of applications and data resources on a single machine is straightforward, but when the many different nodes on a network need to operate in a coordinated fashion, there are many potential pitfalls. On a single machine, there is one platform on which an application must operate. Even within an organization, there can be policies which guarantee a small set of platforms and technology tools are used, making enterprise-wide applications, even if distributed, relatively easy to manage.

When an application is to be shared by many different organizations, however, there is no commonality of platform or technology environment. Each organization builds its systems to its own requirements, and these will naturally be different across organizations. The community of CESSDA archives reflects this type of diversity.

Applications like the CESSDA question bank are fundamentally based on the idea of distributed resources – each of the CESSDA archives has resources – questionnaires, variables, classifications, concepts, etc. – which are of interest to other archives. And although traditional approaches to such applications involved the collection and centralization of these resources into a single large database, there are many problems with this solution. First, because of the size of the resulting database, there may be problems of performance and scale. Second, there are issues of maintenance, ownership, and consistency. These result from the fact that the owner of a particular resource may make changes or corrections which are subsequently not reflected in the copy of the resource stored centrally. Historically, it has been shown that this centralized approach can be made to work, but that it is difficult to govern, maintain, and manage the centralized database, especially when it is shared by an entire community.

These problems have been solved in many modern situations using a distributed approach, where the maintenance, scalability, and integrity problems are handled by the new “web services”¹ standards, architecture, and technologies. The CESSDA question bank is a very typical example of a distributed application, because an entire community of archives is using a single system. As such, it is an excellent candidate for the use of a service-oriented architecture.

¹ http://en.wikipedia.org/wiki/Web_service



Existing CESSDA Metadata Resources

CESSDA has many advantages in approaching the creation of a shared question bank. The use of Nesstar – being based on DDI – provides a common approach to the modeling of metadata, and serves to limit the number of approaches used to describing data, including the questions used to collect it. Further, it provides for a shared view of the variable, which is an important point when querying a question bank for questions that might be of interest to a researcher or the designer of a survey, as the question – variable relationship is a primary way of constructing such queries. DDI also provides a common approach to the relationships between concepts, variables, and questions, and these relationships are also important when navigating the contents of a question bank.

CESSDA has another major asset, in the form of the HASSET² classification. Shared applications require that the users have agreed on the fundamental organization of resources, and HASSET provides a very useful classification for the description of metadata relating to social sciences data, questions among them.

Thus, the community of CESSDA archives is reasonably well-equipped to approach the creation of a question bank relative to some other domains deploying distributed applications, from the perspective of its metadata modeling and resources.

The Shared Metadata Model

Every application has an underlying information model, in which the relationships between various types of data and metadata are described. One challenge for the creation of a CESSDA question bank is the creation of a single, agreed metadata model against which the similar (as a result of DDI) but not-identical modeling found in the many databases and other applications which store questions and other needed metadata in the many CESSDA archives.

When a metadata model is created for a single database or other local application, the relatively narrow requirements of the organization are reflected in the application's design. When a metadata model is created to act as the basis of a shared application, then a very broad set of requirements is faced, and it can require a huge amount of effort to make sure that every needed variation in each of the many systems participating in the application is covered.

Typically, the best approach to this problem is to come up with an agreed metadata model against which the local databases and applications of participating organizations can be mapped. This is the recommended approach for the creation of the CESSDA question bank, and it is fortunate that a standard metadata model already exists which will meet many of the question bank's requirements – DDI.

Maintenance and Ownership

One of the problems that must be addressed in the creation of applications which serve an entire community is the handling of maintenance and ownership issues around the shared content. As mentioned above, if a single centralized database is created, it becomes difficult to keep the contents of that database in synch with the many systems from which its contents are drawn. Further, some agency must exist which houses and operates the centralized database, and this agency's approach to managing the shared resources often does not match that of the source organizations providing the content.

There are several requirements which are not always easy to meet:

² <http://www.data-archive.ac.uk/search/hassetAbout.asp>



- The contents must be maintained, including corrections, changes, etc. which typically occur in the source system and have to be replicated in the shared application or otherwise kept in synch
- The ownership of each metadata resource must be clear, so that questions around its use and provenance can be answered
- Attribution may be required in cases where intellectual metadata resources such as questions are re-used

The best strategy for handling these problems is one that is fundamental to good database design – store each bit of information once, and use it by reference elsewhere. Although the question bank is not a simple database, but instead a highly distributed one, this principle can still be observed. There are fewer problems of maintenance and ownership if the resources to be shared live only within the institution which provides them to the community. That organization is best suited to maintain and manage the resources it owns. Other institutions should get the questions and other metadata from the best source – that is, from the owning institutions.

This approach comes with its own requirements, however – in a scenario where the centralized application references questions housed in the source institutions, there needs to be a clear way to describe the ownership and provenance of the resources, and an agreed system for identifying, packaging, and accessing them. The CESSDA question bank as described here utilizes some standard and proven mechanisms for meeting these requirements, based on work which has occurred both within the data archive domain, and in the broader technology world.

Technology Considerations

The CESSDA question bank benefits from technology developed over the past decade, which has solved many of the difficult problems of distributed computing across the Internet. There is a set of standards, technology tools, and approaches referred to as “web services” or “service-oriented”³, and these tools and approaches are well-suited for the development of the CESSDA question bank. Several types of web-services exist, and are useful in creating this application.

The key to web-services technology is the interaction of the various components which provide content and functionality to the overall application. These components provide access using a standard approach based on the exchange of XML documents. Because we have DDI, in both versions, we have the basis on which to build a standard set of messages for packaging and exchanging data. DDI, however, is not in itself sufficient to support the web-services interactions which will be needed, and the XML used by the CESSDA question bank will use an extended set of XML documents.

One advantage to web services architectures is the solution they provide to issues of platform compatibility – web services are designed to support very flexible, “loosely coupled” interactions, and these meet the needs of the diverse technology platforms found within the CESSDA archive community very well. This will make the integration of the many archives’ systems into the shared system much easier, and support the use of the many web-services-based integration packages which are found in many modern database and development tools.

Another web-services artifact which will be very helpful in creating the CESSDA question bank is the use of a registry/repository. This is a type of technology designed to support the

³ http://en.wikipedia.org/wiki/Service-oriented_architecture



location of all of a distributed applications components around a network. In the case of the CESSDA question bank, all of the various metadata resources can be indexed in a central registry, but stored on the sites of the organizations which own and maintain them. They can thus be retrieved from their source, but still have a centralized place where it is possibly to query and search for whatever particular resource is needed.

An additional useful aspect of web-services technology that will be deployed is a subscription/notification mechanism. In order to guarantee referential integrity of the registry around the network, and to support high availability of the system, it is possible to put in place a mechanism for broadcasting the occurrence of significant events such as the publication of new questions or other metadata to any applications which care about that type of resource. This broadcast allows for applications which use that particular type of resource to immediately harvest it from its source.

Web-services are extremely well-suited to creating the type of application represented by the CESSDA question bank, and this approach has been used in designing an appropriate architecture for it.

Integration and Deployment Strategies

There are a number of practical issues in the integration of archives' resources into the network, and in how a complex set of applications – of which the CESSDA question bank is only one – can be effectively deployed. A good architecture can help in addressing these issues.

One problem is the integration of many different types of applications and databases into a single integrated system. As mentioned above, some web-services tools are well-suited to this task, and there is a typical approach to their use. This involves providing a standard “gateway” software package for interacting with the CESSDA question bank, while providing a set of tools for integration with the many different back-end systems found in the archives. Some approaches to the integration can be based on the use of different versions of DDI to capture metadata today within the archives, as this allows for re-use of a gateway software package among those archives which already have metadata in this standard format. Other integrations will need to be performed against the specific database structures used within an archive.

Good architecture dictates that processing be performed in the nodes of a distributed application as much as possible, so as to not overwhelm the centralized computers which connect the nodes together. Thus, we propose the creation of a gateway software package to be used as the basis for integration with each of the CESSDA archives.

Another challenge which will be faced in deploying the entire suited of networked applications envisioned by CESSDA is how to bring them online in a phased way, even though it will take time to integrate with all the different archives and to integrate the question bank with such related metadata-based applications as the CCCDB. The design proposed has a solution to this problem which has been used in similar applications in the past, and which solves a number of problems simultaneously. The subscription/notification mechanism in web services is used to create self-updating caches, sitting within each application, so that they have a local copy of all of the needed resources, but one which cannot get out of synch with the original source. If such a cache relies on a source which has not yet been integrated into the system, then the local cache can be a temporary repository, so that the needed metadata resources are available while the integration with their source is performed. Thus, a phased deployment is enabled.



A related aspect of the design is the possibility of hosting metadata for some institutions either temporarily or permanently, as needed, in a dedicated repository. This is not an optimal choice, but it is something which may be practically necessary for some institutions, until they come up with the resources to integrate into the system.

Summary

Overall, it can be seen that many of the needed tools, resources, and approaches to create a CESSDA question bank are readily available, so long as the application is understood to be an example of a typical distributed application. Some resources come from within the social sciences data archive world – others are taken from the technology toolset which is used to address distributed applications in many different domains.

There is no question that the CESSDA question bank is well-suited to the use of web-services technologies, and that this set of technologies is both proven and mature. This is clearly the best approach to the architecture of an application with difficult requirements, but ones that have known technology solutions today.



Question Banks – Purpose and Functionality

There is no strict definition or consensus of what a Question Bank is, what type of information it should include or what type of functionality it should provide. However, as a minimum requirement you would expect a Question Bank to support storage and retrieval of questions or items from a single survey program or from a wider collection of surveys.

Content and Structure

Question Banks will often contain a wider set of metadata than the questions, but will normally not include the complete metadata record associated with a sample survey. Nor would you expect a Question Bank to hold or provide access to the physical data collected by the survey instruments. However, Question Banks will often provide links to further sets of metadata as well as physical data held in parallel or neighboring repositories. Question Banks are therefore quite often part of a wider information infrastructure supporting the production and use of survey data.

This is the approach taken by CESSDA where the CESSDA QDB will live in an environment of a series of more general purpose metadata repositories as well as a dedicated database for concepts, classifications and information related to comparison and harmonization (the 3CDB). For an approach like this to work, a well defined model of the entire information space is required along with standardized interfaces between the various components of the system.

The main object of a Question Bank is the question and the related response categories (normally also including any interviewer instructions associated with the question). This is distinguishing a Question Bank from Concept Banks (focusing on generic concepts or terms), Questionnaire Banks (focusing on complete survey instruments) as well as Data Banks (focusing on variables in physical datasets).

From a logical point of view questions are located somewhere between concepts and questionnaires on the relevant ladder of abstraction. A concrete question represents a concept in such a way that it can be measured in the real world. The relationship between concepts and questions are however complex. A single concept might be measured by more than one question (and quite often batteries of questions are required to tap the various aspects of more abstract constructs in the behavioral sciences). A single question can also be used to measure more than one concept, at least related concepts belonging to disparate concept spaces (e.g. the concept banks of two different organizations).

Questions will always be used in the context of a survey questionnaire, but will in a well modeled Question Bank be treated as a more generic object that can live independently of this context. A single question will normally be used in more than one questionnaire and quite often also be used across survey programs and languages. This is typically the case for standard demographic questions or established batteries of questions used to measure values, attitudes, skills or behaviors.

Questions are for this reason also “metadata before data”. They are used as instruments to produce physical data, but should in a well modeled system live independently of these data. Firstly, a single question will often be used across several data collecting processes. Secondly, there is no one-to-one relationship between questions and physical variables even within a single data collection process. A single question might as the result of data transformations end up as more than one physical variable, or alternatively, a group of variables might be needed to construct a single additive or composite index.



A well functioning Question Bank should consequently not only include the basic object, the questions, but also rich references to the other main objects of the survey-based information system: the concepts, the questionnaires and the dataset/variables. All of the complex relationships described above must consequently be modeled in to support the types of metadata and data mining functionality required by CESSDA. Whether these additional objects are stored / cached in the Question bank or just included by reference, is an implementation issue. These issues are further elaborated and detailed in the Conceptual Model described in this document (see Question Bank Conceptual Model, p.33).

Functionality

Also when it comes to functionality there is no general consensus of what a Question Bank should support. However, several different areas of use might be envisaged:

- **Functionality supporting data producers in their efforts to design new surveys:** With a well designed Question Bank at hand a data producer (an individual researcher as well as a regular data producing agency), will be able to search for questions used by other researchers and agencies to measure the concepts they are interested in. This could be done in order to locate well-proven and quality assured measurement instruments, or in order to find questions that when used in a survey will produce data that are comparable to already existing data. For this to work well the ability to locate questions through concepts, as well as the ability to link questions to additional survey metadata and maybe even physical data will be required.
- **Functionality supporting users of data in their efforts to assess/understand a dataset:** When analyzing a dataset, a researcher will often need access to the question text as well as the wider context of the question within the questionnaire (neighboring questions, questionnaire flow etc.). For a Question Bank to support this functionality, references to questions from variables in the dataset are required, along with links from the question to the questionnaire where it is used.
- **Functionality supporting data users looking for comparable data:** A well defined Question Bank might serve as an efficient entry point for researchers looking for comparable data. If the references between physical variables and questions are semantically rich and well designed, a Question Bank might help answer queries like: a) give me all datasets/variables where question A has been used, or b) give me all other datasets/variables using the same question as variable X, or c) give me all datasets including the two questions I need to construct a comparable crosstab to the one I am looking at from dataset Y.
- **Functionality supporting comparison and post hoc data harmonization:** In order to locate data which are potentially comparable (not comparable by design) and to perform post hoc harmonization of these data, a Question Bank will provide valuable information. Potentially comparable questions can be identified through the common concepts that they are measuring and further links from the questions to the data will help the researcher locate the relevant data sources. Once located, the question Bank will provide most of the information needed to assess whether post hoc harmonization is feasible or not, and if yes, how it best can be achieved.

These and other functional requirements are further elaborated in the detailed use-case descriptions of this document (see Requirements and Use Cases, p.49).



Architecture

Overview

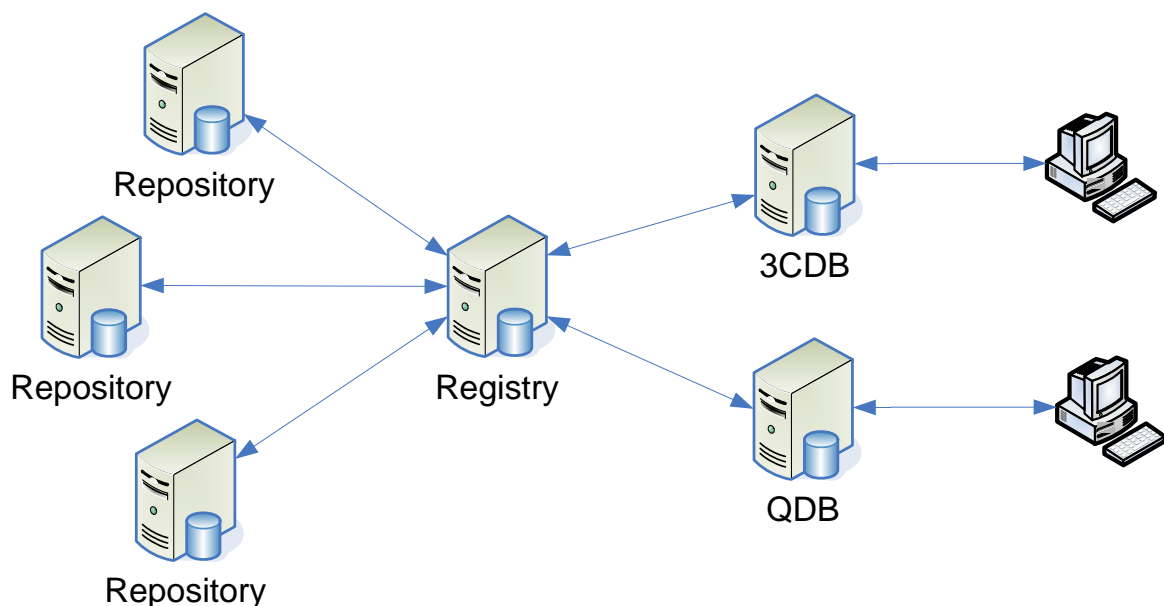
The proposed architecture expands beyond the scope of the QBD and addresses broader issues related to the federated CESSDA environment, the technology diversity, variations in metadata structures and ownership. It was necessary to solve some of these in order to design a solution for the QBD.

The following constraints also drove the design:

- the QBD focuses on questions and will only store the information it needs. It will rely on other sources (such as the 3CDB) to provide the metadata elements
- the 3DCB itself will likely start as an empty database that will be populated over time by its users

This essentially means that both databases will rely from the beginning on external metadata sources and there is a need for mechanisms to efficiently store, reference, search, and retrieve such information.

Based on the previous section argumentations, we propose for this backend metadata storage to be driven by a web services based architecture composed of a registry and multiple repositories. This provides support for future CESSDA applications as well and allows for metadata currently not directly needed by the 3CDB and QBD to be stored in the system. The exchange of information between the different systems will take place using web services and based on a common XML metadata model.



The repositories shown here can be shared by multiple archives or hosted by individual agencies. Their implementation is further discussed below.

Metadata Model

The metadata coming out of the various archives and stored in the repositories may vary considerably based on the provider:

- some may be using Nesstar to document their surveys, therefore having the ability to deliver survey, file and variable, question, classification level information based on the DDI 1 specification



- some may be using different DDI tools or legacy databases to capture their metadata and may therefore provide different DDI elements or a subset of the survey metadata
- some may not be “codebook” centric and may manage metadata structures such as universes, questionnaires, or concept banks
- Down the road, the 3CDB, QDB, and possibly other systems, will also in essence become metadata providers

The metadata model used across the system must therefore be sufficiently flexible to account for this diversity.

The metadata in the QDB and related metadata sources can actually be broken down into smaller independent components such as: concepts, universes, surveys, files, summary statistics, variables, classifications, questions, questionnaire flows, instructions, tables, and geographies. A metadata repository should therefore be able to store any one or all of these components and make them available to the outside world through standard interfaces.

In order to capture this using XML, we need to create a model that can independently represent these objects with the ability to maintain their referential integrity and ensure their unique identification. The above list is remarkably, but not surprisingly, very close to the model of the latest version 3 of the DDI specification. This should therefore be used as a base for the CESSDA model which would then at the same time ensure compliance with the DDI (an optional requirement of the QDB).

Describing a complete model for CESSDA is a task beyond the scope of this document that primarily focuses on the QDB. The DDI 3 model however provides a good basis for this. A partial model that supports the QDB functionalities is provided in the “Question Bank Conceptual Model” section (p.33)

Repository

The repositories that hold the metadata essentially provides a storage and retrieval structure for the conceptual model (full or subset). Such database can be implemented using open source or commercial products (file system, relational databases, or native XML databases). Interacting with the database will occur through web services that abstract the underlying proprietary storage and retrieval system or the platform. This also means that the repository interfaces could be implemented on top of an existing system, as long as the model metadata consistency and integrity can be assured. These implementation options are further discussed below (p.15).

Note that in addition to the core metadata, a repository must also provide storage space and services for administrative features such as usage statistics, system logs and security. These will need to be further defined based on CESSDA’s requirements.

Object Banks

A metadata repository consists in a collection of object “Banks”, with each bank storing a particular type of object. Based on the current breakdown, we define the following Banks:

- Study Bank: hold general information about the surveys.
- Variable Bank: describes the data file variables
- File Bank: describes the dataset files structure, relationships, content and summary statistics
- Concept Bank: collection of concepts used by variables, questions, and other elements
- Universe Bank:: contains universe documented in the surveys
- Question Bank: contains questions
- Questionnaire Bank: bring together questions in to control flow and instruments
- Classification Bank: code and categories used by variables or questions
- Geography Bank: shapes and related hierarchies



- Table Bank: collection of aggregated cubes or tabulated data structures
- Organization Bank: address book of agencies and individuals involved in the survey various processes
- External Resources Bank: list of other materials reference by metadata elements

Banks expose a set of common and specialized interfaces and will also have the ability to group their objects for organizational, comparison or mapping purposes (see p.16).

Object stored in a bank will be uniquely identified using a URN mechanisms based on the combination of maintenance agency, version and identifier (similar as the approach used by SDMX and DDI 3). This also means that each object becomes reusable and can be referenced for various purposes such as comparison or harmonization.

Repository Implementation Options

There are essentially two ways a repository can be implemented: as a stand alone generic product or on top of a legacy infrastructure. The first approach assumes that the metadata will be packaged and published into the registry. The second essentially hides the legacy system from the rest of the world by exposing the set of interfaces required by a repository as web services. Whichever is used makes little difference to consuming applications.

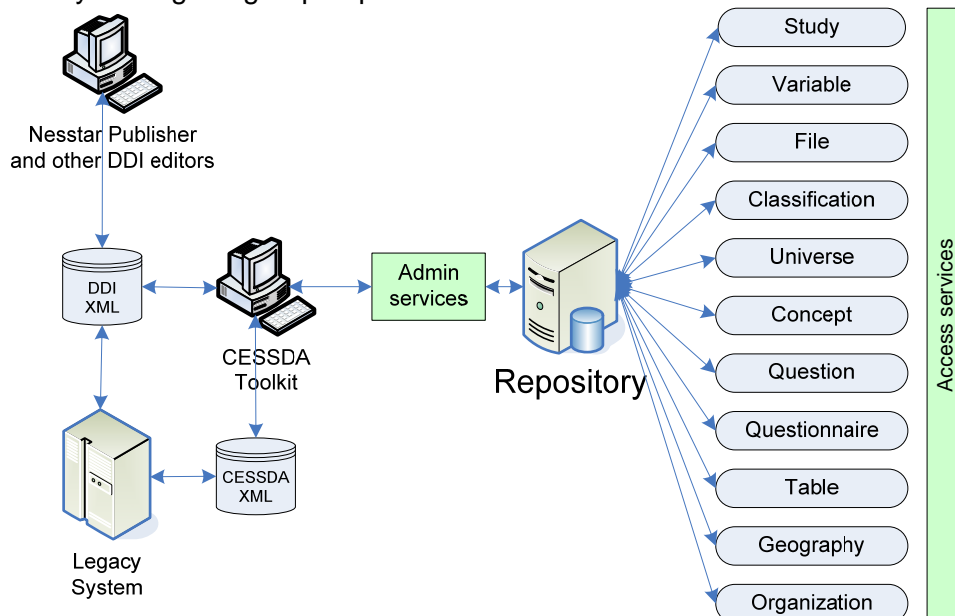
Generic Repository

A stand alone generic repository is typically required for:

- Archives that do not have metadata management systems
- Archive with limited IT capacity
- Archives whose legacy system cannot meet the metadata model requirements
- New applications such as 3CDB and QDB

The advantage of this approach is that it only needs to be developed and maintained once for everyone. It does requires however require packaging and publication processes that can be seen as a drawback by some. Since the new 3CDB and QDB database will require local metadata repositories, we see this as a required step.

Such repository will typically be hosted by the metadata provider who therefore retains full ownership of its information. In some case however, it might be desirable to share such facility amongst a group of providers.





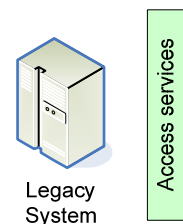
The packaging and publication tools can be made available to metadata providers as a CESSDA Toolkit that:

- Facilitates the import of metadata from existing systems
- Interact with the repository services for the management of metadata
- Provide functionalities to optimize the metadata in order to take advantage of the model and maximize reusability. This can be seen as a first harmonization stage at the metadata provider level.
- Ensure metadata integrity
- Provide various quality assurance functionalities
- Register metadata in the CESSDA registry (see below)

Note that most of these operations, depending on the consistency and completeness of the source metadata, can be automated or semi-automated.

Legacy Repository

For agencies that have a strong IT capacity and a solid metadata management system in place, implementing the set of required repository interfaces on top of an their existing platform might be an attractive solution. An important requirement however is that the existing metadata must be able to meet the requirements of the CESSDA XML conceptual model. The main advantage of this approach is that it eliminates the packaging and publication processes and minimizes impact on the rest of the system. It does however imply considerable custom development and maintenance. Note that building this around a Nesstar server could be an option to consider. However, the DDI metadata structure and content currently in use are likely insufficient to meet the repository requirements.



Bank General Functionalities

All banks in a repository will be expected to be able to perform a common set of operations. The most important ones are defined below. These functions will be exposed as access or administrative interfaces through web services.

Retrieve

- Retrieves partial or full object
- This should expose multiple standard views of an object such as: full (all the metadata), summary (common subset of metadata), index (subset of metadata subject to registration), id (only identification elements), etc.
- This operation could also offer the option to resolve external reference to return a stand alone instance

Create/Update

- Adds or updates an object in the bank
- New objects are by default considered unpublished

Delete

- Removes or archives an object in the bank
- Depends on the state of the object
- Objects that have been published cannot be deleted.
- Delete can cascade to other objects

Deprecate

- Provides the ability to “retire” and object from the system
- Deprecated objects continue to exist in the system but should normally not gain new references (unless necessary)
- A deprecated object should point to another object that replaces it



- For example, a classification used in a survey may be considered deprecated when a new harmonized version becomes available. Future surveys / questionnaires should use the new version.

Version:

- Creates a new (unpublished) version of an object
- Can be major or minor

Grouping

- Create and maintain groups to bring together objects in the bank in various flat and hierarchical structures

Comparability (when applicable)

- Brings together two objects in the bank for the purpose of comparing them or creating mappings
- This is particularly important for the 3CDB and QDB

Bank-Specific Functionalities

In addition to the standard operations, a set of custom interfaces may be developed, depending on the specific type of objects used by the bank. These objects are those which typically will not be shared within the network, other than as exposed through these dedicated interfaces. The use of such objects tends to be restricted to applications which do not require access to other resources within the network, but only require connectivity to that specific bank.

Repository Level Functionalities

In addition to the bank interfaces, a repository will expose a set of higher level services to support features such as:

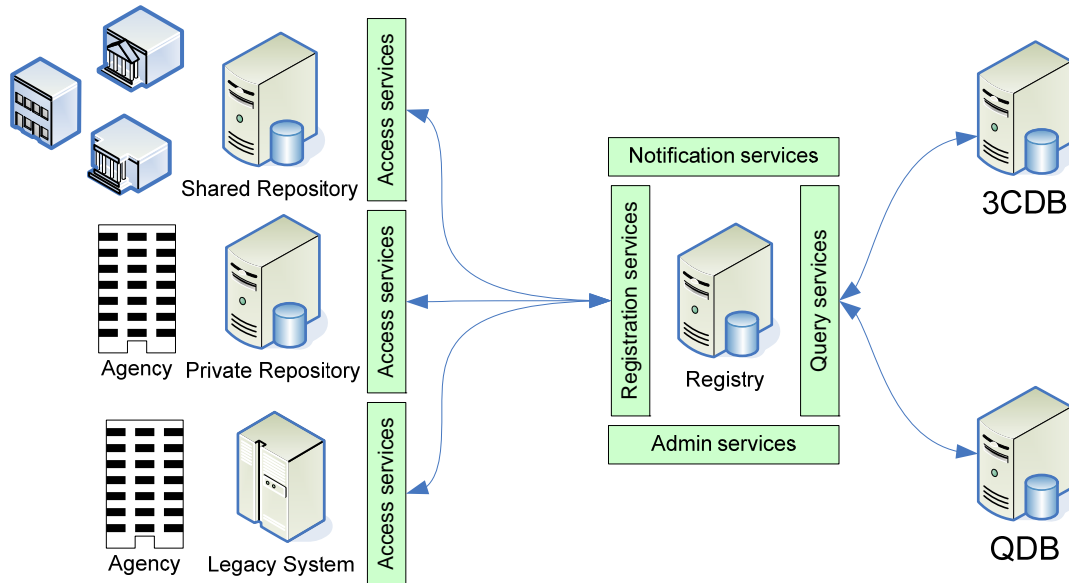
- Usage statistics
- Administration
- Security
- Basic search (less complex than a registry)

Registry

Once metadata stored in a repository is ready for access by external applications and users, it needs to be “registered”. This process consists in submitting the object to a registry that will then index it and expose it through its public interfaces.

The role of the registry is to find things around the network. It receives queries from applications - such as the 3CDB, QDB, or others - and performs lookups in its local database. It then returns the locations where objects can be retrieved (in our case the repository). This is typically a URL. A registry is essentially a specialized search engine for the CESSDA conceptual model. It fully understands the structure and complexities of the metadata in order to perform fast and efficient lookups.

One important aspect of the registry is that it only requires the metadata elements it needs to perform a search. The bulk of the metadata remains in the repositories. Things are expected to be stored in the registry include elements such as study title & year, variable labels and names, variable categories, question text, concepts, universes, and others. Large metadata collections such as, for example, summary statistics are excluded from indexing.



Status of Registered Objects

Once an object is registered, it means it becomes available to others for reuse and can therefore be referenced from anywhere on the network (for example the 3CDB). When this occurs, such objects can never be deleted as it would otherwise break down referential integrity. It also can no longer be modified as this could generate unpredictable results in other systems. Registered objects are therefore essentially read only and undeletable. This characteristic allows for the implementation of local caches to increase performances and reduce the network load. This is used for example in the QDB.

Note that the metadata model includes a simple mechanism to modify a published object: *versioning*. Whenever a registered object needs to be changed, a new *version* is created, updated as needed, published in a repository and in turn registered. This is actually one of the mechanisms that trigger notifications.

Interfaces

A registry requires various sets of interfaces in order to support registration of object in the index, searches, and subscription/notification services. This is described in details in the “Registry” section (p.23). In addition, like for repositories, a set of administrative interfaces will be needed to measure usage, monitor the service and security purposes.

Query Language

The 3CDB, QDB and future applications using the registry will also need a formalized mechanism to formulate a registry “query”. As the DDI specification lacks such feature, we would highly recommend taking an approach similar to the SDMX standard. This is further explored in the “Registry” section (p.23). Another standard (but less XML) that could be considered is Z39.50 (a client/server-based protocol for searching and retrieving information from remote databases.).

Question Data Bank (QDB)

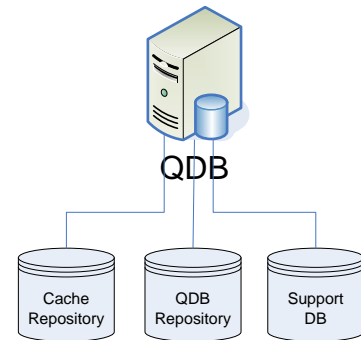
With the repository and registry in place, putting together the question bank becomes a lighter challenge.



Content

The content of the QDB can be broken down into the following components:

- New objects that will be produced by users when creating new questionnaires or capturing comparison information
- Referenced objects coming from the external sources (repositories and 3CDB) and can be cached locally to improve performances and reduce the network load
- Other data or metadata elements (not in the conceptual model) required for the management of the QDB (admin, logs, security, etc.)

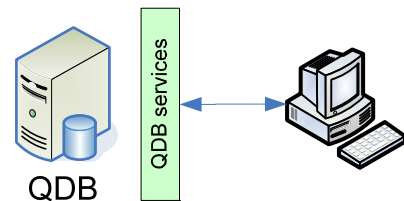


The first two databases can actually be implemented as generic repositories containing the relevant subset of metadata (questions, questionnaires, and cached objects like concepts, classifications and variables). The content of the support database will vary depending on identified administrative and security needs. It will also include user metadata elements (scripts, external resources, etc.).

Interfaces

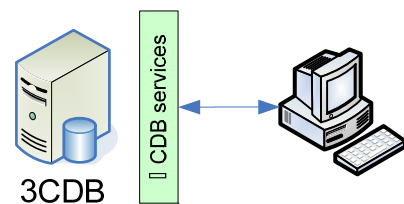
The QDB will expose a set of services to provide access to end user applications. These consist of

- Local Repository interfaces to support the creation of local objects and, when finalized, their registration in the registry
- A proxy of the registry interfaces so the end user application does not actually need to know who to contact at the main registry (though they can do so if desired)
- Local search interfaces for objects not stored in the registry
- Admin and security interfaces



3CDB

The 3CDB design is out of the scope of this document but we do recommend taking an approach similar to the QDB. To integrate into the proposed framework, the 3CDB will be expected to expose some of the interface to interact with the registry and possibly the QDB.



New Content from QDB and 3CDB

Both the QDB and the 3CDB will produce new harmonized metadata. Making these available to each other, the metadata providers and the CESSDA community will simply follow the same principles as presented for a metadata provider: both the 3CDB and QDB can essentially behave as a generic repository and, when a particular object is ready for reuse, it is simply registered (by the user or the administrator if clearance is required).



QDB Specific Notes

QDB vs Repository vs Registry

When it comes to questions and questionnaire structures, it is probably useful at this stage to emphasize the differences between various components their roles in the CESSDA environment:

- A Repository is where the original “legacy” questions appearing in the surveys will be stored by the CESSDA archives (along with all other archive metadata elements). When publishing metadata, the questionnaires flow, question texts and reference metadata elements will be stored in a local Repository Banks where they can be retrieved. Note that a “Repository Question Bank” in this context is primarily a storage and retrieval facility and it does not implement the more advanced functionalities of a QDB.
- The QDB is essentially a repository that focuses and specializes on "questions" and "questionnaires". It comes with enhanced functionalities that support the harmonization of questions or the design of new questionnaires such as:
 - o Provide services that are specific to the QDB related applications (relevant web services API, custom functions, specialized local/remote search features, etc.)
 - o Store the new harmonized objects (in progress or finalized form)
 - o Isolate the QDB applications from the back-end registry, repositories, 3CDB and possibly other metadata sources / services (proxy)
 - o Store additional information necessary to support the QDB applications (extended metadata, non-xml metadata)
 - o Implement a local cache for externally references objects to improve performances and reduce network and repositories traffic.
 - o Etc.
- The role of the Registry in relation to the question and questionnaires is simply to facilitate the search and retrieval operations of questions and related metadata. Any question, whether from an existing survey or a QDB can be registered. It then becomes discoverable by other CESSDA application and potentially reusable in questionnaires (and therefore new surveys)

Note that as for repositories, it is possible for more than one QDB to exist in the CESSDA space to allow for responsibilities to be distributed or local groups/communities to establish their own research domain.

Minimal Information for Questions and Questionnaires

The conceptual model presented later in this document (p.33) assumes the availability of a sufficient amount of information to describe questions and questionnaire. The current prototype is based on the DDI 3 model which in the case of question/questionnaires has the following object needs:

- A QuestionScheme to hold the set of questions
- A QuestionItem per question with at the minimum a response domain
- If capturing the interviewer instructions
 - o An InterviewerInstructionScheme to store the instruction(s)
 - o A ControlStructureScheme to associate the QuestionItem with the Instruction
- If you need to document this question in the context of a survey:
 - o A StudyUnit holding a DataCollection that wraps the above objects
- If you need to associate the Question with a Variable in the survey
 - o A LogicalProduct in the StudyUnit that contains a VariableScheme with the Variable. The Variable contains a Question Reference (which essentially also put the question in the context of a survey)
- If you need to associate Concept with the Question:
 - o A ConceptualComponent in the StudyUnit that includes or makes reference to a ConceptScheme. The Question has a reference that points to the Concept (note that this link can also be established via a Variable)



So, technically the minimal metadata required to define a question is the response domain. The question text itself is of course highly desirable as well. Note that the above is purely an internal storage structure; this can be presented in a user friendly way by end user applications.

System Administration and Security

In addition to the metadata from the conceptual model, the system will require storage and interface for maintenance, administration, system logs and security. These will need to be defined based on CESSDA requirements. Administrative services may include features such as system monitoring, logs, backup/restore, import/export, service management, etc... Security will likely play a considerable role in the system. While metadata can most of the time be considered public domain, this is not always the case and object level access control will likely be necessary. Note that this can have significant impact on the implementation complexity and system performances. Security constraints should therefore be kept to a minimum.

Other Services

In addition to the web services described above, other system wide services will likely be deployed to support the registry, QDB, 3CDB, and repositories. This includes:

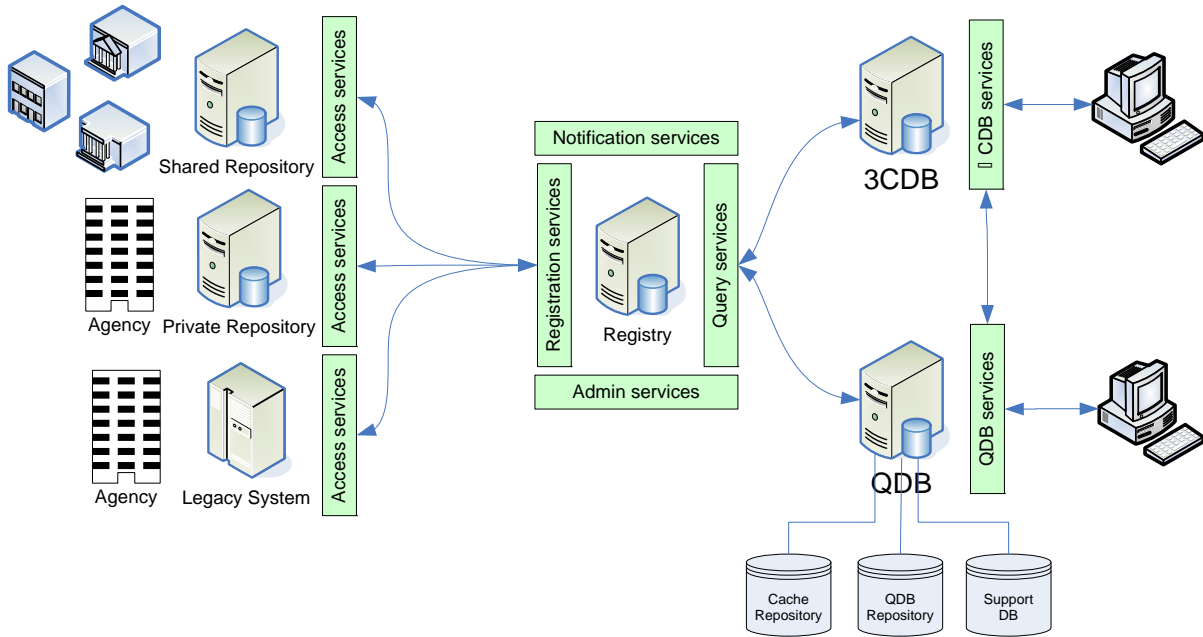
- Security services (user authentication, authorization)
- Multilingual and thesaurus services
- Reporting Services
- Virtual File storage (for external resources or access to data files)

Summary

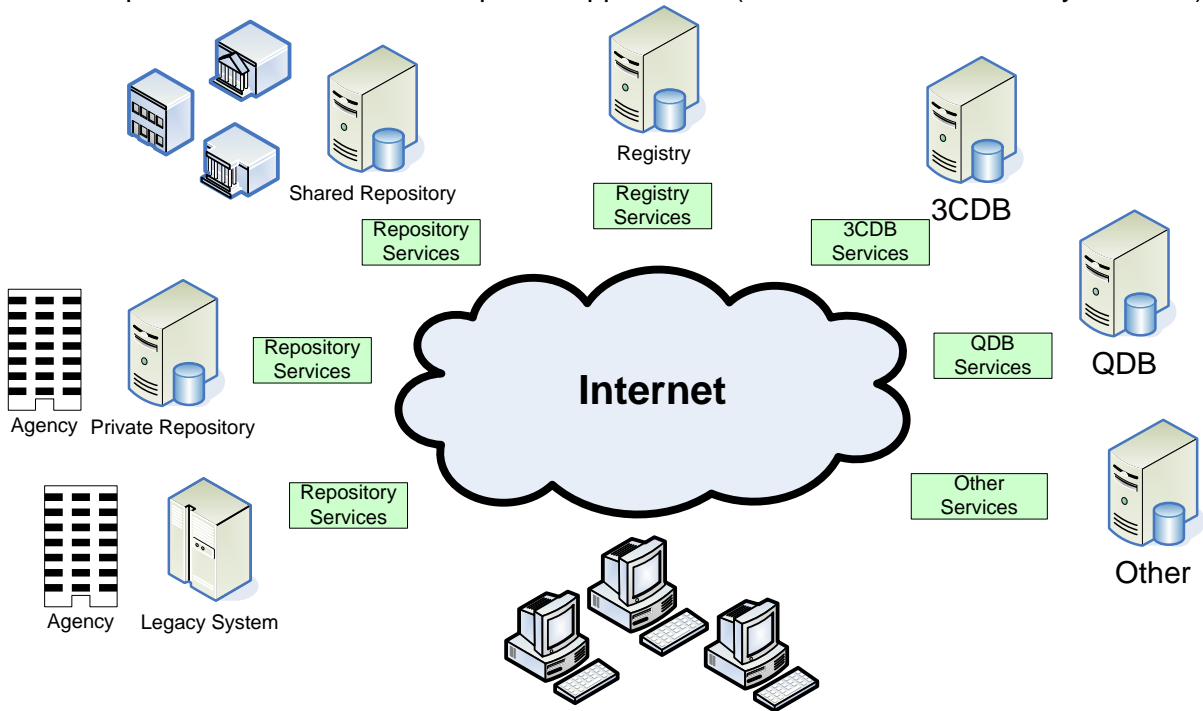
The proposed architecture provides a loosely coupled web services oriented environment that meets the CESSDA needs and isolates the different components.

It provides the following benefits:

- Support for 3CDB, QDB and future CESSDA applications
- Open to non-CESSDA applications (public services)
- Can scale based on the local needs: repositories, registry, 3CDB and QDB can grow independently
- Technology and platform independence
- Compliance with XML specifications (can map to DDI and other standards such as SDMX, Dublin Core, etc.)
- Integrates with other services (CESSDA and non-CESSDA) using industry standard technologies



Note that in reality, the arrows between the various applications are not really relevant as all applications can talk to each other over the network. Relevant access control mechanisms will be in place to restrict access to specific applications (firewall and other security software)





Registry

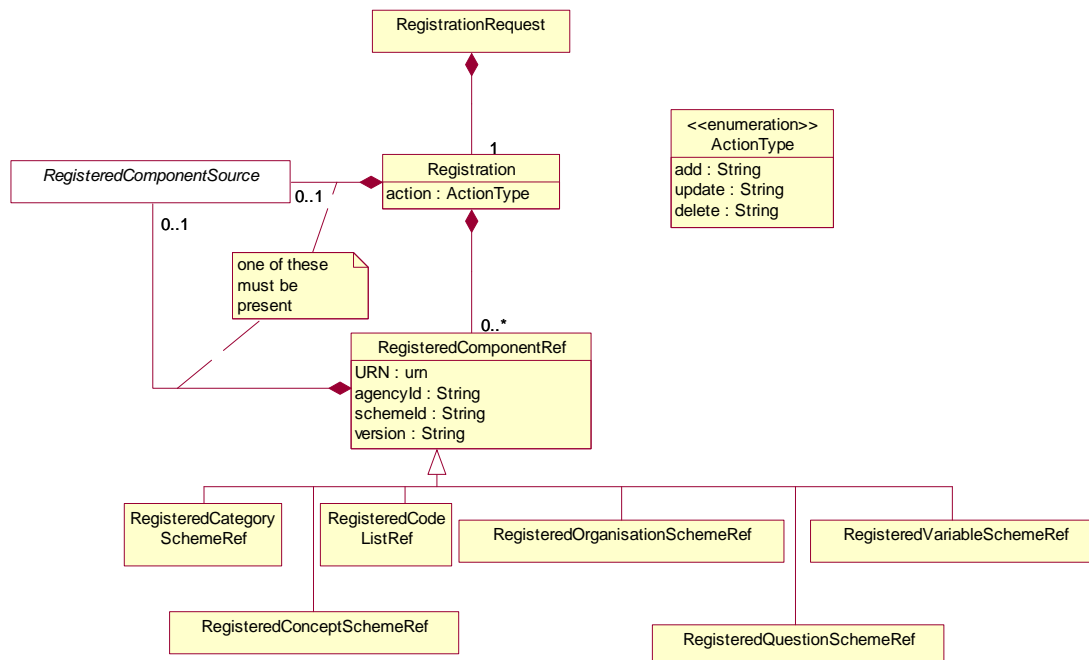
Scope

The purpose of the registry is to enable applications to find resources such as Questions, Concepts, Variables etc.

In order for the registry to supply this information the resources must be registered. Applications can then query the registry to find the resource required, and also to discover which resources reference other resources e.g. which Concepts are referenced from which Questions.

Conceptual Model

Registration



The Registration is contained in a RegistrationRequest (this will contain relevant administrative information). The Registration defines the action to take (add, delete, update). This action specifies the action on the details stored for the registered content: it does not specify any action on the actual artifact itself which is not held in the registry and is maintained independent of the registry. For example, a replace action on a ConceptScheme will replace the current registered details for that ConceptScheme, it will not update the ConceptScheme itself.

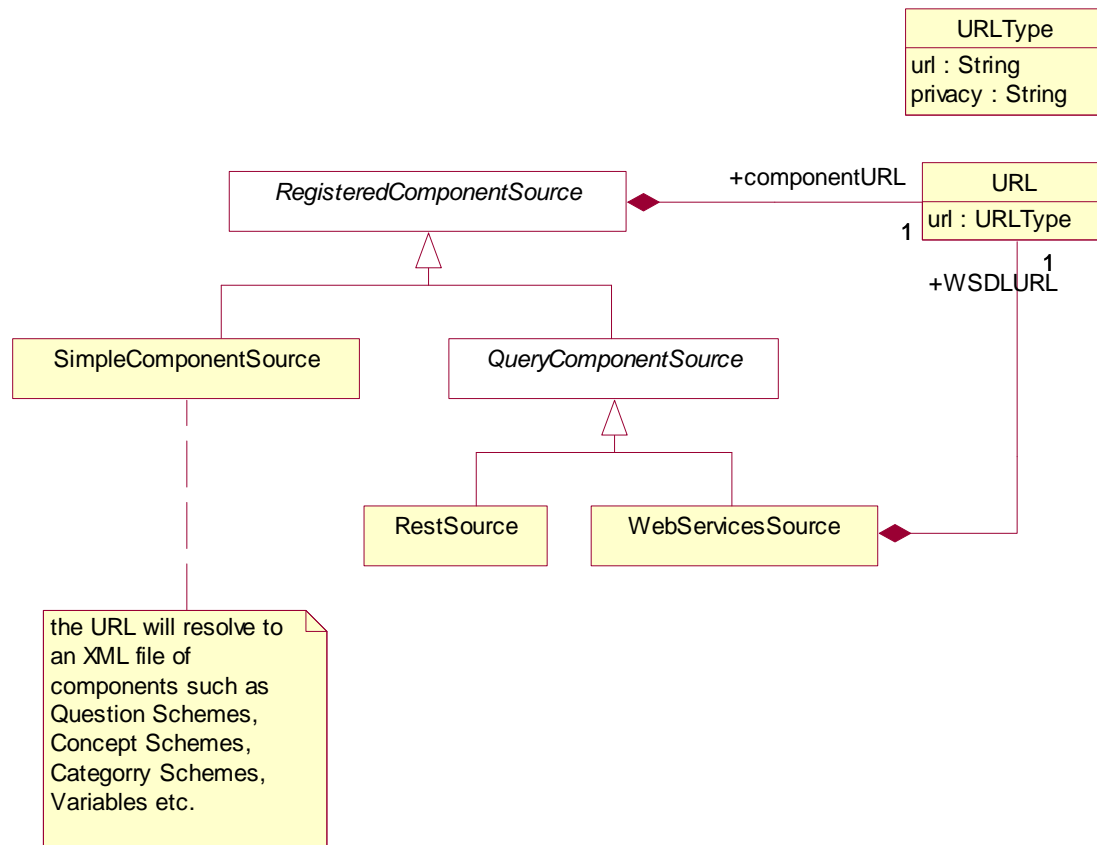
The Registration may contain the identifiers of the actual resources that require registering so that the registration process can index them accordingly, or can query for them and then index them. This is only relevant if the RegisteredComponentSource is a QueryComponentSource (see below). If the source is SimpleComponentSource then the registration process can, by default, register all of the components in the source file.

Although not shown here, it is also possible that the actual resource to be registered is embedded with the registration request. In this case, the registration process need not



retrieve it from the source, as it has all the information it needs to register and index the component.

Registered Source



The RegisteredComponentSource is a URL from which the components can be obtained. If this is a SimpleComponentSource then the components are contained in an XML file. The registration process will retrieve the file and index its contents. If this is a QueryComponentSource then the URL must be a web service that can be queried for the component(s). The list of components is contained in the Registration (MaintainableArtefact). Each MaintainableArtefact will be identified and this allows the registration process to retrieve each artifact and index it. Alternatively, the full contents of the artifact can be submitted in the Registration, and if this is the case, then the registration process will build the index from the submitted component.

The service identified in the QueryComponentSource must be able to consume an XML formatted query and return an XML formatted component (e.g. ConceptScheme, Concept,,Variable etc.)

Indexing

The Registry index process extracts relevant information from the registered artifact in order to support queries. It is expected that the indexed information will contain, for each component:



- The URL of its location (file containing the component or queryable source)
- The child components of the artifact (e.g. the Concepts in a Concept Scheme)
- The unique identifier of each of the component
- The unique identifier of any referenced component

Note that in order for the referenced components to be discovered in the registry, these must also have been registered.

Using the Registry

The Registry contains information that can be used by applications to assist in the discovery of linked components such as which Questions are used by which Variables. The Registry itself does not know the actual use cases to be supported: its job is to index the information submitted, support queries, and return registered information. Applications using the Registry will use this information to provide services to users. The Registry is not responsible for retrieving the information from a registered source, except when it is originally submitted for registration/indexing.

Registry Interfaces

Scope

The Registry interfaces allow applications to:

- submit structures for registration and indexing
- query for structural components
- query for usage of components (e.g. where a Question is used)
- subscribe to registry events
- be notified of registry events

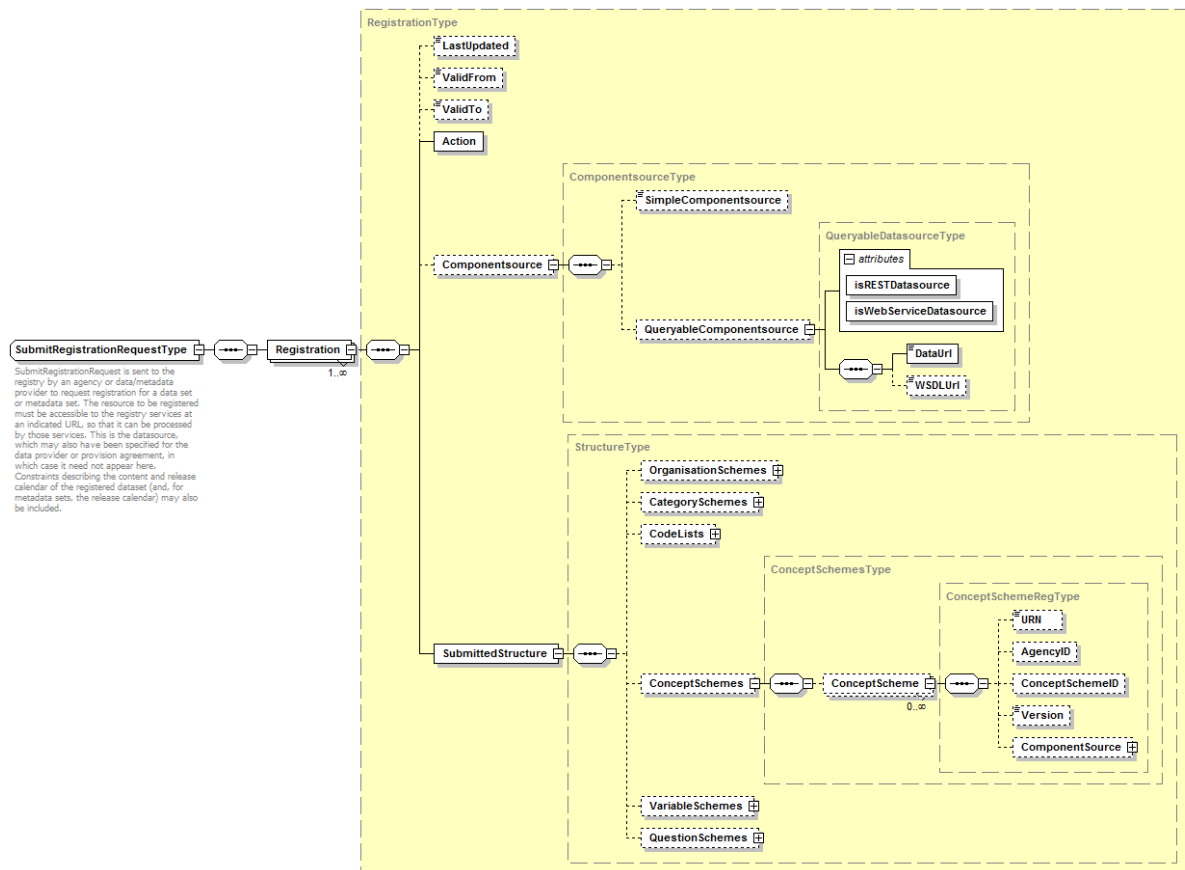
The draft interfaces below show the registration, querying etc. for complete schemes (e.g. Concept Scheme). In the full system this will clearly be extended to include individual components (such as Concept).

The specification of the actual queries that are supported by the Registry are not in the scope of this document. The queries, and therefore the content, that needs to be indexed needs careful analysis and this will include the actual needs of the users and other “bank” specific systems. It is not the intention of the Registry to replicate the detailed searching that would be available in a “bank” such as a QDB. Rather the intention is to support queries that link the “banks” such as “what variables use the concept ‘Age’”, “what questions are used to determine the values of the concept ‘Age’” The Registry can respond to these queries and provide the links so that the user application can be directed to the actual QDB that contains the relevant artifacts. More detailed queries can be made once inside the QDB.

An important aspect of the Registry approach is that the Registry offers a common interface to external applications. The Registry may, but is not necessarily required to, have its own user interfaces (save for administrative interfaces). Rather, applications supporting a specific “bank” such as the 3CDB will interact with the Registry in order to discover the linked components. The end user will stay connected to the 3CDB application that will render the information returned from the Registry in whatever form is appropriate.



Registration

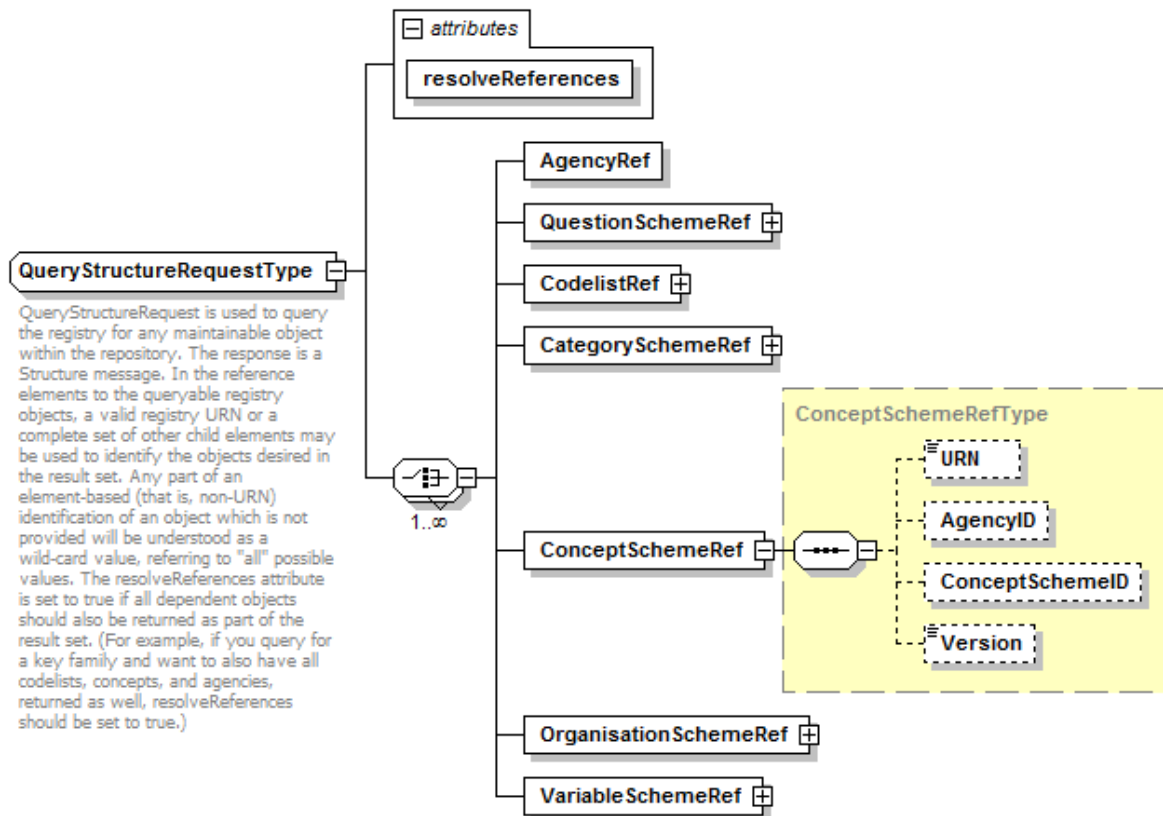


Only complete schemes can be registered. If the source of the component is the same for all of the Submitted Structures (this would typically be the case where the source is a queryable source such as a database), then the ComponentSource can be declared just once. If this is not the case (this would typically be the case where the source is a simple source – i.e. a file) then the ComponentSource is declared for each individual component.

The registration process will retrieve the component from the ComponentSource and index its contents e.g. index the Concepts in a Concept Scheme, index any relevant usage of other components (e.g. A Code List used by a Question in a Response Domain).



Query Structures

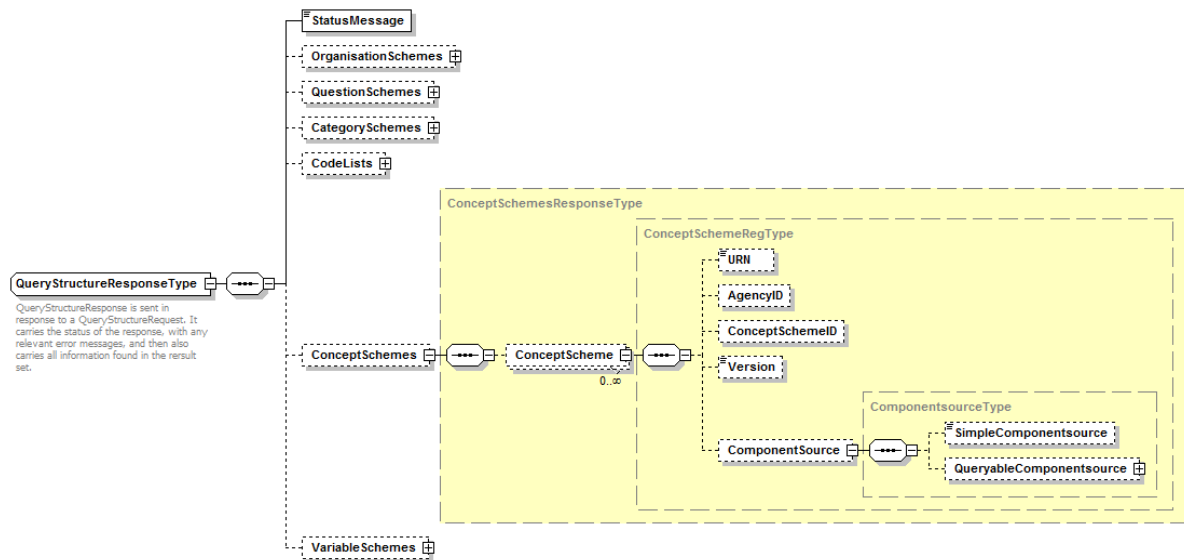


Structures can be queried by giving the unique id of the component (either by URN or by the composite id of the object). The query can be quite flexible such as:

- if only the AgencyRef is supplied then details of all components of that agency are returned
- if only the AgencyId of a structure (such as a ConceptSchemeRef) is supplied, then all objects of that type for the agency will be returned.

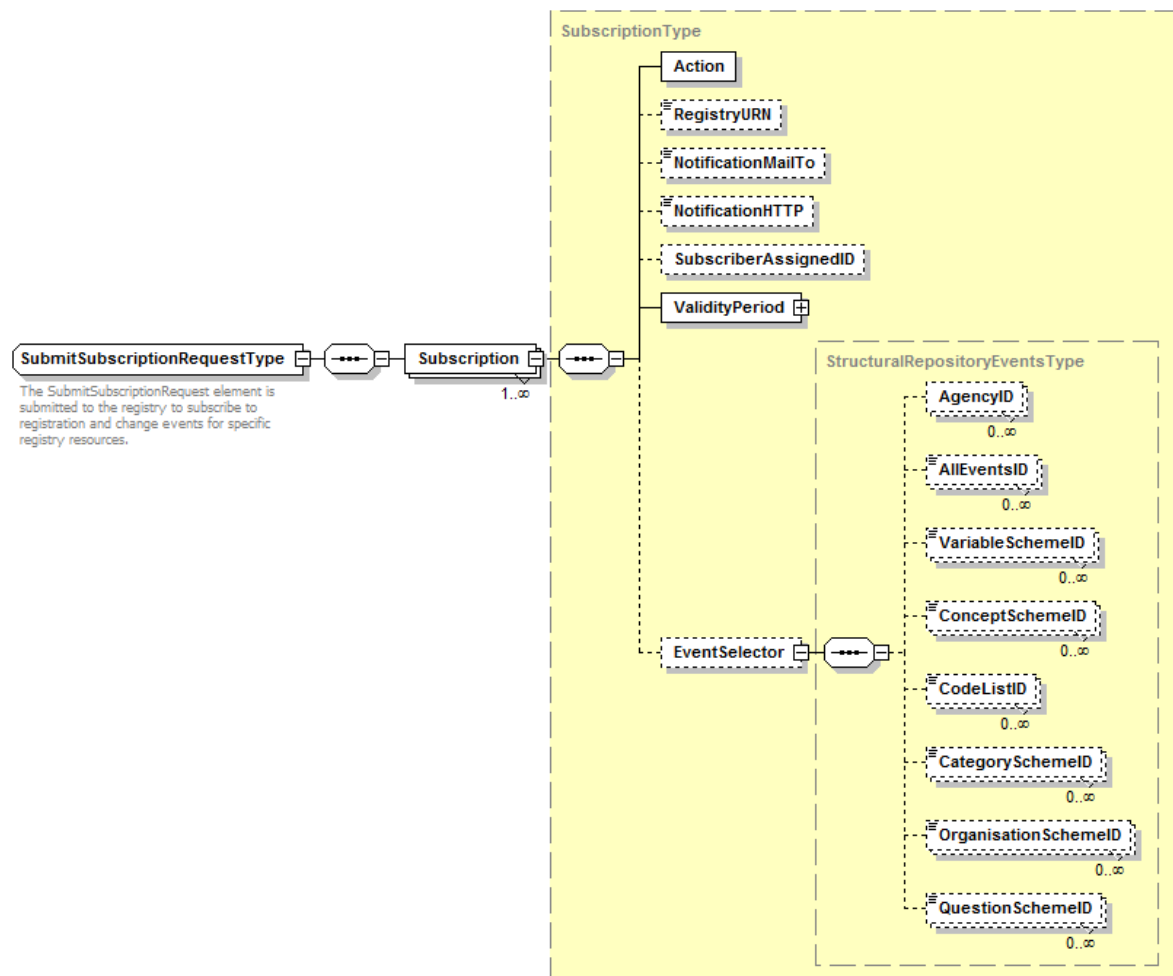


Query Structure Response



The response to the query will return all objects that meet the query parameters. The location of each of the objects is returned (ComponentSource).

Subscription





An application can subscribe to events that occur in the registry. Typically, the events will be a change to a specific object such as a Code List. Like the query, this is a flexible mechanism. If just the AgencyId is submitted, then a change (including a new registration) to any structure maintained by that agency will be notified to the subscriber. If AllEvents is submitted, then all events in the registry will be notified. If there is no specific content in the element then the subscription is to all objects of that type

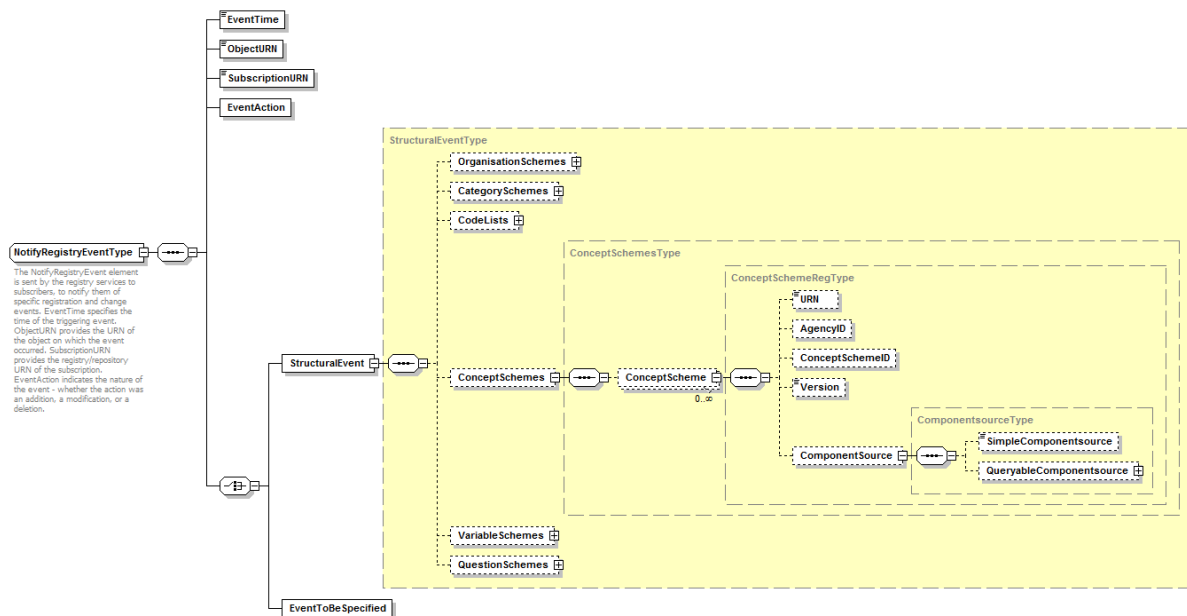
e.g. <CodeListId>

will subscribe to events on all code lists.

The subscription can be limited to a ValidityPeriod.

The subscription gives details of how the notification is to be made (e-mail, direct to an application (NotificationHTTP)).

Notification

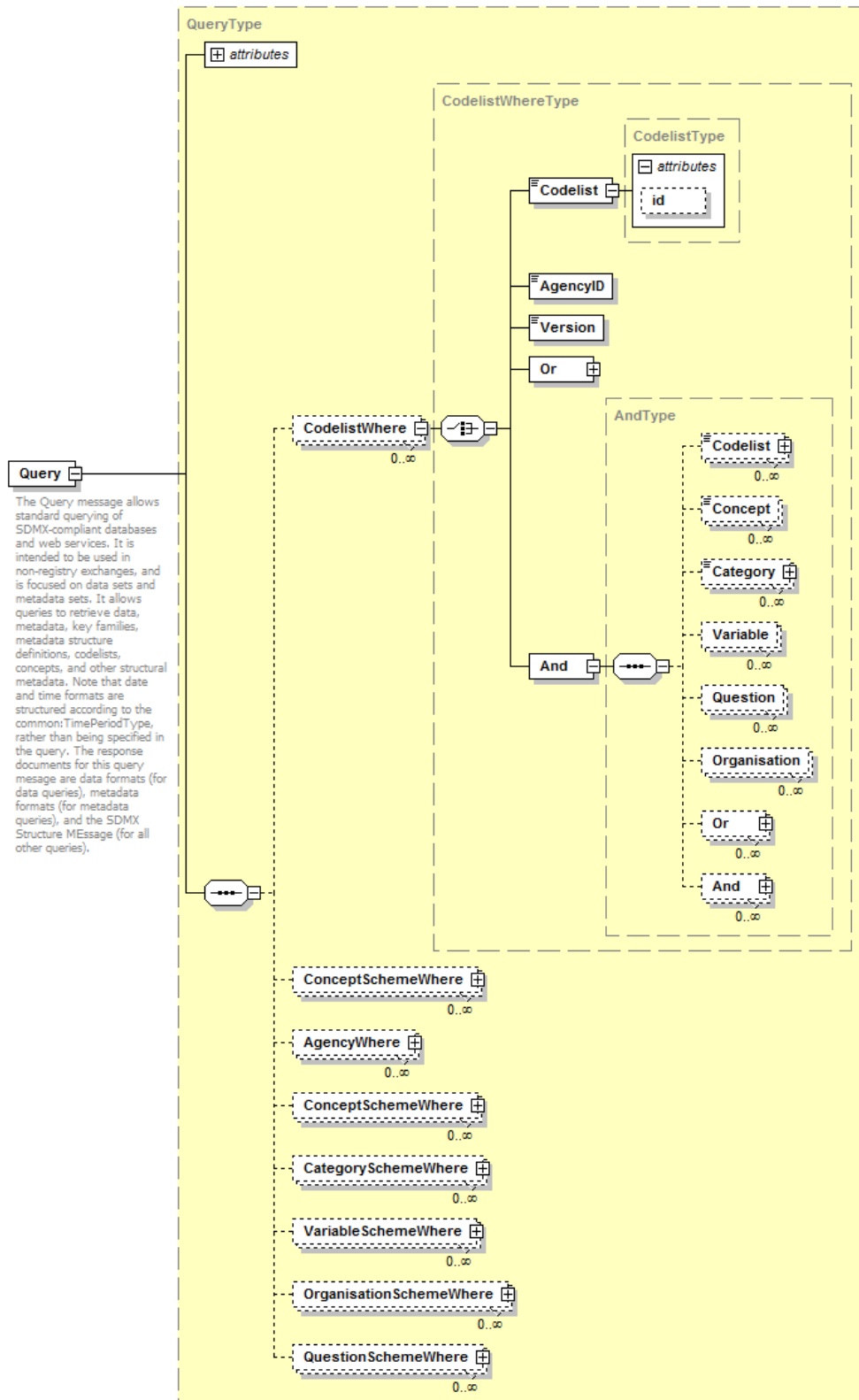


The notification gives details of the structure that has been changed/added. The notified application is responsible for retrieving the structure and determining the actual change.

An EventToBeSpecified indicates that further events can be added in the detail specification stage.



Query for Components



The query is consumed by a service such as a question bank service, or the 3CDB service (Concepts, Codelists etc.) and returns the information requested in an agreed format (DDI 3.0). The query schema shown above gives an outline only of the structure. It is possible in the query to `<And>` and `<Or>` the query constructs to query for lists of Codes or Concepts, Categories, Questions etc.



Core Services

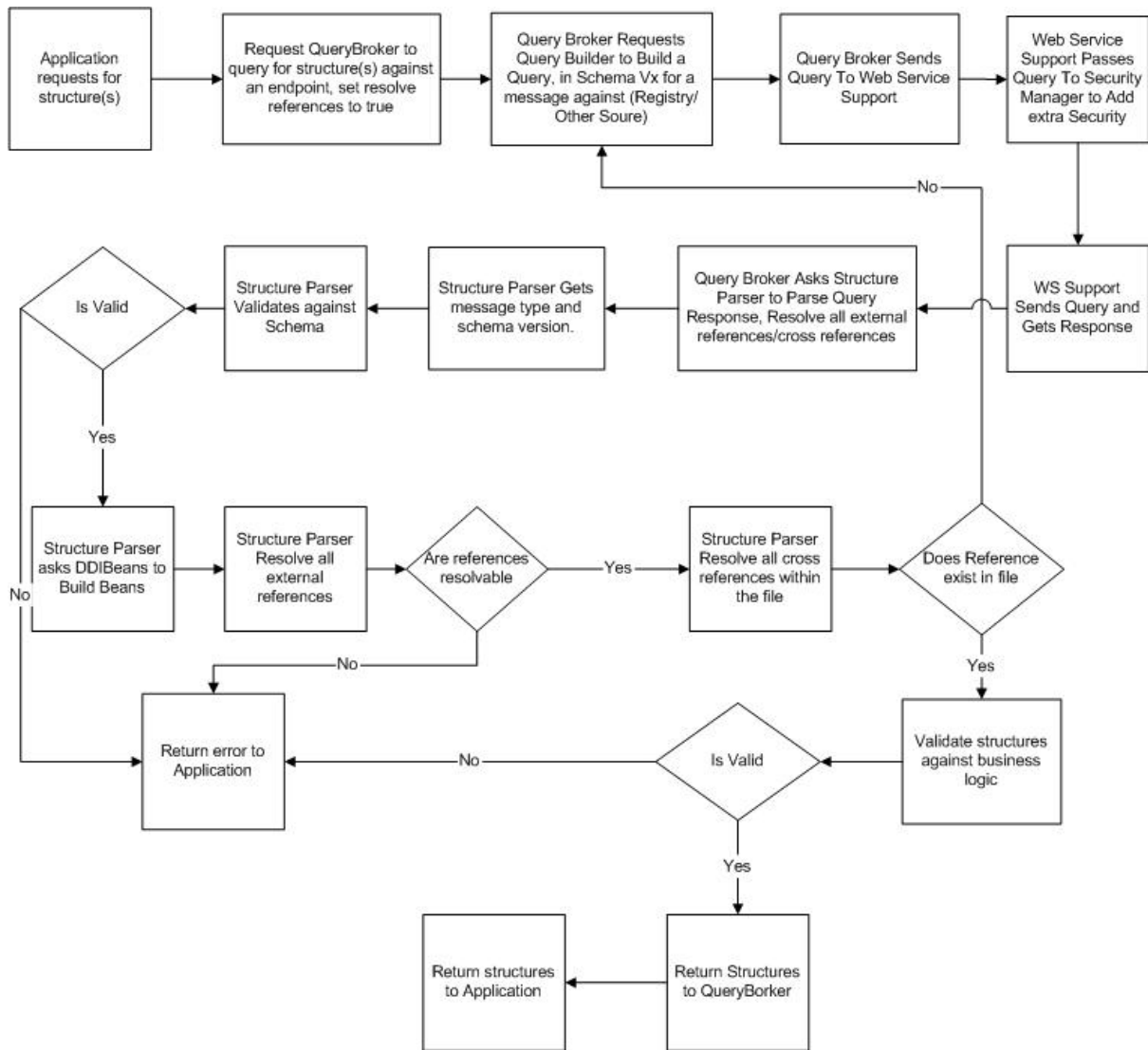
In order to support applications that require access to components, a set of core services is envisaged. These services will perform the following functions:

- Generate queries for the Registry
- Generate queries for queryable sources
- Retrieve components
- Resolve referenced components if required (e.g. obtain referenced components)
 - Embedded in the same source/repository
 - Referenced as external
 - Not referenced as external - this may involve further registry queries to locate the component
- Validate the components returned
- Resolve any references in these returned components (this can be iterative)
- Transformations between versions and standards

The table below defines the different processes.

QueryBroker	Queries the source repository for structural metadata (CESSDA Registry)
QueryBuilder	Builds a query in any schema supported (e.g. DDI 3.0), to either a RegistryRequest or a Query Request
WSBroker	Queries REST or SOAP repository for structures, supports GZIP
StructureParser	<ul style="list-style-type: none">• Validates structures against schema• Validates structures against business logic• Transforms structures between all schema versions and DDI Beans• Resolve cross references• Resolve external references
Security	Applies relevant security to structure repositories
DDIBeans	Creates schema independent Java Beans from DDI structural metadata (all versions)

A typical scenario to support an application requiring information on components is shown below. Note that multiple sources may be registered but the application must pass to the QueryBroker the URL of the web service that is the Registry to be used.



In this scenario an application queries for structures such as questions, concepts etc. The query support processes find where the resources are located (by querying the registry), and then the actual location of the components are queried to find the components. All referenced components are also located and retrieved. The retrieved components are validated to ensure they conform to the XML schema. Transformations are made between different schema versions is required.

Note that the requesting application does not need to know where the resources are located, nor does it need to query the services that will return these resources. All these functions are performed by the Core Services.



Question Bank Conceptual Model

Scope

The scope of the CEESDA Question Bank (QDB) is to contain all of the necessary information for a user (e.g. researcher) that supports:

- immediate comparison between questions by presenting different versions (national or time-specific) of a question in a display
- the development of new questionnaires by giving researchers access to a selection of viable operationalisations of a construct

In order to achieve this it is necessary to be able to search for questions via concepts and classifications managed in the 3CDB.

In terms of the DDI version 3.0 conceptual model the QDB will contain not only questions but also other constructs that will be necessary to place the question into one or more contexts; i.e. the operationalisations. This means that some constructs from other parts of the DDI model will be required, either as references to a location or repository where it can be found, or contained explicitly in the QDB.

Location of Objects

The conceptual model makes no assumption on where these constructs are located. However, the model does make an implementation demand that all objects in a **composition aggregation** are contained as part of the parent object i.e. they cannot exist on their own and must be co-located with the parent.

In UML an association with an aggregation relationship indicates that one class is a subordinate class (or a part) of another class. There are two types of aggregation, a simple aggregation where the child class instance can outlive its parent class, and a **composition aggregation** where the child class's instance lifecycle is dependent on the parent class's instance lifecycle. It is the objects in this second type of aggregation that must co-exist in the same location as the parent object, whether this be in the QDB itself or in a different repository.

In UML the aggregation is shown by a diamond at the parent class end of the association. A filled-in diamond represents a **composition aggregation** and a non-filled-in diamond represents a simple aggregation.

Furthermore, the diagrams below show relationships between major constructs in DDI such as `Question`, `Variable`, `Concept`. These constructs are maintained in their own schemes and the associations are of a reference type indicated by the arrow at the target class end of the association (in strict UML terms the arrow means that the association is navigable only in the direction of the arrow).

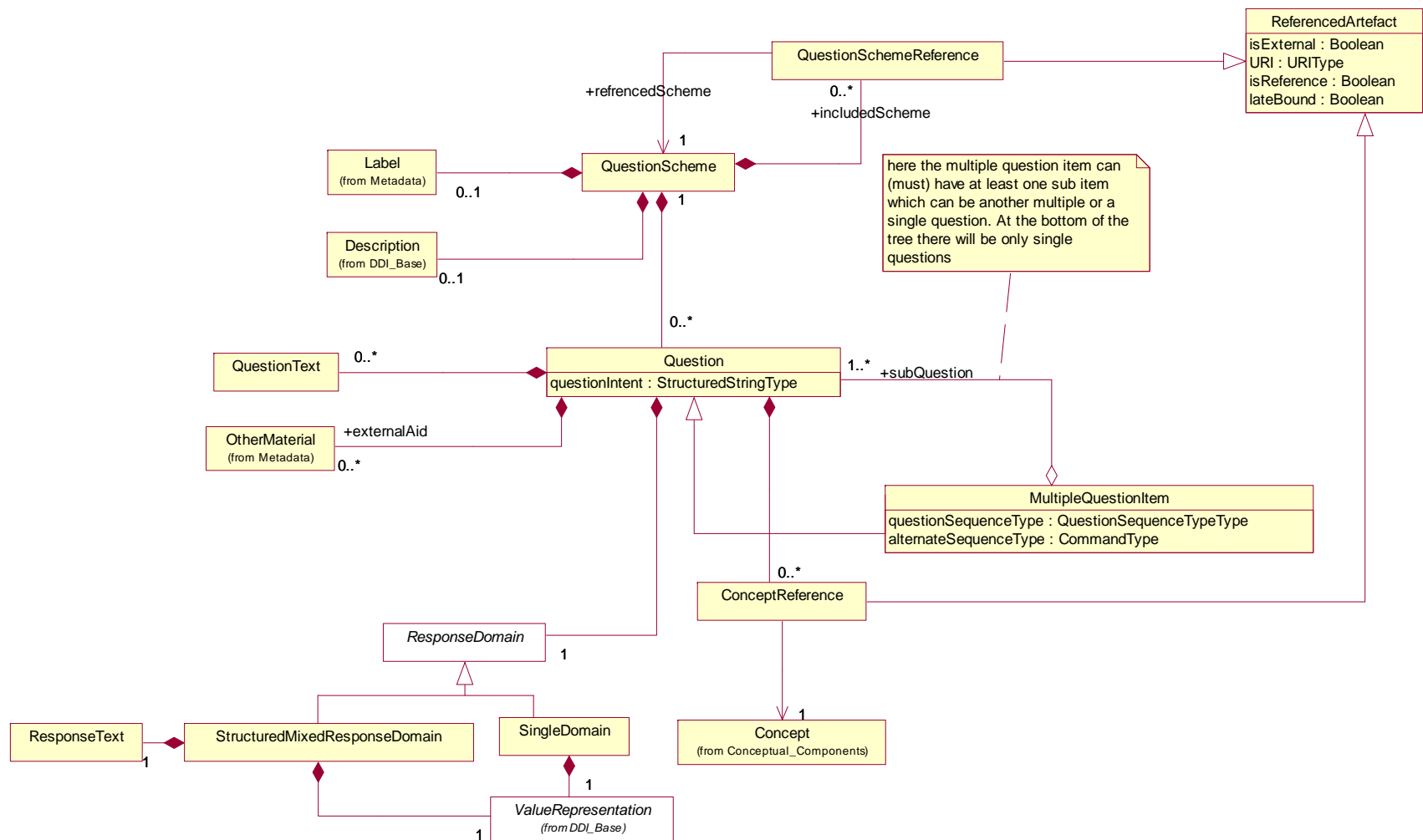
In other words, the referenced construct does not belong to (i.e. is not maintained in) the scheme from which it is referenced. The scheme in which it is maintained need not, therefore, be in the same repository as the scheme from which it is referenced. For example, a `Concept` (which is maintained in a `ConceptScheme`) may not be located on the same repository as the `Question` from which it is referenced.



These are, however, implementation decisions and, indeed, at the implementation the application actually using the construct may not have any knowledge of where these constructs are located as this may be resolved by a separate “repository resolver” process.



Question Scheme – Core Model





The `QuestionScheme` may include the contents of another `QuestionScheme` (`QuestionSchemeReference`). This scheme may reside in the QDB but may exist external to the QDB in another repository (`isExternal = "true"`)

The `Question` is maintained in a `QuestionScheme`. A `Question` may have a sub structure of multiple questions (which can have a prescribed sequence (`QuestionSequenceType`), and any of these can also split into multiple questions.

The `Question` can be documented with a `questionIntent` – i.e. the purpose of the question vis à vis the data being collected.

The `Question` may reference one or more `Concepts` that are linked to the `Question` (e.g. a question “how old are you” may be linked to the `Concept` of Age). The `Concept` may be maintained in a `ConceptScheme` that resides in the QDB, but this scheme may exist external to the QDB in another repository (`isExternal = "true"`).

The `Question` may be documented with a pointer to an `externalAid` presented by an instrument in which may be used to aid the person responding to the question, such as a text card, image, audio, or audiovisual aid. Typically this is a URN. The type attribute of `OtherMaterial` is used to describe the type of external aid provided. Example would include: `imageOnly` `audioOnly` `audiovisual`, `multiMedia`. The detail of `OtherMaterial` is described below.

The `Question` may have `QuestionText`. `QuestionText` is just one form of `DynamicText` in the DDI and the structure of this is described below. There may be multiple language versions of the `QuestionText`.

All questions have a designated `ResponseDomain`. This may be a reference to previously defined `CategorySchemes` or `CodingSchemes` (see the detail of `ValueRepresentation` below).

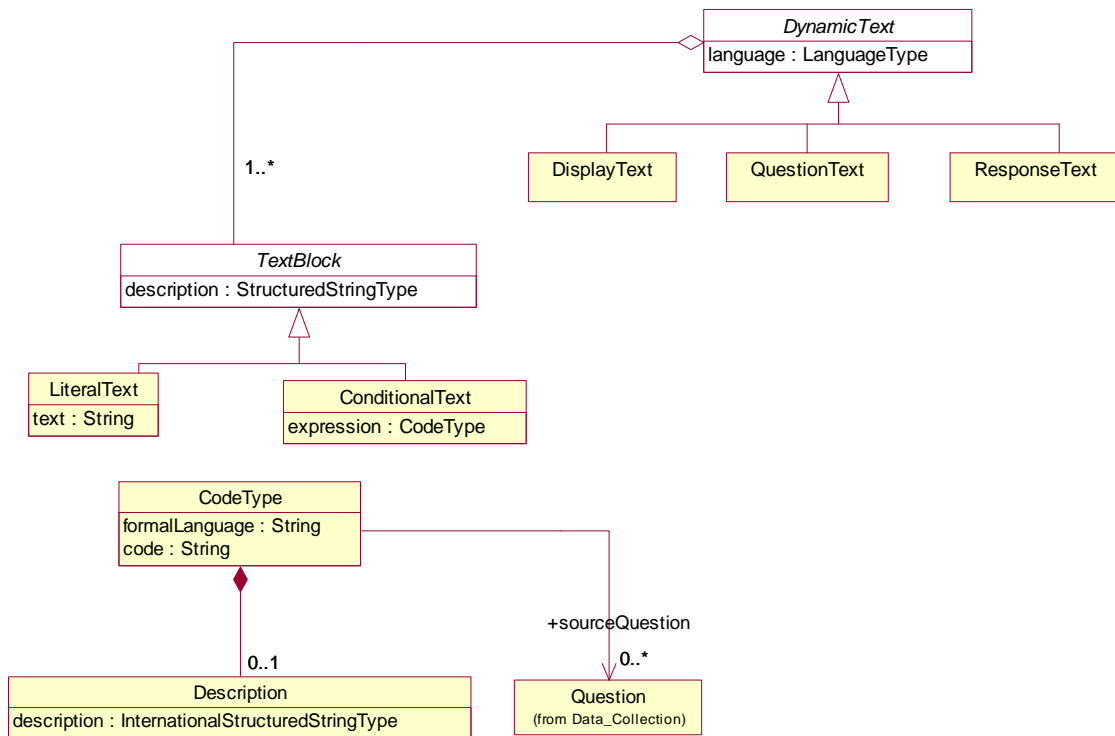
Question Scheme – Embedded Constructs

Scope

These constructs are an integral part of the `Question` and therefore of the QDB.



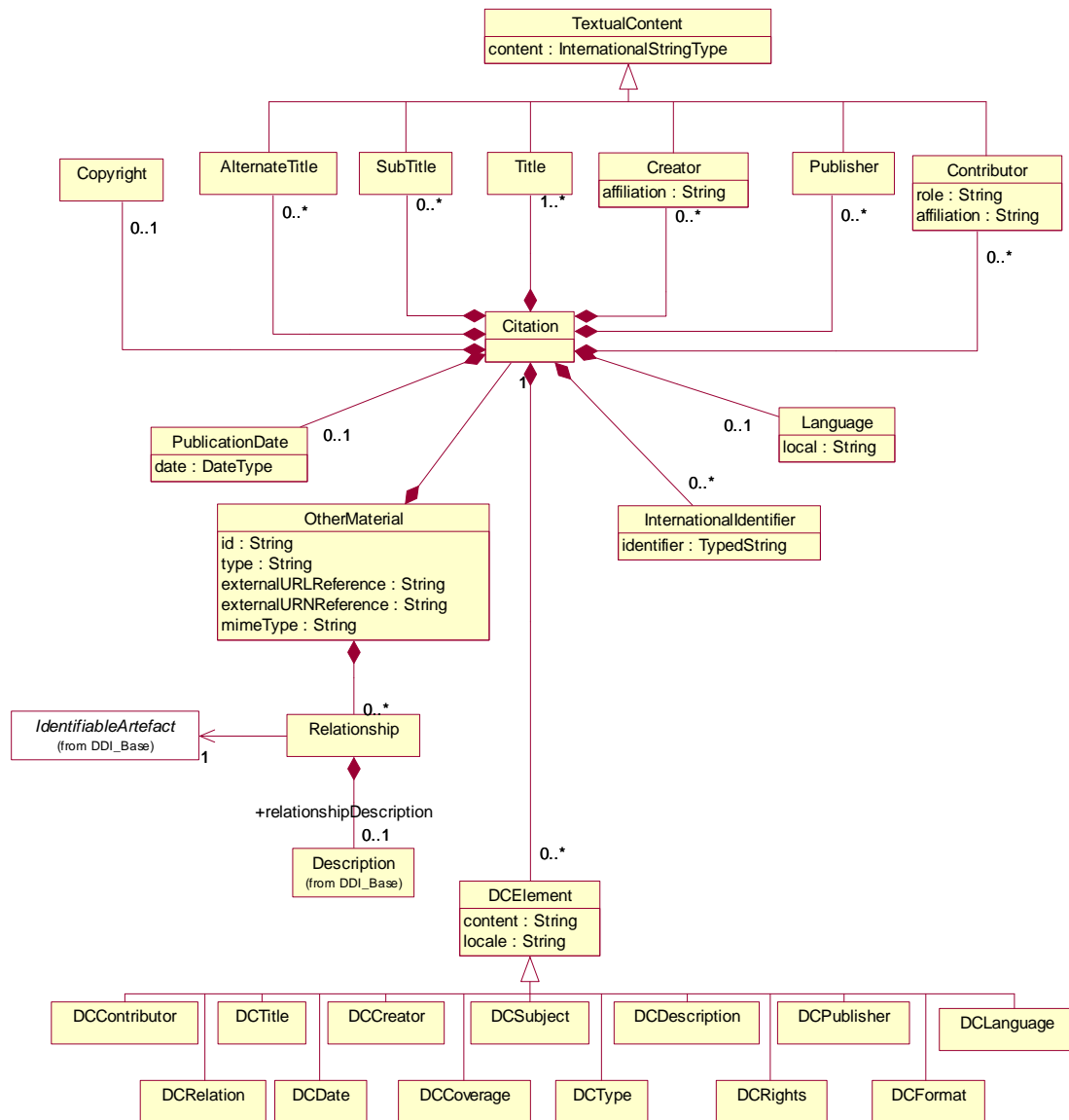
Dynamic Text



QuestionText is a form of DynamicText. This comprises one or more TextBlock which can be either LiteralText (i.e. it is actual text) or it can be ConditionalText, which itself can be linked to one or more Questions.



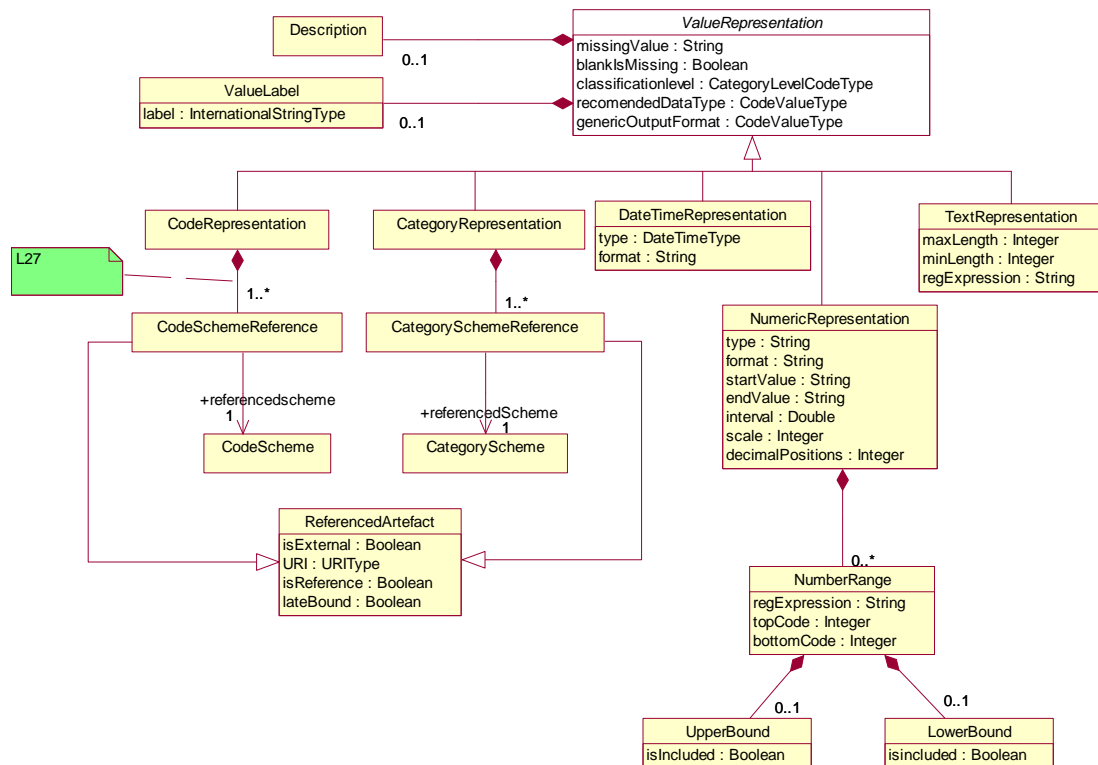
Other Material



This is used to contain the `externalAid` information. This can be a reference (URI or URN) of where the information is to be found, or the information can be explicitly defined in the QDB. There must be `Citation` information which is mostly textual (this can be based on Dublin Core (`DCElement`) and/or other construct (`TextualContent`)). The `externalAid` information can also describe (`Description`) a relationship to another object (`IdentifiableArtefact`).



Value Representation



This contains the valid representation for a ResponseDomain. This can be one of the following types:

- A CodeScheme (CodeRepresentation)
- A CategoryScheme
- Date/Time (DateTimeRepresentation)
- NumericRepresentation
- TextRepresentation

Note that the actual CodeScheme or CategoryScheme may be held externally (`isExternal="true"`).

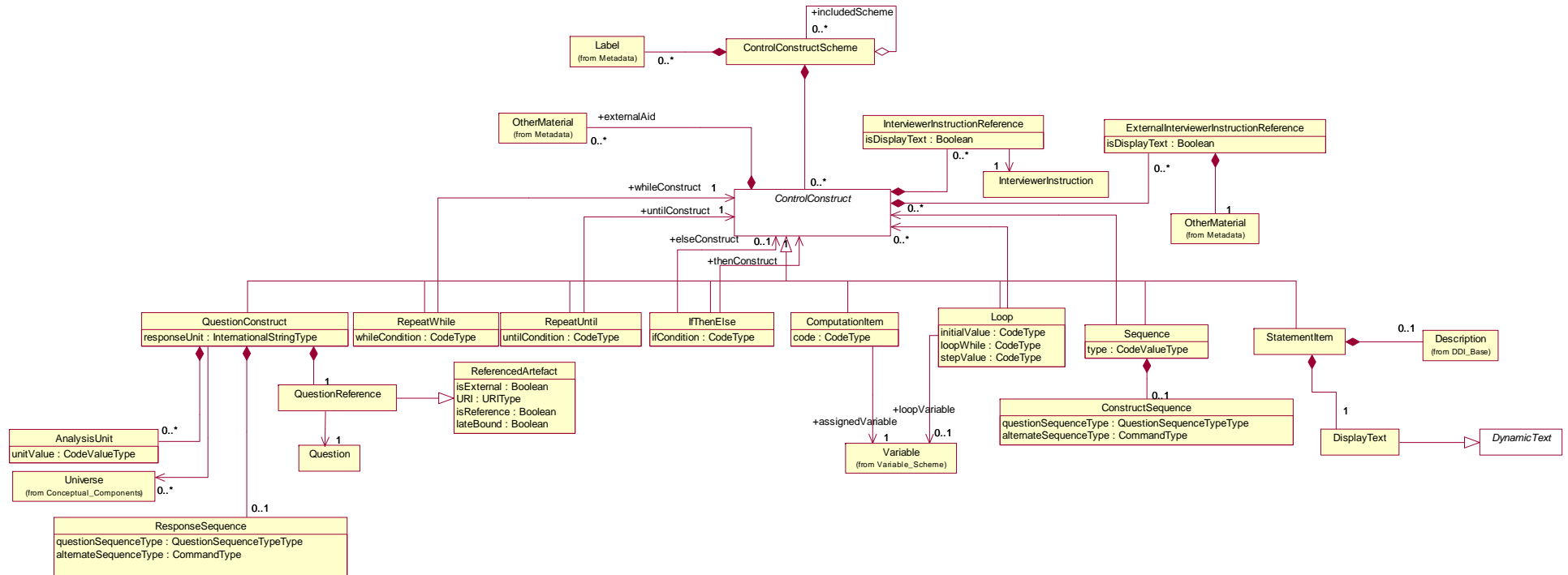
Question Scheme – Associated Constructs

Scope

These constructs are either referenced directly from constructs in the QuestionScheme or have an association with the Question i.e. the construct references the Question or QuestionScheme.



Control Construct Scheme



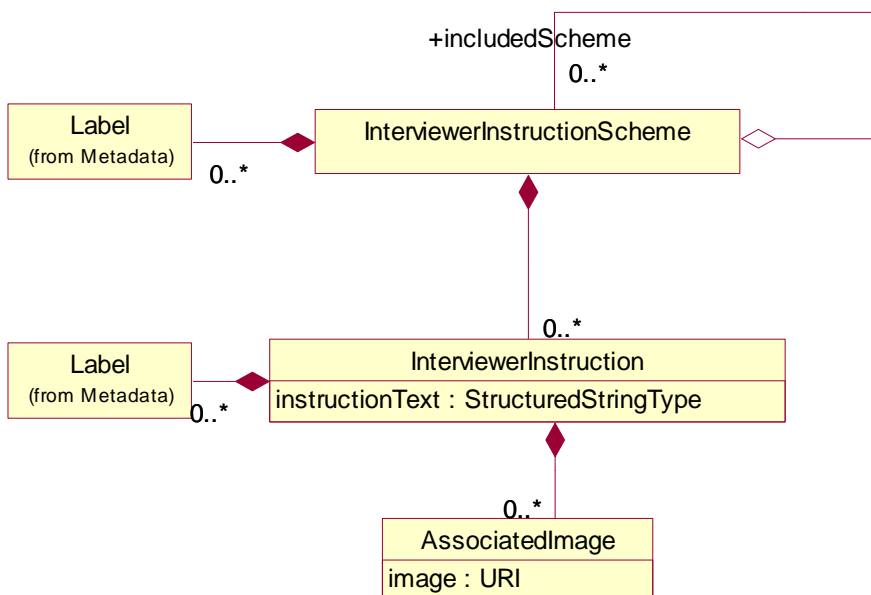


A Question is just one of many possible ControlConstructs (QuestionConstruct) that are maintained in a ControlConstructScheme. Note that the actual Question may be held externally (isExternal="true") in a QuestionScheme. For example, A ControlConstruct of type QuestionConstruct used in an Instrument may reference a Question that may be stored in the QDB, whilst the actual ControlConstructScheme resides in a different repository.

In order to enable the researcher to understand the contexts in which the Question is used, it will be necessary for the QDB to have knowledge of which ControlConstructs in which ControlConstructSchemes reference the Question, and possibly, in addition, the data collection instruments that use the ControlConstruct.

Furthermore, in order to understand the context of the Question it may be useful for the researcher to have access to the ControlConstructs that precede or follow the QuestionConstruct, or whether the QuestionConstruct is a part of a larger set of constructs e.g. IfThenElse, RepeatWhile etc.

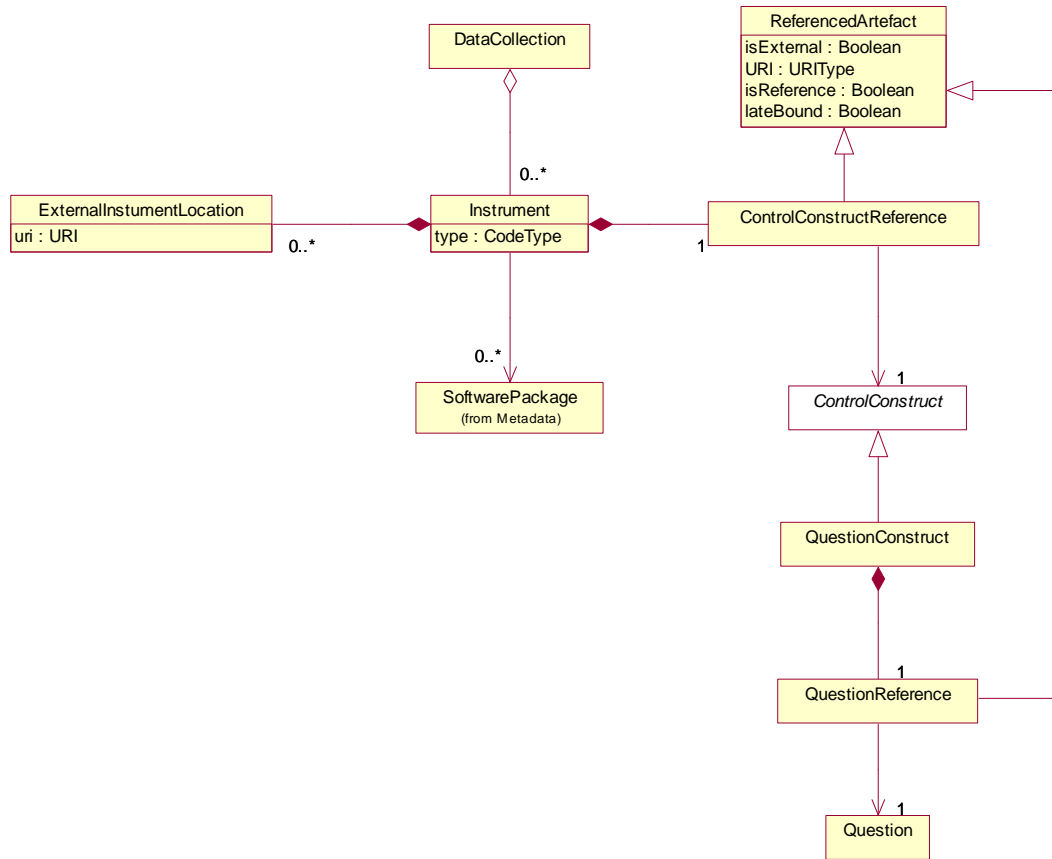
Interviewer Instruction Scheme



The QuestionConstruct may be linked to an InterviewerInstruction which is maintained in an InterviewerInstructionScheme. Access to these instructions could help the researcher understand better the context of the Question when used in a data collection instrument.



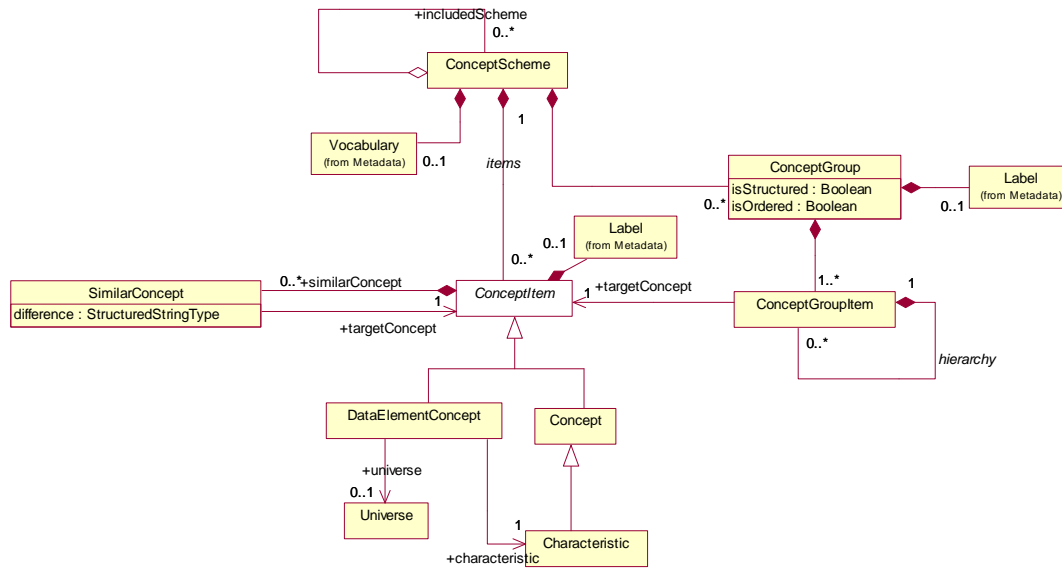
Data Collection



The `QuestionConstruct` (`ControlConstruct`) may be used in a collection `Instrument` which itself may be used for `DataCollection`. Access to the `Instrument` and related `SoftwarePackage` information could help the researcher understand better the context of the `Question` when used in a data collection instrument. The `Instrument` may reside in a different repository to the `ControlConstructScheme` and the actual `Question` referenced by the `QuestionConstruct` may reside in the QDB.



Concept Scheme

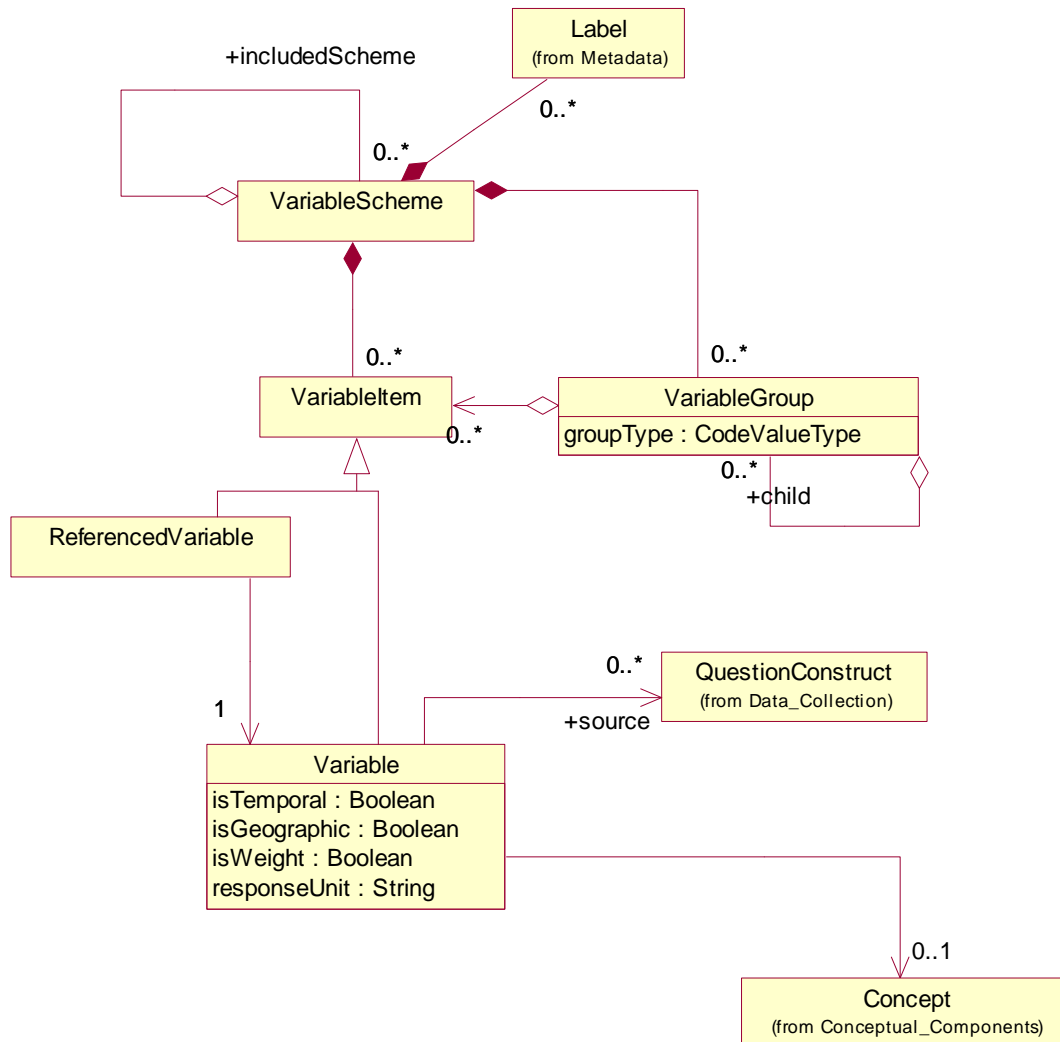


The Question in a QuestionScheme may reference one or more Concepts, and each of these may be in different schemes and may be held external to the QDB. Access to these Concepts could help the researcher understand better the scope of the Question.



Variable Scheme

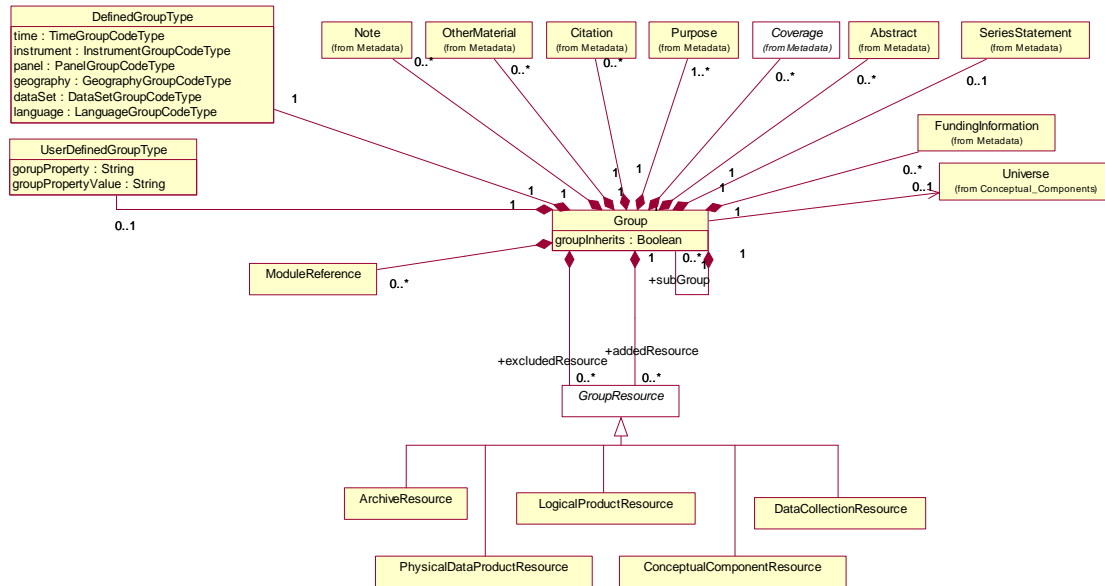
This is an abridged content of the VariableScheme. It does not contain the detailed structure of a Variable.



The Variable content may be derived from one or more Questions used in a data collection instrument (QuestionConstruct). Access to these Variables could help the researcher understand better the context of the Question when used to construct Variables.



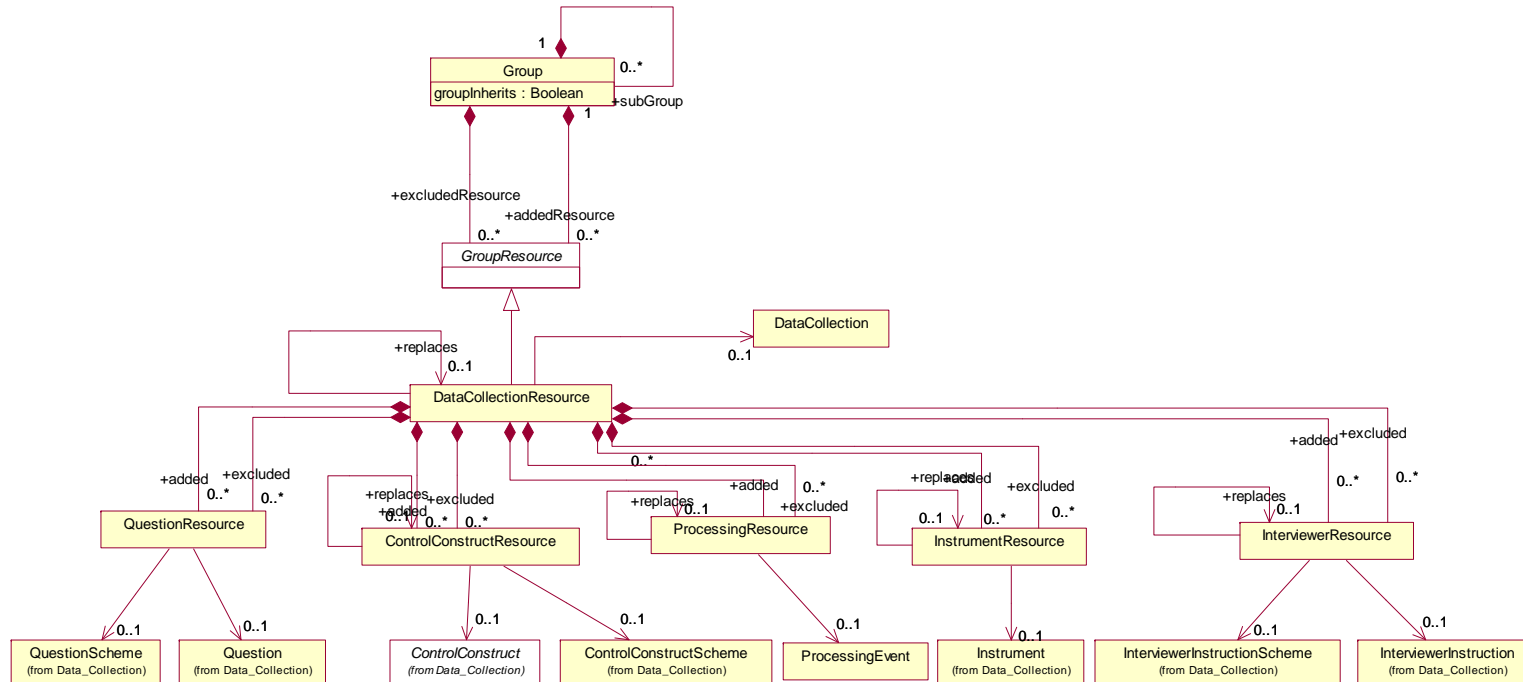
Grouping



The grouping mechanism in DDI allows artifacts to be grouped. Group provides an umbrella structure to pull together two or more studies into a structured series or unstructured group. It provides basic information on relationships among members of the group that affect processing decisions. Importantly, Groups can be specified in a hierarchy and lower level Groups can add or amend the artifacts inherited from the higher level groups.

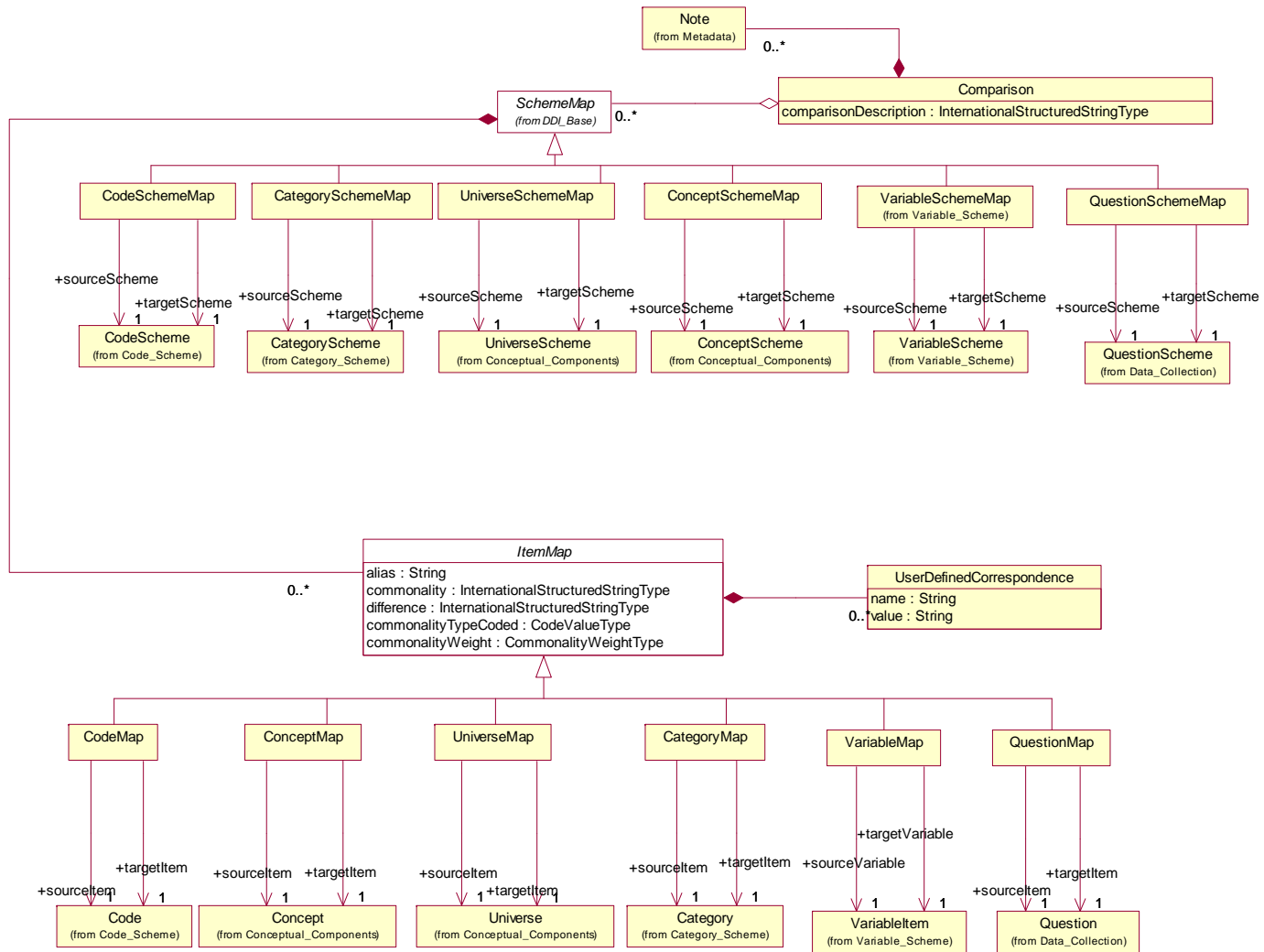
Studies can be grouped for a number of reasons but generally fall into the category of grouping by design or ad hoc groups. Grouping by design takes place when studies are either intended to be a series or when a repetition of the study takes place. The key factor is that the second study in the series is intended to inherit features of the first study (questions, variables, study design, universe, etc.) for the purpose of comparability. Group allows you to define which parts of the major components are shared, where overrides take place, and how to relate or link data in one study to data in a subsequent survey. Note that an unstructured group does not have inheritance as it is ad hoc and describes a single study.

Each of the sub classes of Group Resource has a sub model that shows the content of these exclusions and additions. Each of these is similar in nature and for the sake of brevity only the Data Collection Resource is shown here.





Comparability





The Comparability structure provides for pair-wise comparison of individual concept, question, or variable items. A particular useful case is to map different artefacts to a common harmonized structure, where each study unit is compared with the harmonized structure. Comparison between study units works on the principle "If $A=B$ and $A=C$ then $B=C$." The item level mapping structure allows the user to define the relationship, for example equivalency, parent-child, or relationship formulas.

The Scheme Map provides for identifying the source and target Schemes, a description of the correspondence, and a specific item map.

ItemMap provides for similar comparison for item pairs within the Source and Target Schemes.



Requirements and Use Cases

This section documents how the proposed architecture meets the requirement outlined in the initial CESSDA tender T216 document. It also presents a few additional use cases that illustrate specific system functionalities.

CESSDA Requirements

Must Have

1. The QDB supports integration with the Constructs, Classifications, Conversions Databases: 3CDB (a system for data harmonization). See Task 4 of work package 9 (http://www.cessda.org/project/doc/wp09_descr2.pdf).
→ Supported by design
2. The QDB integrates existing databases from CESSDA-partners (see should have 5).
→ CESSDA partners use private/shared repositories to publish their metadata or can build standard web services API on top of their legacy infrastructure. Generic and custom tools to exports to CESSDA XML format will need to be developed.
3. The QDB deals with questions and corresponding answer categories (or variable values).
→ This feature directly supported by the conceptual model through compliance with DDI3. The model also takes into account that questions may not always be related to a variable (or even use a questionnaire)
4. The QDB records the context of the question (neighborhood, respondent universe, sequence and position in questionnaire).
→ Supported by the conceptual model through compliance with DDI3 assuming that sufficient metadata is capture (universe, questionnaire flow, etc.). Universe harmonization/mappings will be necessary to support searches across multiple providers or data collections. Such harmonization can take place after the fact through the 3CDB.
5. The QDB allows storage and display of human interpretable routing-information.
→ Supported by the conceptual model through compliance with DDI3
6. The QDB allows storage of Computer Assisted Interview software (CAI)-routing-info if this is available. The source or type of the routing-info should be registered to allow future interpretation of the routing.
→ Supported by the conceptual model through compliance with DDI3. Tools already exist to import such information from Blaise, CASES, and CPro.
7. The QDB allows searching for questions and returns search-results that enable direct comparison.
→ Information can be retrieved from the system but how it is displayed is an application issue. Comparability information can be inferred from the metadata or captured using the DDI model comparability features.
8. The QDB allows storage of current and future questionnaires.
→ Existing questionnaire can be published by providers. New questionnaires can be constructed using the QDB and registered. Search can occur in both the local QDB



repository and the registry.

9. All QDB software under full control of CESSDA (no proprietary software).
→ *This is an implementation issue. In some case, proprietary / commercial packages may be desired for performance issues (see Technologies p.71)*
10. Where possible, open standards are used.
→ *DDI, DC and other metadata standards are at the core of the design*
11. The QDB must be prepared for integration with a question reuse module (a system to create new questionnaires based on existing questions).
→ *Reuse is a fundamental feature of the conceptual model (same as in SDMX or DDI). This information can be captured and stored locally by the QDB as well as published in the registry for reuse.*
12. The QDB allows registration of user-generated metadata with questionnaires and questions.
→ *User-generated metadata can be captured using Notes like elements or by extending the conceptual model. Generic Notes have the advantage to allow for any user attributes to be attached to metadata elements but then do not carry much knowledge about the “meaning” of the Notes’ content which makes it difficult to process by external application or users. Commonly attached user generated attributes should be integrated in the conceptual model. Examples would be useful.*
13. The QDB can register and recognize versions and translations of questionnaires and questions.
→ *Supported by the conceptual model through compliance with DDI3*

Should Have

1. The elements of Instrumentation-documentation and Comparison from DDI 3.0 (Data Documentation Initiative: <http://www.ddialliance.org/>) should form the basis of input and output of QDB (where possible).
→ *Supported by the conceptual model through compliance with DDI3*
2. The QDB is prepared for integration with existing and future CESSDA infrastructure.
→ *This is a fundamental feature of the design*
3. It should be possible to do multi-lingual search.
→ *This can be supported by the registry and the QDB. Multilingual search tools should however be available for all CESSDA applications and be implemented as independent utility services. This would then be integrated in the registry and the QDB.*
4. The QDB has a modular architecture to allow extension with new (future) functionalities.
→ *This is a fundamental feature of the design*
5. The questions themselves remain in the systems of CESSDA-partners (see must have 2).
→ *This is similar to “Must have” (2). Questions are stored in a repository that can be hosted at a partner side, shared amongst partners or be the QDB.*



CESSDA Tender Use Cases

Use Case 1: Optimization of Comparative Data Analysis

- The actor starts a search for survey data which contain variables used to measure specific concepts.
 - *the application retrieves the list of concepts from the registry (using the query service). This result can be a subset selected from a particular concept collection or based on text search. This can include thesaurus based searches.*
 - *the application then queries the registry for variables that are associated with the specific concepts. Further restrictions can be applied based on the survey collection, year, country, etc.*
 - Searching for free text
 - Browsing via a thesaurus
- CCCDB returns a list of items that match the search-criteria
 - *to query the registry, the application can actually directly interact with the 3CDB who can proxy the registry interfaces.*
 - *the result content is highly dependent on completeness and quality of metadata in the repositories.*
 - *the above assumes that the user was looking for variables. If the variables have been linked to a question and the related questionnaire flow has also been captured, the information below can be displayed. Another approach would be to perform a search for “questions” rather than “variables*
 - *Note that the descriptive metadata are not returned by the registry, QBD or 3CDB but by the repository where the object is actually stored.*
 - Variable descriptions
 - Question descriptions via the QDB
 - Survey questionnaires descriptions via QDB
 - Study Description (e.g. methodological aspects)
- Actor selects sub-set for further inspection.
 - *We assume this selects a set of questions (a collection of object identifiers)*
- The system returns full question material for each item of sub-set via QDB
 - *the returned information is highly dependent on completeness and quality of the metadata*
 - *the information set below assumes that a question is only used by a single variable used in a single survey and stored in a single file. While this is the traditional archive view of metadata, it is not necessarily always true. The same question can be reused across survey questionnaires, the same variable can be stored in several files (like by region for a census) thereby leading to several frequency tables, etc. While the simple view is what will likely initially be made available by the metadata providers, down the road, end user applications will need to take into account that objects get reused and have the ability to display complex relationships.*
 - Structured displays of question texts in different national languages with response categories, variable values, and frequencies, allow for quickly assessing face validity of comparisons. For each question, the user is informed about its questionnaire context (filter/routing information is explicitly given with the question texts. As options, preceding and following questions can be retrieved on request; the user can follow a link to a PDF/image of the full questionnaire possibly held in an external



system) its study/survey context, i.e. the universe/coverage of the survey the question comes from. These context displays are mostly hidden in the application interface until the user explicitly requests them.

- Actor performs checks of comparability
→ *Visually within the application (like side by side comparison)*

The actor marks a list for further processing with actual conversion coding
→ *the conceptual grouping and comparability mechanisms are used to bring objects together*
→ *mappings can be used to recode categories*
→ *this new information is now stored in the QDB local repository (only new elements, existing ones are simply referenced).*
→ *A copy of the references element may be placed in the QDB cache repository to improve future metadata retrieval operations*
→ *Once the analysis is completed, it's resulting metadata can be registered and becomes available for reuse (in 3CDB, QDB, or any other CESSDA application)*

Use Case 2: Facilitate Optimization of Questionnaire Design

- The actor searches for a specific term
→ *User can query the registry looking for questions based on various constraints (metadata, text based search). Can optionally search thesaurus, across languages or translations (if captured).*
- The system finds all questions that contain:
 - the specified term
 - related terms via a thesaurus
 - related terms via translations
- The system returns a summary of all question texts
→ *The registry provides the question locations and they retrieve from their respective repositories*
- The actor selects a question for detailed inspection
The system returns detailed information about this question:
→ *the question metadata contains references to it's related objects or the registry can be searched for object referencing this question (like variables). The related metadata is retrieved from the repositories as needed. Like in use case 1, this can lead to complex visualization.*
 - original question
 - the questionnaire
 - the survey
 - the dataset
 - Desirable: basic statistical information (frequencies and means) on underlying data
- The actor decides to add the question to 'shopping cart'
- Export 'shopping cart'
→ *this is an application feature*
→ *the shopping cart can be implemented as a resource package containing collections of questions (possibly organized in groups). This can be stored in the QDB repository*
→ *new questionnaire flows using these questions can be designed and stored*
→ *When the project is completed, the results can be registered for reuse*



Use Case 3: Facilitate Searching for Data Sets Containing Relevant Data

→ *Similar to use case 2. The survey metadata may provide information on how to access the dataset, the ability to actually download the data is the responsibility of the provider*

- The actor searches for a specific term
- The system finds all questions that contain:
 - The term or (defined) related terms
 - Translations of the term
- The system returns a summary of all question texts
- The actor selects a question for detailed inspection
- The system returns detailed information about this question:
 - original question
 - the questionnaire
 - the survey
 - the dataset
 - Desirable: basic statistical information (frequencies and means) on underlying data
- The actor decides to download the dataset for further analysis

Other CESSDA Use Cases

Use case 1: Documenting and searching for multilingual wordings of the same question for the same study

Submitted by Tolis Linardis

(a) Documenting

The questions of the figure below, concern work orientation in the year 2005. The survey used here is the International Social Survey Programme (ISSP). The example has been taken from a paper of Granda P., Hadorn R., Wolf C., titled “Harmonizing Survey Data”. ISSP is ex ante input harmonized meaning that all three questions have been derived by the same source question. Nevertheless, there are meaning discrepancies between the questions.

UK	France	Germany
How satisfied are you in your main job?	Etes-vous satisfait ou insatisfait de votre emploi principal? (Entourer seulement un chiffre)	Wie zufrieden sind Sie im allgemeinen in Ihrem Beruf? Nur EINE Markierung möglich!
1. Completely Satisfied 2. Very Satisfied 3. Fairly Satisfied 4. Neither Satisfied nor dissatisfied 5. Fairly dissatisfied 6. Very dissatisfied 7. Completely dissatisfied 8. Can't choose	1. complètement satisfait 2. très satisfait 3. assez satisfait 4. ni satisfait, ni insatisfait 5. assez insatisfait 6. très insatisfait 7. complètement insatisfait 8. ne peut pas dire	1. Völlig zufrieden 2. Sehr zufrieden 3. Ziemlich zufrieden 4. Weder zufrieden noch unzufrieden 5. Ziemlich unzufrieden 6. Sehr unzufrieden 7. Völlig unzufrieden 8. Kann ich nicht sagen

According to DDI3, each question above is one question item with a unique identifier. The question item is composed of a “Literal text” and a “Codescheme”. The Codescheme is composed of “Category reference” and “values”.



Let's assume firstly that we would like to document the relationships between the three questions stored in three different repositories. The question items have already been registered at the registry. The arising documentation problems in such a case are:

- How shall we document the relationships between these 3 questions? Do we have to make relationships between UK-France, UK-Germany and France – Germany or since ISSP is ex ante input harmonized we should have documented the source question and make references between source and country – specific questions? Where are these references stored?

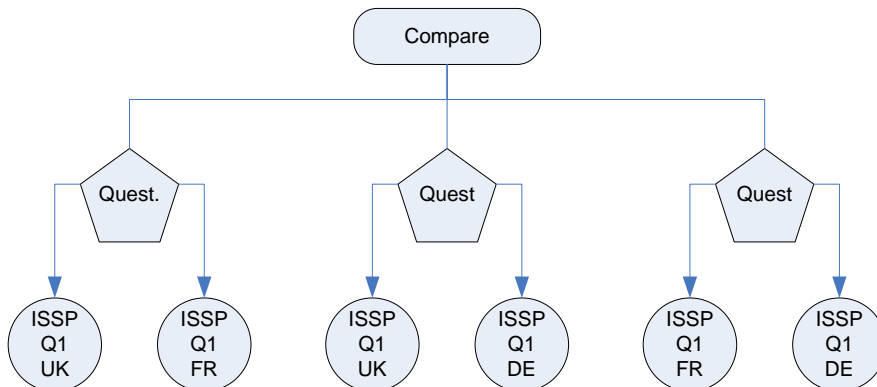
(b) Searching....

QDB provides interfaces for searching multilingual wordings of the same question for the same study. Search criteria: a) study title b) free text upon different structural elements of the question item

Use case 1: Implementation

From the conceptual model perspective, the 3 questions have indeed unique identifiers (URN's) that for the sake of simplicity we will abbreviate: "urn:ISSP_QuestSch_UK.Q1", "urn:ISSP_QuestSch_FR.Q1", "urn:ISSP_QuestSch_DE.Q1"

For documenting the relationship between these three questions, given that is done after the facts, we can use the comparability module. This currently works by creating pair wise comparisons so we could have the information capture as follows



Or in a "simplified" XML:

```
<Comparison>
  <Description>Harmonization of ISSP questionnaires. These questions have been derived by
the same source question. Nevertheless, there are meaning discrepancies between the
questions.</Description>
  <Questions>
    <Source>urn:ISSP_QuestSch_UK.Q1</Source>
    <Target>urn:ISSP_QuestSch_FR.Q1</Target>
    <Commonality>Similar but literal text differs slightly</Commonality>
    <CommonalityWeight>0.90</CommonalityWeight>
  </Questions>
  <Questions>
    <Source>urn:ISSP_ QuestSch_UK.Q1</Source>
    <Target>urn:ISSP_ QuestSch_DE.Q1</Target>
    <Commonality>Similar but literal text differs slightly</Commonality>
    <CommonalityWeight>0.95</CommonalityWeight>
  </Questions>
  <Questions>
    <Source>urn:ISSP_QuestSch_FR.Q1</Source>
```



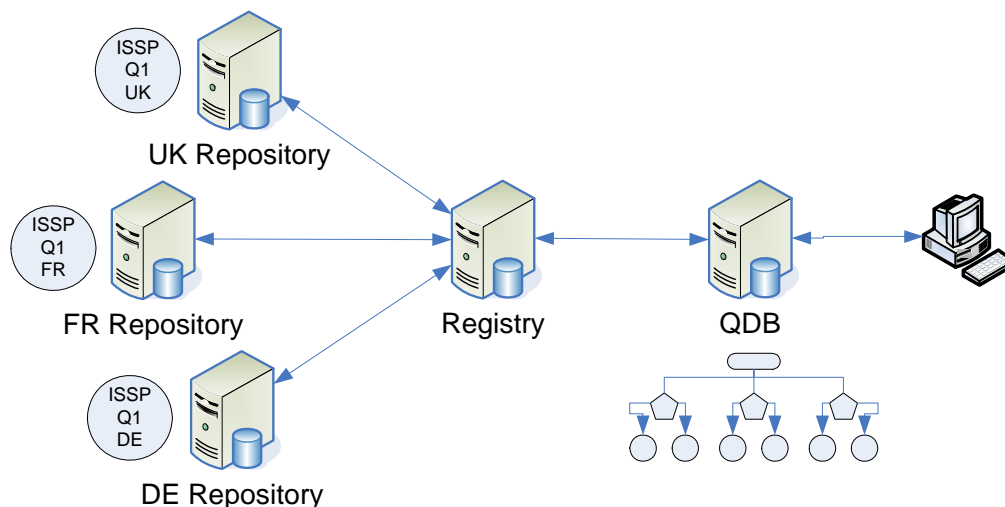
```
<Target>urn:ISSP_QuestSch_DE.Q1</Target>
<Commonality>Similar but literal text differs slightly</Commonality>
<CommonalityWeight>0.85</CommonalityWeight>
</Questions>
</Comparison>
```

Note that an alternate approach could be to use a simple grouping mechanism and for example create a new QuestionScheme that makes “references” to the three questions. This would allow capturing information for all of them at once but loses the comparability functionalities (and pair wise comparison is more accurate)

Storage

From the storage perspective, the three original questions are stored in their national repositories. Having been registered, their URNs / locations are known to the registry and their literal texts have been indexed. This means their content (and related variable, classification, concept, etc.) can be retrieved by performing a URN lookups and then querying the respective sources.

The comparison XML above is stored in the QDB (where the harmonization work is taking place). This is therefore where this new knowledge is being captured. As long as the work is unpublished, it can only be seen by the QDB users. Once the “comparison” work is completed, it can be registered and becomes accessible to the community. The source repositories could also be notified that new information relating to their studies has been published.



Information Retrieval Process

Whenever the end user application needs to retrieve all the information on these items, it:

- Calls the QDB `getComparison()` method to retrieve the base structure (assuming this comparison work belongs to a user)
- To resolve the references, the QDB then calls the registry `getObjectByURN()` method three times to lookup the source question locations. This returns three URLs to query the national repositories (note that these operations can be multi-threaded and executed in parallel)
- The QDB then calls the respective repositories to retrieve the full question XML (again, can be in simultaneous)
- Additional calls may then be placed to retrieve the questions' code and categories (likely through the question → variable path)



- The QDB then puts this information back together and returns it to the application that can display it as a whole to the end user. The whole process can be hidden behind a single application call.
- Note that in general, once these URNs have been resolved, the source XML will be placed in the QDB local metadata cache to alleviate the need to call the registry and back end repositories all the time. How long this is maintained in the cache is a question of implementation. Likewise, the registry itself will provide caching mechanisms

Gold Standard

Now from the “Gold Standard” perspective, the “master source question” should ideally down the road be represented as a single question with three questions texts (one for each language) pointing to a single harmonized code scheme itself making reference to a single category scheme with labels translated into the three languages. This harmonized classification will likely be produced through the 3CDB (and registered). So you could have something like (in simplified XML):

```
<QuestionScheme urn="urn:ISSP_QuestSch_GS">
  <Question id="Q1">
    <Literal xml:lang="en">How satisfied are you in your main job</Literal>
    <Literal xml:lang="fr">Etes vous satisfait de votre emploi principal</Literal>
    <Literal xml:lang="de">Wie zufrieden zind Sie im allgemeinen...</Literal>
  </Question>
  <CodeSchemeReference>urn:ISSP_CodSch1_GS</CodeSchemeReference>
</QuestionScheme>

<CodeScheme urn="urn:ISSP_CodSch1_GS">
  <Description xml:lang="en">Harmonized codes for job satisfaction</Description>
  <CategorySchemeReference>urn:ISSP_CatSch1_GS</CategorySchemeReference>
  <Code categoryId="C1" value="1"/>
  <Code categoryId="C2" value="2"/>
  <Code categoryId="C3" value="3"/>
  Etc...
</CodeScheme>

<CategoryScheme urn="urn:ISSP_CatSch1_GS">
  <Description xml:lang="en">Harmonized categories for job satisfaction</Description>
  <Description xml:lang="fr">...French description...</Description>
  <Description xml:lang="de">...German description...</Description>
  <Category id="C1">
    <Label xml:lang="en">Completely Satisfied </Label>
    <Label xml:lang="fr">Completement satisfait</Label>
    <Label xml:lang="de">Completely Satisfied </Label>
    <Definition xml:lang="en">...English definition...</Label>
    Etc...
  </Category>
  <Category id="C2">
    Etc...
  </Category>
</CategoryScheme>
```

You can then capture comparison information between the gold standard version and the original versions (using four pairs instead of three), thereby maintaining linkages regarding how the gold standard was established.

```
<Comparison>
  <Description xml:lang="en">Comparative information between Gold Standard questions and
past survey questions</Description>
  <Questions>
    <Source>urn:ISSP_Q1_GS.Q1</Source>
    <Target>urn:ISSP_Q1_UK.Q1</Target>
    <Commonality>Explain commonalities between Gold and EN version</Commonality>
    <Difference>Explain differences between Gold and EN version</Difference>
    <CommonalityWeight>0.90</CommonalityWeight>
  </Questions>
</Comparison>
```



```
</Questions>
<Questions>
  <Source>urn:ISSP_Q1_GS.Q1</Source>
  <Target>urn:ISSP_Q1_FR.Q1</Target>
  <Commonality>Explain commonalities between Gold and FR version</Commonality>
  <Difference>Explain differences between Gold and FR version</Difference>
  <CommonalityWeight>0.90</CommonalityWeight>
</Questions>
Etc...
</Comparison>
```

Searching

The text search on literal question text will be performed at the registry level. It is however likely the registry itself will rely on external services to perform multi-lingual or thesaurus based lookups as other CESSDA application will require similar functionalities and would therefore benefit from their availability as a service. In either case, the registry will perform a full text search based on a set of terms in the question text (this is an element indexed at the registry level). The list of questions will typically be a subset based on other criteria such as survey title, year, geography, concepts, etc.

For example, performing a search on questions containing the term “job” in the ISSP will trigger in the registry:

- A multi-lingual term lookup for “job” in other languages (this can be a local or remote service). It would for example return “emploi” in French and “Beruf” for German. Note that a multi-lingual thesaurus could also return “travail” in French.
- A query that creates a subset containing all questions in the ISSP survey
- multiple searches (one for each language) for “job” in questions with language set to “en”, “emploi” for “fr” and “Beruf” for “de”
- The aggregated results are then used to provide a list of URN that can then be looked up in repositories to collect the full question XML.

Note that multi-lingual searches can be very tricky to implement automatically and fail for fuzzy searches. For example, you can’t automatically translate “satisf*” into a French text search. It might sometimes be easier for users to perform multiple searches for different languages.

Use case 2: Documenting and searching for questions from different studies sharing the same concept (or the same universe)

Submitted by Tolis Linardis

Documenting.....

Let’s assume that there is already a concept registered at the registry called “Exposure to National TV News”. Each concept in DDI3 is composed of one identifier and one description. Two different question items from two different repositories reference the same concept (or the same universe or both the same concept and same universe). Where are the references between questions and concept stored?

Common Concept: Exposure to National TV news	
Question item 1	Question item 2
How many days in the PAST WEEK did you watch the NATIONAL network news on TV?	How many days in the PAST MONTH did you watch the NATIONAL network news on TV?



<ol style="list-style-type: none">1. None2. One day3. Two days4. Three days5. Four days6. Five days7. Six days8. Every day9. Don't know10. Refused	<ol style="list-style-type: none">1. 1-52. 6-103. 11-154. 16-205. 21-256. 26- end of month7. Don't know8. Refused
---	--

Searching...

QDB provides interfaces for searching questions via a common concept. Search criteria: a) full concept description coming from the registry b) free text search in concept description.

Use case 2: Implementation

This is a fairly simple case that follows the conceptual model relationships. There are two ways a question can be related to a concept:

- By pointing directly to the concept from the question itself
- By having a Variable simultaneously pointing to a Question and to a Concept

This can be seen in the conceptual model under Question Scheme – Core Model (p.35) and Variable Scheme (p.44). The first one is typical of question banks and the second common in legacy /archive metadata.

Storage

The relationship between these objects are described in their repositories and indexed by the registry (so it resides in the registry as well). Performing lookup queries such as “provide me a list of questions associated with this concept” or “provide me a list of questions associated with variables associated with this concept” are then straightforward for the registry.

Search

A full or free text search for concept(s) occurs in the registry and follows the same principles as text search for question text described in use case 1 above.

Gold Standard and Mappings

The challenges in this use case are actually more content related than technical:

- First it assumes that concepts have been documented at the variable or question level which is often not the case today in legacy metadata. Who will perform this task (it could be done by the archive, in the 3CDB or by a third party)?
- Second this will only work if either there is a gold standard list of concepts that all the questions or variables are using (at least within the scope of the collection of questions being queried) or if clear relationships exists between different concepts lists. In the later situation, such mappings across concepts lists can be described using the comparison module.

What is likely to happen is that some of the questions and variables may initially be associated with archive specific or national list of concepts. In some cases, the CESSDA concept list might have been used as well. Many questions will initially not be associated with concepts. An initial metadata enhancement effort will need to take place to harmonize the concept space. This will likely lead into a hybrid situation with a gold standard collection of concepts and an archive specific list with mappings.



In that case, there are basically two mechanisms that can be used to support a search:

- The user can select concepts across lists (for example using free text search)
- The registry can leverage on mappings to perform a smart search across concept lists using the degree of commonality and search depth as threshold criteria

A mapping example is illustrated below (in simplified XML).

```
<ConceptScheme urn="urn:ConceptSch_GS">
  <Description xml:lang="en">Harmonized concept scheme</Description>
  <Concept id="C1">
    <Label>Exposure to National TV news</Label>
  </Concept>
  Etc...
</ConceptScheme>

<ConceptScheme urn="urn:ConceptSch_Custom1">
  <Description xml:lang="en">Local concept list A</Description>
  <Concept id="C1">
    <Label>Watching Television</Label>
  </Concept>
  Etc...
</ConceptScheme>

<Comparison>
  <Description xml:lang="en">Mappings between Gold Standard and Custom Concept list</Description>
  <Concept>
    <Source>urn:ConceptSch_GS.C1</Source>
    <Target>urn: ConceptSch_Custom1.C1</Target>
    <Commonality>Both concepts cover watching TV</Commonality>
    <Difference>Gold Standard focuses on TV news only</Difference>
    <CommonalityWeight>0.85</CommonalityWeight>
  </Concept>
  </Questions>
  Etc...
</Comparison>
```

With the above knowledge at hand, the registry could perform a search on “national TV news” which would return the first concept from the Gold Standard. If we allow a commonality threshold of let’s say 80%. The second concept from a different concept space would also be found. This would then allow the lookup a list of questions associated with these two concepts with a measure of relevance. For example questions associated with the gold standard concept could have a score of 100% while others a score of 85% (the commonality weight). More complex scoring mechanisms and matches can of course be implemented.

Information Retrieval Process

To perform a lookup, an end user application could go directly to the registry or through the QDB or 3CDB (who would proxy the back end calls). The service would:

- Call the registry query() method with a statement like “find questions where concept label or description contains the terms national+TV+news within 80% relevance”
- The registry will then perform a full text search on the concept labels and description to find a match on the three keywords.
- A second search will then be performed to find related concepts with at least 80% commonality
- The result of both searches will be aggregated to form a list of concepts (score will be kept)
- The next step will be to locate the questions which will be performed in two steps
- The first will be a direct search for questions pointing to these concepts
- The second will be to find variables pointing to these concepts and also having a reference to a question



- Both query results will be aggregated (duplicates removed) and a relevant score will be assigned
- At this point, we only have the list of question identifiers and their location on the network. This will then be returned to the caller.
- The last step is then to retrieve the full question XML so calls need to be made to the repositories. If this is going through the 3CDB, QDB or another service, it could then itself collect the information on behalf of the end user application. Otherwise, the application can call the repositories getObjectByURN() method. The registry or QDB/3CDB caching mechanisms may also simplify this process through local lookups.

Note that the various pieces of information used by the registry to perform the search can come from anywhere on the network. For example: questions and variables metadata from repositories, concept list from a concept bank, and mapping information from the 3CDB. The query could also be expanded to multiple terms or languages using multi-lingual or thesaurus services.

Use case 3: Documenting and searching for questions from different studies sharing the same codescheme of a classification

Submitted by Tolis Linardis

Documenting....

The questions of this use case come from European Social Survey. Both of them use the same codescheme. This codescheme belongs to the structural elements of one classification. The codescheme used in ESS, concerning education level, is a modification of ISCED classification.

Question item 1	Question item 2
What is the highest level of education you have achieved? 1. Not completed primary (compulsory) education 2. Primary education or first stage of basic education 3. Lower level secondary education or second stage of basic education 4. Upper Secondary education 5. Post-secondary, non tertiary education 6. First stage of tertiary education (not leading directly to an advanced research qualification) 7. Second stage of tertiary education (not leading directly to an advanced research qualification) 8. Don't know	What is the highest level of education your husband/wife/partner has achieved? 1. Not completed primary (compulsory) education 2. Primary education or first stage of basic education 3. Lower level secondary education or second stage of basic education 4. Upper Secondary education 5. Post-secondary, non tertiary education 6. First stage of tertiary education (not leading directly to an advanced research qualification) 7. Second stage of tertiary education (not leading directly to an advanced research qualification) 8. Don't know

The two questions should be documented via DDI3 in a way that should reference the same code scheme of one classification.



Searching

QDB provides interfaces for searching questions via a common codescheme of a classification. Search criteria: a) title of the classification used b) free text search in codescheme and classification documentation.

Use case 3 implementation

This example is actually very similar to the previous one except that we are using categories instead of concepts as a basis. The implementation will therefore not be repeated. The important overlap here is actually the category scheme. The code scheme is essentially only a map between the question / variable and the category. In the particular example, a mapping between the ESS and the ISCED would also be useful to document to allow for searches across classifications

Use case 4: Automated comparison between questions

Submitted by Tolis Linardis

Documenting....

Let's assume that there is no available referencing documentation for questions is provided by the repositories. Is there any way to automatically detect similarities between questions?

1) Is there such a clever system to understand semantic similarities between multilingual or even monolingual questions when there is no appropriate documentation?	2) Is there any system to record similarities between monolingual questions based on common wordings , when there is no appropriate documentation?	3) Is there any system to record similarities between multilingual questions based on common wordings , when there is no appropriate documentation?
<p>If yes, what is the name of this software? What are the general principles this software is based on, so as to “understand” the similarities between questions? How should these similarities be documented in the system?</p> <p>If no, is there a possibility for such software to be implemented? If yes, which are the general principles this software is based on, so as to “understand” the similarities between questions? How should these similarities be documented in the system?</p>		

Searching

While searching for questions using different criteria (concept, universe, codescheme, or combination of these criteria) the system provides information on other related questions coming as outcome from the automated procedure.

Use Case 4: Discussion

Automatically inferring meaning out of free text is a challenging process even for the human brain and may be beyond today's information technology capacity. This issue is not specific to social science and is being looked at in the context of the semantic web and linguistics. A Google or Amazon search on terms such as “computational semantics”⁴, “computational

⁴ http://en.wikipedia.org/wiki/Computational_semantics



linguistics”⁵, “corpus linguistics”⁶ or “natural language processing”⁷ returns several interesting sources.

While this could certainly apply to social science metadata, this is unfortunately a question that we are unable to answer at this time. This would certainly be an interesting research topic for CESSDA. We have sent queries to other experts but have not received feedback at the time of this report. We will keep the CESSDA team informed of future findings.

Use Case 5: Display difference between questions

Submitted by Ken Miller

Overview

Here is a more specific use case based on the Jan-Mar QLFS over 5 years 2004 to 2008. In the QLFS variabledetails2008.pdf page 24 you will see that MARSTA replaces MARSTT and the categories for marital status have changed. There might be other changes in question text, question position, response rates, interviewer instructions.

Would the question bank be able to display these differences in the attached DDI2 records in maybe a centrally held specifically produced DDI3 instance?

Use Case 5: Analysis

The following table has been produced from the DDI 2 files provided using a simple XQuery (available on request) and the open source BaseX⁸ native XML database package.

Year	Variable	Question	Categories
2004	marstt	Are you single, that is never married, married and living with husband/wife, married and separated from husband/wife, divorced, or widowed?	1 = Single, never married 2 = Married, living with husband/wife 3 = Married, separated from husband/wife 4 = Divorced 5 = Widowed -9 = Does not apply -8 = No answer
2005	marstt	Are you single, that is never married, married and living with husband/wife, married and separated from husband/wife, divorced, or widowed?	1 = Single, never married 2 = Married, living with husband/wife 3 = Married, separated from husband/wife 4 = Divorced 5 = Widowed -9 = Does not apply -8 = No answer
2006	marsta	Are you single, that is never married, married and living with husband/wife, married and separated from husband/wife, divorced, widowed?	1 = Single, never married 2 = Married, living with husband/wife 3 = Married, separated from husband/wife 4 = Divorced 5 = Widowed 6 = Civil Partner 7 = Separated Civil Partner 8 = Former Civil Partner, legally dissolved 9 = Surviving Civil Partner, partner died -9 = Does not apply -8 = No answer
2007	marsta	Are you single, that is never married, married and living with husband/wife, a civil partner in a legally recognised civil partnership, married and separated from husband/wife, divorced	1 = Single, never married 2 = Married, living with husband/wife 3 = Married, separated from husband/wife

⁵ http://en.wikipedia.org/wiki/Computational_linguistics

⁶ http://en.wikipedia.org/wiki/Corpus_linguistics

⁷ http://en.wikipedia.org/wiki/Natural_language_processing

⁸ <http://www.inf.uni-konstanz.de/dbis/baseX/>



		or widowed, [Spontaneous only] in a legally recognised Civil Partnership and separated from his/her civil partner, [Spontaneous only] formerly a civil partner, the Civil Partnership now legally dissolved, [Spontaneous only] a surviving civil partner: his/her partner having since died?	4 = Divorced 5 = Widowed 6 = Civil Partner 7 = Separated Civil Partner 8 = Former Civil Partner, legally dissolved 9 = Surviving Civil Partner, partner died -9 = Does not apply -8 = No answer
2008	marsta	Are you single, that is never married, married and living with husband/wife, a civil partner in a legally recognised civil partnership, married and separated from husband/wife, divorced or widowed, [Spontaneous only] in a legally recognised Civil Partnership and separated from his/her civil partner, [Spontaneous only] formerly a civil partner, the Civil Partnership now legally dissolved, [Spontaneous only] a surviving civil partner: his/her partner having since died?	1 = Single, never married 2 = Married, living with husband/wife 3 = Married, separated from husband/wife 4 = Divorced 5 = Widowed 6 = Civil Partner 7 = Separated Civil Partner 8 = Former Civil Partner, legally dissolved 9 = Surviving Civil Partner, partner died -9 = Does not apply -8 = No answer

Implementation

This presents an interesting use case as out of the box, using DDI 2, it would be difficult to know that “marstt” and “marsta” are actually the same variables. The only overlap is that in 2006, the category changed but the question text remained the same but this is not a clear indicator. The one thing we could actually detect is a question change between 2006 and 2007 for the “marsta” variable.

One of the problems is that there is no possibility in DDI 2 of relating variables, questions or categories across surveys. Some trick could be used such as merging the multiple surveys into a single large DDI document, using groups with special identifiers, or tagging variables, but this would be implementation specific and no one else would be able to know about this besides the implementing agency (it's not formally part of the specification).

This not only means that an engine today would not be able to clearly answer the question but that additional knowledge would need to be provided in the metadata in order to meet the use case requirements.

In the proposed conceptual model, this can be resolved before publication by the archive or after registration in the 3CDB through harmonization of the classification. What we essentially have is a single classification that has been versioned. This can be defined using two versions of the code and category schemes as shown below. Notice the addition of the version number in the URNs (default is 1.0 if not specified, just made explicit here).

For the 2004/2005 variables, we have v1.0:

```
<CodeScheme urn="urn:CodSch_QLFS_MARST(1.0)">
  <Description xml:lang="en">QLFS Marital status codes (2004-2005)</Description>
  <CategorySchemeReference>urn:CatSch_QLFS_MARST(1.0)</CategorySchemeReference>
  <Code categoryId="C1" value="1"/>
  <Code categoryId="C2" value="2"/>
  <Code categoryId="C3" value="3"/>
  <Code categoryId="C4" value="4"/>
  <Code categoryId="C5" value="5"/>
  <Code categoryId="C-9" value="-9"/>
  <Code categoryId="C-8" value="-8"/>
</CodeScheme>

<CategoryScheme urn="urn:CatSch_QLFS_MARST(1.0)">
  <Description xml:lang="en">QLFS Marital status categories (2004-2005)</Description>
```



```
<Category id="C1">
  <Label xml:lang="en">Single, never married</Label>
</Category>
<Category id="C2">
  <Label xml:lang="en">Married, living with husband/wife</Label>
</Category>
<Category id="C3">
  <Label xml:lang="en"> Married, separated from husband/wife </Label>
</Category>
<Category id="C4">
  <Label xml:lang="en">Divorced</Label>
</Category>
<Category id="C5">
  <Label xml:lang="en">Widowed</Label>
</Category>
<Category id="C-9" missing="Y">
  <Label xml:lang="en">Does not apply</Label>
</Category>
<Category id="C-8" missing="Y">
  <Label xml:lang="en">No answer</Label>
</Category>
</CategoryScheme>
```

And we then create version 1.1 with the 4 new codes/categories from 2006:

For the 2004/2005 variables, we have v1.0:

```
<CodeScheme urn="urn:CodSch_QLFS_MARST(1.1)">
  <Description xml:lang="en">QLFS Marital status codes (2004-2005)</Description>
  <versionRationale>Added 4 new codes to classification</VersionRationale>
  <CategorySchemeReference>urn:CatSch_QLFS_MARST(1.1)</CategorySchemeReference>
  <Code categoryId="C1" value="1"/>
  <Code categoryId="C2" value="2"/>
  <Code categoryId="C3" value="3"/>
  <Code categoryId="C4" value="4"/>
  <Code categoryId="C5" value="5"/>
  <Code categoryId="C6" value="6"/>
  <Code categoryId="C7" value="7"/>
  <Code categoryId="C8" value="8"/>
  <Code categoryId="C9" value="9"/>
  <Code categoryId="C-9" value="-9"/>
  <Code categoryId="C-8" value="-8"/>
</CodeScheme>

<CategoryScheme urn="urn:CatSch_QLFS_MARST(1.1)">
  <Description xml:lang="en">QLFS Marital status categories (2004-2005)</Description>
  <versionRationale>Added 4 new categories to classification</VersionRationale>
  <Category id="C1">
    <Label xml:lang="en">Single, never married</Label>
  </Category>
  <Category id="C2">
    <Label xml:lang="en">Married, living with husband/wife</Label>
  </Category>
  <Category id="C3">
    <Label xml:lang="en"> Married, separated from husband/wife </Label>
  </Category>
  <Category id="C4">
    <Label xml:lang="en">Divorced</Label>
  </Category>
  <Category id="C5">
    <Label xml:lang="en">Widowed</Label>
  </Category>
  <Category id="C6">
    <Label xml:lang="en">Civil Partner</Label>
  </Category>
  <Category id="C7">
    <Label xml:lang="en">Separated Civil Partner</Label>
  </Category>
  <Category id="C8">
```



```
        <Label xml:lang="en">Former Civil Partner, legally dissolved</Label>
    </Category>
    <Category id="C9">
        <Label xml:lang="en">Surviving Civil Partner, partner died</Label>
    </Category>
    <Category id="C-9" missing="Y">
        <Label xml:lang="en">Does not apply</Label>
    </Category>
    <Category id="C-8" missing="Y">
        <Label xml:lang="en">No answer</Label>
    </Category>
</CategoryScheme>
```

A more advanced version of the above could also be created by including v1.0 by reference in v1.1 and simply adding the new elements. For example, for the codes:

```
<CodeScheme urn="urn:CodSch_QLFS_MARST(1.1)">
    <Description xml:lang="en">QLFS Marital status codes (2004-2005)</Description>
    <versionRationale>Added 4 new codes to classification</VersionRationale>
    <CategorySchemeReference>urn:CatSch_QLFS_MARST(1.1)</CategorySchemeReference>
    <CodeSchemeReference>urn:CodSch_QLFS_MARST(1.0)</CodeSchemeReference>
    <Code categoryId="C6" value="6"/>
    <Code categoryId="C7" value="7"/>
    <Code categoryId="C8" value="8"/>
    <Code categoryId="C9" value="9"/>
</CodeScheme>
```

Now that we have a uniquely defined versioned classification, the variables “marstt” and “marsta” can point to their relevant code scheme version (and indirectly the right category list). We can then detect that the commonality and that the categories have been versioned in 2006. Similar mechanisms can be used for the questions.

Use Case 6: Publishing a Nesstar Survey

To illustrate how a repository publication could work, here is an example from a hypothetical survey documented in DDI 2 using the Nesstar Publisher or IHSN Metadata Editor. Assuming a simple survey with a dataset is made up of 4 files, each contains 300 variables (a mix of scale and categorical). Variable level documentation includes universe, question text and interviewer instructions. Concepts have been captured in the study description.

- Assuming a CESSDA XML is based on the DDI3 model, the metadata is imported in the CESSDA Toolkit and broken into several components:
 - o 1 The Study Unit (docDscr + stdyDscr)
 - o 1 Logical Product (dataDscr)
 - o 4 Variable Schemes (one per file) also holding variable groups (fileDscr)
 - o Several Category and Code schemes containing categorical variables' code & labels (one per categorical variable)
 - o Question Schemes and Instruction Scheme (likely one per fileDscr)
 - o 1 or 2 Concept /and Universe Schemes (depending whether survey and variable level universes and concepts are merged)
 - o Given that DDI 2 does not provide string mechanisms to capture the questionnaire flow (only previous/next but this is not exposed in the Nesstar Publisher), a simple linear Control Structure Scheme can be created to associate the questions with
 - o 4 Logical Record (in LogicalProduct, one per file)
 - o 4 Physical Data Product (one per file) defining the file characteristics
 - o 4 Physical Date Instance (pointing to the actual data files). These can be ASCII or SPSS, Stata, SAS files. This is where the summary statistics (min, max, mean, frequencies, etc.) are stored
 - o If cubes are present in the DDI 2, they will generate various NCubePhysical DataProducts



- Various other materials can be generated if using the Nesstar external resource editor or the all-in-one IHSN Metadata Editor (RDF file)
- The CESSDA Publisher should then perform some initial integrity test to make sure that enough information is available to comply with the conceptual model requirements. The only required element in DDI 2 is the survey title. This is clearly insufficient in a metadata rich environment. The toolkit will also require an agency, survey ID and possibly other metadata elements. These can be extracted from the DDI metadata if available or taken from local application preferences
- At this stage the user has the option to store the information “as is” in the repository but this would not be taking advantage of the reusability features of the conceptual model
- Once the initial metadata has been validated, various optimization steps can take place:
 - Code and categories used by more than one variable can be merged into a single scheme
 - Questions and Instructions reused by more than one variables can be aggregated
 - Concepts and universes can likewise be aggregated (if applicable)
 - Variables used in multiple files could also be aggregated into a common variable scheme and reused by reference
 - Etc.
- These metadata import / optimization / curation procedures should be accompanied with relevant quality assurance procedures (such as metadata reports) to facilitate the process
- At any time, the various objects can be saved and uploaded into the repository for storage. Note that all of the above metadata is under the umbrella of a StudyUnit so it remains a coherent package (no loose objects)
- Once the optimization and quality assurance processes are completed, the various metadata elements can be registered and become searchable and retrievable by CESSDA applications. They remain part of the original study but can be searched at the “Bank” level (variables, questions, classifications, etc.)
- Note that this entire process can potentially be automated or semi-automated through batch processing



Metadata Specifications

The following metadata specifications should be considered for integration in the implementation of the QDB and related system architecture.

Social science / Statistics

Data Documentation Initiative DDI 2 / 3

<http://www.ddialliance.org>

This is the core specification for microdata that the CESSDA community is likely to align with.

Two flavors of DDI currently co-exist:

- The widely used versions 1.x and 2.x (commonly referred to as DDI 2) based on the “codebook” view of a single archived instance of a survey.
- The recently released version 3.0 that broadened the scope of DDI to the entire survey life cycle, is based on a modern schema design that maximizes metadata reusability.

The CESSDA conceptual model proposed in this document is based on the DDI 3 and SDMX designs to meet the complex requirements of the community and ensure compliance with the DDI specification.

Statistical Data and Metadata Exchange (SDMX) Standard 2.0

<http://www.sdmx.org>

SDMX Technical Standards Version 2.0 provides the technical specifications for the exchange of data and metadata based on a common information model. The scope of this SDMX is to define formats for the exchange of aggregated statistical data and the metadata needed to understand how the data is structured. The major focus is on data presented as time series, although cross-sectional XML formats are also supported.

DDI 3.0 and SDMX have been designed to work together and complement each other. They are based on a similar technical design, have direct mappings at the level of aggregated data (DDI NCube and SDMX objects), and have the ability to share metadata structure for concepts and classifications.

Microdata can be seen as a product for the expert statistician or researcher that gets ultimately consumed by the general public, policy maker, economist or and others in it's easier to understand tabulated or aggregated form. Combining SDMX and DDI provides the ability to maintain the linkages between low level microdata and time series indicators.

Dublin Core

<http://www.dublincore.org>

Dublin core provides a set of fields for providing the citations of resources, and has a core set and an extension mechanism, expressed in XML. This standard has been integrated in DDI 3.0 and is also commonly used by DDI 2 users to document survey related publications, papers and other resources. Its RDF representation is also a fundamental component of the semantic web.

ISO/IEC 11179

<http://metadata-stds.org/11179/index.html>



This standard provides a model for understanding what it terms “data elements” which are as applicable to metadata as they are to data. The model provided gives a standard way of defining terms, the concepts they represent, the value domains which they encompass, and how those value domains are represented. Additionally, a model for lifecycle management is provided. Ultimately, this is a powerful model for defining the semantics of different terms and concepts used with social sciences data.

The DDI 3 model for managing concepts has been designed with ISO/IEC 11179 compliance in mind.

ISO 19115

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=26020

This standard provides a model for defining geographies, and is used by many other systems which care about geography, maps, etc. This model is embedded in DDI 3, for example, but is widely used.

Metadata Encoding and Transmission Standard (METS)

<http://www.loc.gov/standards/mets/>

This is a standard from the world of digital archives, which provides for the packaging of a set of related objects (e.g., a Web page and the image files it references). It allows for other standard metadata format to be embedded in it. The DDI community is currently in contact with METS for an official integration of DDI 3 into this standard.

PREMIS

<http://www.loc.gov/standards/premis/>

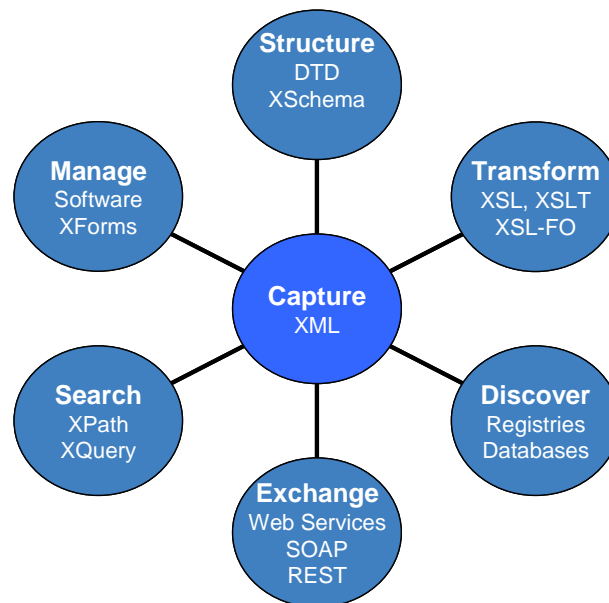
This is an XML format for expressing metadata about the archival lifecycle, and is meant to be used in combination with the OAI archival reference model.

Technology

The following standards are broadly used by the IT industry and are relevant to the CESSDA architecture.

XML

XML is a language but also a collection of standard and technologies. The specifications listed below play a key role in any XML driven architecture.



- The XML language itself (<http://www.w3.org/XML/>) defining the basic syntax and grammar to ensure an XML document is “well-formed”
- XSchema: used to describe the metadata structures and “validate” an XML document. <http://www.w3.org/XML/Schema>
- XPath / XQuery: an XML document can essentially behave as a database and XML come with various mechanisms to “query” XML data/metadata. The commonly used syntax is XPath 1.0 (<http://www.w3.org/TR/xpath>) but a much more powerful version has been available since early 2007 as XPath 2.0 (<http://www.w3.org/TR/xpath20/>) or XQuery 1.0 (<http://www.w3.org/TR/xquery/>). The latter is the recommended approach for working with complex metadata structure such as the proposed CESSDA conceptual model, DDI 3 or SDMX
- XSL transformations: XML often needs to be converted to other formats (HTML, PDF, RTF, Text, other XML, etc.) for presentation or processing purposes. The extensible stylesheet language family (<http://www.w3.org/Style/XSL/>) provides a set of specifications to describe such process. Two versions of the XSL transformation (XSLT) are available: version 1.0 (using XPath 1.0) and version 2.0 (<http://www.w3.org/TR/xslt20/>) (using XPath 2.0 / XQuery 1.0). Version 1.0 is widely used and can be processed natively by most computers and web browser. The more powerful and recommended version 2.0 typically requires specialized processors (such as Saxon). XSL also comes with an XML vocabulary for specifying formatting semantic, XSL-FO (<http://www.w3.org/TR/xsl/>), typically used to produce PDF outputs.
- Web services: a Service Oriented Architecture (SOA) implementation often relies on a set of standards such as the Web Service Description Language (WSDL, <http://www.w3.org/TR/wsdl>) and the Simple Object Access Protocol (SOAP, <http://www.w3.org/TR/soap/>). Another commonly used interface is the Representational State Transfer (REST, http://en.wikipedia.org/wiki/Representational_State_Transfer)
- Schematron: while not an official specification or standard, Schematron (<http://www.schematron.com>) is a useful framework for performing extended document validation that can be very useful for quality assurance purposes.



Security

As security will likely be an important requirement for the CESSDA environment, we recommend implementing authentication and authorization mechanisms based on the following specifications:

- OASIS Security Services: The Security Assertion Markup Language (SAML, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security), developed by the Security Services Technical Committee of OASIS, is an XML-based framework for communicating user authentication, entitlement, and attribute information. SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject (an entity that is often a human user) to other entities, such as a partner company or another enterprise application.
- OASIS eXtensible Access Control Markup Language (XACML, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml), a core XML schema for representing authorization and entitlement policies.



Technologies

The following tools, technologies or products could potentially be used for the implementation of the CESSDA QDB and related components. Most of them are available based on an open source licensing model. It is important to note though that open source products may not always scale up to the demand of large framework such as CESSDA and in some cases a commercial solution may be required to ensure quality of services.

Development

Java

<http://java.com/en/>

[http://en.wikipedia.org/wiki/Java_\(programming_language\)](http://en.wikipedia.org/wiki/Java_(programming_language))

Sun Java is the de facto standard for the development of cross platform solutions and is the recommended core technology for the implementation of the CESSDA QDB and related components. The language should be accompanied with a set of standard developer tools and guidelines:

- Eclipse (<http://www.eclipse.org/>) should be used as common Integrated Development Environment (IDE)
- All Java code should follow Sun's "Code Conventions for the Java Programming Language" (<http://java.sun.com/docs/codeconv/>)
- All source code must be commented using Javadoc (<http://java.sun.com/j2se/javadoc/>) syntax following Sun's recommendations (<http://java.sun.com/j2se/javadoc/writingdoccomments>)
- All Java classes should be accompanied by a JUnit (<http://www.junit.org/>) unit testing use cases
- Implementation of application logs should be using the Apache log4j library (<http://logging.apache.org/log4j/>)

In addition, common development guidelines should include recommendations on the maintenance of the source code in managed repositories based on the Concurrent Versioning System (CVS, http://en.wikipedia.org/wiki/Concurrent_Versions_System) or Subversion ([http://en.wikipedia.org/wiki/Subversion_\(software\)](http://en.wikipedia.org/wiki/Subversion_(software))). Such an environment can be provided using public repositories such a SourceForge (<http://sourceforge.net/>) or Google Code (<http://code.google.com/>), by hosting private repositories using solutions like G-Forge Advanced Server (<http://gforge.org/gf/>), or by subcontracting services to companies like CVSDude (<http://cvsdude.com/>) or Assembla (<http://www.assembla.com/search/home>).

A proper bug tracking tool must also be available to developers. If such framework is not readily available in the source code repository environment, popular tools include BugZilla (<http://www.bugzilla.org/>) and Mantis (<http://mantisbt.org/>)

In some cases it may be required to integrate Java based solutions with the Microsoft .NET environment. The following approaches can be taken to address this issue:

- The web service oriented architecture will make all services technology independent by providing standard based XML interfaces.
- Java libraries can be bridge to the .NET environment using open source or commercial packages such as IKVM (<http://www.ikvm.net/>) or JNBridge (<http://www.jnbridge.com/>)



The following articles also provide additional information on this topic:

- "Integrating Java* and Microsoft .NET**", Jon Jagger, Content Master Ltd., <http://softwarecommunity.intel.com/articles/eng/3105.htm>
- "Portable GUIs Improve Application Flexibility", John Sharp, Content Master Ltd, <http://softwarecommunity.intel.com/articles/eng/3039.htm>
- "Calling Java Classes Directly from .NET", Wayne Citrin, <http://www.devx.com/interop/Article/19945/1763>

Apache XML Beans

<http://xmlbeans.apache.org/>

XMLBeans is a tool that allows you to access the full power of XML in a Java friendly way. The idea is that you can take advantage of the richness and features of XML and XML Schema and have these features mapped as naturally as possible to the equivalent Java language and typing constructs. XMLBeans uses XML Schema to compile Java interfaces and classes that you can then use to access and modify XML instance data. Using XMLBeans is similar to using any other Java interface/class: you will see things like getFoo or setFoo just as you would expect when working with Java. While a major use of XMLBeans is to access your XML instance data with strongly typed Java classes there are also APIs that allow you access to the full XML infoset (XMLBeans keeps XML Infoset fidelity) as well as to allow you to reflect into the XML schema itself through an XML Schema Object model.

Saxon

<http://saxon.sourceforge.net/>

As pointed in the previous section, working with complex metadata structure such as the proposed CESSDA conceptual model, DDI 3, or SDMX is greatly simplified by using the XSLT 2.0, XPath 2.0 or XQuery 1.0. These technologies are often not supported natively by most development platform or computers and require a specialized processor. Saxon, available for both the Java and .Net environments, is our recommended solution.

Saxon comes in two packages: Saxon-B implements the "basic" conformance level for XSLT 2.0 and XQuery, while Saxon-SA is a schema-aware XSLT and XQuery processor. Both packages are available on both platforms (Java and .NET). Saxon-B is an open source product available from this site; Saxon-SA is a commercial product available from Saxonica Limited.

The most obvious difference between Saxon-SA and Saxon-B is that Saxon-SA is schema-aware: it allows stylesheets and queries to import an XML Schema, to validate input and output trees against a schema, and to select elements and attributes based on their schema-defined type. Saxon-SA also incorporates a free-standing XML Schema validator. In addition Saxon-SA incorporates some advanced extensions and optimizations not available in the Saxon-B product

In most cases, the open source version is all that is needed. CESSDA may need to consider the commercial version for performance reasons or if schema validation is a requirement.

VDT-XML

<http://vtd-xml.sourceforge.net/>

Processing very large XML documents using standard SAX or DOM parsers can often be a memory or resource intensive operation. The open source VTD-XML is a suite of innovative



XML processing technologies centered on a non-extractive XML parsing technique that provides high performance processing of large XML documents. The library is available for n C, C# and Java. While VDT-XML does not implement version 2.0 of XPath, it can in some cases be the right tool for bulk parsing, extraction or transformation of large XML instances.

Lucene / Compass

<http://lucene.apache.org/>

Free text search is a fundamental requirement of the CESSDA infrastructure. While most database systems come with some built-in functionality to perform such operations, they often lack the flexibility or performance of dedicated search engines. Apache Lucene is a widely used open source high-performance, full-featured text search engine library written entirely in Java that we recommend for the CESSDA QDB.

To facilitate the integration of into Java applications, the open source Compass provides a simple API for working with Lucene

<http://www.compass-project.org/>

Sun XACML

<http://sunxacml.sourceforge.net/>

This is an open source implementation of the OASIS XACML standard, written in the Java programming language.

This project provides complete support for all the mandatory features of XACML as well as a number of optional features. Specifically, there is full support for parsing both policy and request/response documents, determining applicability of policies, and evaluating requests against policies. All of the standard attribute types, functions, and combining algorithms are supported, and there are APIs for adding new functionality as needed. There are also APIs for writing new retrieval mechanisms used for finding things like policies and attributes.

Server side

A variety of server side product will be required to support the CESSDA architecture. The following are commonly used solutions.

Apache HTTP

<http://httpd.apache.org/>

http://en.wikipedia.org/wiki/Apache_HTTP_Server

Delivering content to web browsers, any infrastructure requires a web server. Apache has been the most popular web server on the Internet since April 1996 and is available for all commonly used operating environments. While most windows servers come with Microsoft Internet Information Server, using the same products on all platform (and replacing IIS with Apache on Windows server) would facilitate the production of harmonize guidelines and configuration procedures across all CESSDA servers.

Tomcat / JSP

<http://tomcat.apache.org/>

http://en.wikipedia.org/wiki/Apache_Tomcat



CESSDA based web services and web sites will based on Java technologies will require a server side product that directly supports this platform.

Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

Apache Tomcat is developed in an open and participatory environment and released under the Apache Software License. Apache Tomcat is intended to be a collaboration of the best-of-breed developers from around the world. Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.

pHp

<http://www.php.net/>

<http://en.wikipedia.org/wiki/Php>

If for various reasons a Java based (JSP) platform cannot be used for delivering web content to users, pHp is the recommended alternative. pHp is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

pHp is available for commonly used platforms and easily integrates with web servers such a Apache and Microsoft Internet Information Server.

Hibernate

<http://www.hibernate.org/>

[http://en.wikipedia.org/wiki/Hibernate %28Java](http://en.wikipedia.org/wiki/Hibernate_%28Java)

Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API.

Unlike many other persistence solutions, Hibernate does not hide the power of SQL from you and guarantees that your investment in relational technology and knowledge is as valid as always. The LGPL open source license allows the use of Hibernate and NHibernate in open source and commercial projects.

Spring

<http://www.springsource.org/>

http://en.wikipedia.org/wiki/Spring_Framework

<http://www.springframework.net/>

The Spring Framework is an open source application framework for the Java platform and .NET Framework. Spring includes:

- The most complete lightweight container, providing centralized, automated configuration and wiring of your application objects. The container is non-invasive, capable of assembling a



complex system from a set of loosely-coupled components (POJOs) in a consistent and transparent fashion. The container brings agility and leverage, and improves application testability and scalability by allowing software components to be first developed and tested in isolation, then scaled up for deployment in any environment (J2SE or J2EE).

- A common abstraction layer for transaction management, allowing for pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Generic strategies for JTA and a single JDBC DataSource are included. In contrast to plain JTA or EJB CMT, Spring's transaction support is not tied to J2EE environments.
- A JDBC abstraction layer that offers a meaningful exception hierarchy (no more pulling vendor codes out of SQLException), simplifies error handling, and greatly reduces the amount of code you'll need to write. You'll never need to write another final block to use JDBC again. The JDBC-oriented exceptions comply with Spring's generic DAO exception hierarchy.
- Integration with Toplink, Hibernate, JDO, and iBATIS SQL Maps: in terms of resource holders, DAO implementation support, and transaction strategies. First-class Hibernate support with lots of IoC convenience features, addressing many typical Hibernate integration issues. All of these comply with Spring's generic transaction and DAO exception hierarchies.
- AOP functionality, fully integrated into Spring configuration management. You can AOP-enable any object managed by Spring, adding aspects such as declarative transaction management. With Spring, you can have declarative transaction management without EJB... even without JTA, if you're using a single database in Tomcat or another web container without JTA support.
- A flexible MVC web application framework, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. Note that a Spring middle tier can easily be combined with a web tier based on any other web MVC framework, like Struts, WebWork, or Tapestry.

You can use all of Spring's functionality in any J2EE server, and most of it also in non-managed environments. A central focus of Spring is to allow for reusable business and data access objects that are not tied to specific J2EE services. Such objects can be reused across J2EE environments (web or EJB), standalone applications, test environments, etc without any hassle.

Spring's layered architecture gives you a lot of flexibility. All its functionality builds on lower levels. So you can e.g. use the JavaBeans configuration management without using the MVC framework or AOP support. But if you use the web MVC framework or AOP support, you'll find they build on the core Spring configuration, so you can apply your knowledge about it immediately.

BlazeDS

<http://opensource.adobe.com/wiki/display/blazeds/BlazeDS/>
<http://en.wikipedia.org/wiki/BlazeDS>

If Adobe Flex / AIR is used for the delivery of web or desktop based Rich Internet Applications (RIA), Adobe BlazeDS is the server-based Java remoting and web messaging technology that enables developers to easily connect to back-end distributed data and push data in real-time to such applications for more responsive rich Internet application (RIA) experiences.

SOLR

<http://lucene.apache.org/solr/>



Solr is an open source enterprise search server based on the Lucene Java search library, with XML/HTTP and JSON APIs, hit highlighting, faceted search, caching, replication, and a web administration interface. It runs in a Java servlet container such as Tomcat.

Web Front-End

Nowadays, delivering web based content to users typically means going beyond the simple HTML based interfaces and providing smart user friendly interfaces or rich internet applications (http://en.wikipedia.org/wiki/Rich_Internet_application). Several technologies are available for such purposes (http://en.wikipedia.org/wiki/Category:Rich_Internet_application_frameworks) and, while no clear choice can be made, the following should help making a choice. We would suggest adopting both Adobe Flex and an AJAX based framework as options for CESSDA.

Adobe Flex/AIR

<http://www.adobe.com/products/flex/>
http://en.wikipedia.org/wiki/Adobe_Flex

Adobe Flex is a collection of technologies released by Adobe Systems for the development and deployment of cross-platform rich Internet applications based on the Adobe Flash platform. In February 2008, Adobe released the Flex 3 SDK under the open source Mozilla Public License. Adobe Flash Player, the runtime on which Flex applications are viewed, and Adobe Flex Builder, the IDE built on the open source Eclipse platform and used to build Flex applications, remain proprietary.

Flex is a highly productive, free open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems.

In addition to delivering web based solutions, desktop based application can also be implemented using the Adobe Integrated Runtime (AIR):

<http://www.adobe.com/products/air/>
http://en.wikipedia.org/wiki/Adobe_Integrated_Runtime

Note that while the Flex SDX is an open source product, professional developers typically require the commercial Adobe Flex Builder () to develop Flex/AIR applications.

AJAX-Based Framework

Ajax ([http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))), sometimes written as AJAX (shorthand for Asynchronous JavaScript + XML), is a group of interrelated web development techniques used to create interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.

An Ajax framework (http://en.wikipedia.org/wiki/Ajax_framework) is a framework that helps to develop web applications that use Ajax, a collection of technologies used to build dynamic web pages on the client side.

A wide selection of Ajax based framework is available which often makes the choice difficult to make (http://en.wikipedia.org/wiki/List_of_Ajax_frameworks and <http://ajaxian.com/resources>).



DoJo, JQuery and the Yahoo! UI, backed by a large community of users or a commercial company seem to be safe choices.

Others

The following RIA frameworks are mainly competing with Adobe Flex. While providing similar functionalities and performances, they have not reached the popularity of the Adobe

- Microsoft Silverlight (<http://silverlight.net> , <http://en.wikipedia.org/wiki/Silverlight>)
- Sun JavaFX (<http://java.sun.com/javafx/> , <http://en.wikipedia.org/wiki/JavaFX>)
- OpenLazlo (<http://www.openlaszlo.org/>, <http://en.wikipedia.org/wiki/OpenLaszlo>)

Databases

Storing metadata is a fundamental requirement of the CESSDA framework. This section examines various choices of open source and commercial solutions.

Open Source vs Commercial?

While CESSDA in general expresses a preference for open source solutions, the performance and scalability requirement of some core components may require the licensing of commercial products. Database is certainly one such area where we anticipate such need. In the particular case of the QDB and the underlying architecture, we suspect the metadata registry and very large repositories will require the licensing of commercial products.

Relational vs Native XML vs Hybrid?

Traditionally, data and metadata has been stored in relational database systems (RDBMS) (http://en.wikipedia.org/wiki/Relational_database_management_system). Relational database products have been available for decades and usually based on the Structured Query Language (SQL) (http://en.wikipedia.org/wiki/Structured_Query_Language). RDBMS require developers to initially create a set of tables and indexes to efficiently store the information and establish relationships to maintain referential integrity. This can be a complex process that typically needs to be repeated for each data structure.

When it comes to XML though, the information structure is actually well defined in the form of the XML schema and, as pointed out earlier, XML data source are natively searchable using the XPath and XQuery language. The growing popularity of XML as a representation format has therefore lead to the emerge of Native XML Databases (http://en.wikipedia.org/wiki/XML_database). These specialized databases are designed to efficiently store, process and query XML documents. They support the XQuery (<http://www.w3.org/XML/Query/>) language and many also implement the XML: DB API (<http://xmldb-org.sourceforge.net/>). One of the major advantages of native XML databases is that they do not require developers to pre-generate storage structures which significantly reduce the development effort and maintenance burden. A native XML database typically simply directly ingests XML instances and automatically generates the necessary structures. The ability to use XQuery greatly facilitates the search and retrieval operations on XML metadata. Initial implementations of native XML database typically lacked the ability to perform efficient text based searches. To alleviate this issue, some of them incorporated proprietary extensions or relied on external products such as Apache Lucene. Recognizing this as a fundamental need, the W3C released last year of the Xpath / XQuery Full Text specification (<http://www.w3.org/TR/xquery-full-text/>) that now provide an official syntax for text based search.



When it comes to managing XML and non-XML metadata for CESSDA, a choice would therefore need to be taken as to which approach to take. A solution could be built purely on a relational system (in which case the entire conceptual model will need to be expressed in a relational format), a purely XML system (in which case non-model based metadata should be serialized in XML) or as a hybrid system that mixes both. The latter is probably the best option as it supports both XML and non-XML based metadata and could also partially store XML metadata in a hierarchical format (for practical or performance reasons). A hybrid solution can be implemented in two different ways: by using different products (on relational DB and one native XML DB) or by adopting hybrid packages that support both. The last option is the recommended choice as it also reduces the need to support multiple products.

The section below provides a list of various open source and commercial packages that could be used by CESSDA. This is not exhaustive and other similar products may be available. Given that XML DB products have not been around for very long, little information is available today on their performances and how they compare to each other. It is clear that if such an approach is used by CESSDA, the product will need to be able to ingest and manage large XML collections and efficiently perform value based queries and free text searches. We would therefore recommend, if such information does not become available, to perform a product evaluation study before making a decision. In the all in one hybrid category, IBM DB 2 seems to provide a good option with both an enterprise and free version available and Oracle is of course a strong choice. MySQL remains an excellent choice as a relational database product (but little support for XML). The recent acquisition of Sun Microsystems by Oracle may impact the future of this product (positively or negatively). When it comes to native XML products, a choice is difficult to make without further performance evaluation.

Database Products

MySQL (Relational / OSS)

<http://mysql.com/>

<http://en.wikipedia.org/wiki/MySQL>

MySQL is a relational database management system (RDBMS) which has more than 11 million installations. The program runs as a server providing multi-user access to a number of databases. MySQL is owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now a subsidiary of Sun Microsystems, which holds the copyright to most of the codebase. The project's source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements.

MySQL is our recommended solution as RDBMS for standard CESSDA components.

Oracle (Hybrid / Commercial)

<http://www.oracle.com/>

<http://www.oracle.com/technology>

<http://www.oracle.com/technology/tech/xml/xquery/index.html>

When high performance and high scalability is required and open source solutions don't meet the enterprise demand. In such cases, the industry leading Oracle platform is likely the best choice but comes at a considerable price. In addition to its database products, the platform offer additional middleware and applications solutions that facilitate the management of XML objects or the implementation of web services.



Since version 10g release2, Oracle has introduced in its product line a full-featured and high performance database-native XQuery engine. This native XQuery engine empowers organizations with a versatile tool for querying, aggregating, and transforming the rapidly growing volume of XML data from diverse data sources. Version 11g has also introduced new features that make the platform the commercial solution of choice for both relational and native XML database.

IBM DB2 (Hybrid / OSS & Commercial)

<http://www-01.ibm.com/software/data/db2/>

<http://www-01.ibm.com/software/data/db2/xml/>

DB2 offers industry leading performance, scale, and reliability on your choice of platform from Linux to z/OS. IBM offers 4 versions of DB2:

- Enterprise Server Edition: The ideal data server for the most demanding workloads
- Express Edition: The ideal entry level data server that is suitable for transaction processing
- Express-C: A full-function no-charge data server with optional subscription
- Workgroup Server Edition: The ideal data server for deployment in medium-sized business environment

IBM DB2 Express-C (<http://www-01.ibm.com/software/data/db2/express/>) is a full-function relational and XML data server no-charge community edition of the DB2 data server. It is ideal for small businesses and multi-branch companies, as well as developers and business partners who serve these clients. DB2 Express-C can be setup quickly, is easy-to-use, and includes self-managing capabilities. It also embodies all the core features of the more scalable DB2 editions, including the revolutionary pure XML technology for powering a new breed of Web 2.0 and SOA based solutions.

Other Database products

Other products of interest include:

- PostgreSQL (<http://en.wikipedia.org/wiki/PostgreSQL>) [Relational]
- SQLite (<http://en.wikipedia.org/wiki/SQLite>) [Relational]
- Microsoft SQL Server (<http://www.microsoft.com/sqlserver>) [Hybrid?]
- Sybase (<http://www.sybase.com/>) [Relational, Hybrid with commercial extension]
- eXist (<http://exist.sourceforge.net/>) [XML]
- Oracle Berkeley DB Product Family (<http://www.oracle.com/technology/products/berkeley-db/>) [Hybrid]
- XHive (<http://www.x-hive.com/>) [XML]
- Tamino (<http://www.softwareag.com/Corporate/products/wm/tamino>). [XML]
- BaseX (<http://www.inf.uni-konstanz.de/dbis/basex/>)
- Several additional native XML database products are listed at Wikipedia (http://en.wikipedia.org/wiki/XML_database)

Other Products

Shibboleth

<http://shibboleth.internet2.edu/>



The Shibboleth® System is a standards based, open source software package for web single sign-on across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner.

The Shibboleth software implements widely used federated identity standards, principally OASIS' Security Assertion Markup Language (SAML), to provide a federated single sign-on and attribute exchange framework. Shibboleth also provides extended privacy functionality allowing the browser user and their home site to control the attributes released to each application. Using Shibboleth-enabled access simplifies management of identity and permissions for organizations supporting users and applications. Shibboleth is developed in an open and participatory environment, is freely available, and is released under the Apache Software License.

iRODS

<https://www.irods.org>

<http://www.dicerresearch.org>

CESSDA archives often need to store and maintain very large collections of data and other electronic files. Managing such systems can be challenging and often requires a complex system to keep track of the resources location, ensuring their safe preservation (backup, mirroring), controlling access or provide efficient delivery mechanisms. In addition, standard file management systems also rarely provide the necessary functionalities to associate metadata with the stored resources, a feature that would be highly beneficial when it comes to maintaining referential integrity between metadata and underlying resources.

Such a challenge is not unique to social sciences and is commonly faced in other domains such as hard sciences. The grid community has been looking into these issues for many years and software that could be of interest to CESSDA is iRODS, a rule based virtual file system that build on the Storage Resource Broker (SRB) (http://en.wikipedia.org/wiki/Storage_Resource_Broker). SRB is a data grid middleware software system operating in many national and international computational science research projects as a data grid, a digital library, persistent archive, and/or distributed file system.

iRODS™, the Integrated Rule-Oriented Data System, is a data grid software system developed by the Data Intensive Cyber Environments (DICE) group (developers of the SRB, the Storage Resource Broker), and collaborators. The iRODS system is based on expertise gained through nearly a decade of applying the SRB technology in support of Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems. iRODS management policies (sets of assertions these communities make about their digital collections) are characterized in iRODS Rules and state information. At the iRODS core, a Rule Engine interprets the Rules to decide how the system is to respond to various requests and conditions. iRODS is open source under a BSD license.

While not directly related to metadata or the QBD, we think this is a solution the CESSDA community might benefit from. A quick introduction is available at https://www.irods.org/pubs/iRODS_Overview_0903.pdf.



Implementation Issues

A fully fledged implementation of the architecture as described in this document is likely to be a multi-year endeavour that will require significant development efforts and resources. Investing in such an enterprise is therefore a decision that needs to be carefully made. Given the many dependencies and design options, a single strategy cannot be outlined today. We would recommend starting with prototype implementations that demonstrate the functionalities and benefits of the architecture as proof of concept.

This section discusses some of the challenges that will need to be addressed in order to draft a project implementation plan and examines some of the actions that could facilitate prototyping or supporting an incremental design approach.

Metadata

Conceptual Model

Knowing how the metadata will generically be represented across the system is a fundamental need for planning the development phases and for metadata preparation purposes. The conceptual model presented in this document focuses on the QDB and will need to be expanded in order to cover the broader collection of metadata elements.

The DDI 3 model should be used as a starting point as it essentially meets most of the current requirements. This also ensures compliance with the specification and facilitates the ingestion of DDI 2 based documents. The initial implementation could simply be based on the existing model, with adjustments introduced if such a need arises. If light changes are required, they could be submitted to the DDI Alliance for integration in the specification. In some cases however, this may not be possible and the CESSDA model could be implemented as extensions. This may particularly be true for the management of workflow metadata. Drawing from the SDMX or other models should also be useful.

Completing the conceptual model to support at least the 3CDB and QDB should be one of the first activities of the project.

Availability of Information

Given that the initial 3CDB and QDB applications seem to require the full collection of legacy metadata (for search/mining purposes) for reference purposes, one of the initial challenges will actually not be technical in nature:

- Searching for questions, classification or and other elements using concepts implies that such information has properly been documented in the surveys
- Exploring questionnaire structure supposes that flows have been included in the metadata
- Queries based on variable level information such as code/categories, questions text or requires documentation at the file/variable level
- Etc.

A significant amount of effort may therefore initially be required to ensure that the QDB /3CDB applications can get access to a comprehensive and consistent set of the source survey metadata. For example, the concept based search seems to be a fundamental aspect of this system but concept is commonly one of the least documented features in archives, particularly at the variable level. Can the QDB/3CDB start operating without such information available?



An initial step will therefore be to determine the minimal source metadata requirements for the 3CDB and QDB applications. This can then drive legacy metadata improvement activities, in parallel to the developments, to support the initial versions of the QDB or 3CDB. This could be undertaken by pilot archives that could then share their experience and potentially draft guidelines for others to use. Such effort would also lead to the creation of an initial set of utilities or enhancements to existing systems.

Repositories

Implementation of metadata repositories will be required from the early stage of the project. The conceptual model and the set of interfaces that a repository needs to expose (ingestion, search & retrieval and admin) are essential components to have in order for this to get this started. These are partially available today through this document and the DDI 3 information model (assuming it gets adopted for CESSDA). It will however need to be refined to provide more granular functionalities.

Defining repository search specifications will need to be examined for each of the storage banks but this should be fairly straightforward as repositories are not expected to support complex queries (this is the role of the registry or QDB/3CDB). Drafting repository API could therefore rapidly be completed once the model is stabilized and the set of repository banks well defined. Also, not all storage banks may be required from day one which could facilitate a gradual implementation. One aspect that could introduce some level of complexity in this system is metadata security. This however may not be necessary at the early implementation stages.

The development of a full fledge generic repository server can however be substantial work and one way to rapidly prototype the architecture would be to emulate a repository by building a service stack on top of a legacy system. The most obvious candidate for this is a Nesstar server. Being based on DDI 2 and on a non-service oriented architecture however, this may limit the model features (metadata reusability, cross-survey groupings, concept banks, etc.) and it would likely require enhancement to the underlying metadata. It however seems a conceivable approach. The bulk of the work would be to essentially hide the RDF based object model and search/retrieval functionalities behind the Repository API. A similar approach for example was used at the World Bank to implement cross-survey tabulations on harmonized datasets. A feasibility study would probably be helpful to assess the amount of effort this would represent. Note that the same principle could also be used with other non-Nesstar CESSDA archives.

Such prototype repositories could then be used to perform initial testing, support the implementation of the registry component, and demonstrate the functionalities to stakeholders. It would leave time to implement the actual repository system, likely in a multi-stage process, starting with basic repository, with essential banks, and scaling up to full / high performance implementation (year 2-3). Metadata publication tools will need to be developed in parallel.

Registry

The registry is at the heart of the system and is likely the most complex component. A fully functional high performance implementation is an ambitious objective. This however will not be required from the start and can be achieved incrementally.

An interesting approach that could initially be taken for prototyping purposes would be to use an SDMX registry. SDMX is actually very generic and can be used to manage non-SDMX documents using Metadata Reports. SDMX features such as registration services, workflows or



query mechanisms would then immediately become available to provide the initial registry services for the 3CDB or QDB. The CESSDA web services API could be implemented in front of the SDMX web services to expose from the beginning the right set of interfaces to the repositories, 3CDB and QDB applications. Two SDMX registry implementations are available today: one from Eurostat and one from Metadata Technology, both as open source products. One feature that is typically not required for SDMX but is more essential for CESSDA is full text search. Extensions to the existing registries may therefore be required to support such features.

As an alternate or second stage approach, the Canadian Research Data Centres Network recently initiated a development project aiming at integrating a DDI 3 driven metadata framework across their networked centers for the management of the surveys and research processes. The resulting products will be open source and will include a light DDI registry component. While this registry is not expected to implement all the features required by CESSDA, it could potentially be used as a starting point which could save considerable development work. The availability of the initial version of this registry is expected in the first-half of 2010 with the final release in 2011. Joint developments could be discussed with the Canada RDC Network.

Whether working in collaboration with others or independently, a CESSDA registry will need at some point to be implemented. An incremental approach that starts with basic registration and search services and gradually adds features such as complex text searches, multilingual support, and thesaurus should be taken to ensure that services become operational rapidly and support the development of the QDB and 3CDB. Integration of scalable technologies and support for large metadata collection should be planned from day one but can also be integrated in later stages of the project. This should also allow for other CESSDA services (such as multilingual thesaurus) to be implemented and become available as external components. Like for repositories, security can become a considerable bottleneck in such a system and requirements will have to be carefully planned.

QDB

The QDB is essentially a new product that needs to be developed from scratch. As it is essentially a specialized form of a repository, it is likely the right place to initiate the development of the generic repository architecture. The database design (whether relational, native XML or both) will only initially need to support the pieces of the conceptual model required by the QDB (questions, instrument, grouping and comparability). The first version should provide simple search functions and the ability to harmonize questions / instruments along with the relevant API to allow for the development of the QDB end user applications. Once this is completed, the admin and local cache components can be implemented. The advanced search capabilities will become available to the user as they become integrated in the QDB or back-end registry.

To function properly, the QDB will however need access to a registry and to external repositories from where reference metadata can be retrieved. This could be initially simulated until prototypes or other implementations become available.

3CDB

The 3CDB is a different product and outside the scope of this document. We recommend however for the technologies and the development model to be aligned on the QDB.



Infrastructure

In addition to product implementation and metadata enhancements, the right architecture to support the various services will need to come online. Such system deployment will therefore need to be planned side by side with the development activities. The initial environment is likely to follow a traditional approach (set of networked physical servers). Down the road however, CESSDA should anticipate the need for a high performance architecture that takes advantage of emerging virtualization and possibly cloud technologies.



Conclusions

The proposed architecture has many benefits – it leverages the current generation of proven technologies, which are designed to provide a sound basis for highly distributed and modular applications like the European Question Bank. Further, it provides a high degree of flexibility, both in terms of how services are combined, to provide functionality, and in terms of how archives can be integrated into the system. This flexibility promises easier maintenance and lower cost of maintenance in the long term.

The challenge of deploying this type of application is significant – different organizations have different types of legacy systems, and the task of maintaining the many types of metadata resources around a network of archives is considerable. The flexible, modular, service-based architecture proposed here will meet this challenge, allowing for a deliberate start-up, and incremental, phased deployments.

Further, the question bank is only one of the many types of applications which can fit into this overall architectural framework. An obvious next step is the 3CDB, but many different types of applications can be built, based on the modular services approach described here.

This document has provided a wealth of technical detail regarding the underlying models and recommended technologies, but the use cases show how these details meet the business requirements of the question bank. Although the architecture may seem complex, implementations of these technologies in other domains have proven this type of architecture actually simplifies the creation and maintenance of the system over time. Web services architectures also place a minimum requirement for interacting with any single part of the overall system, so that implementers may find that it is easier, rather than harder, to integrate applications and databases.

Overall, it is felt that the European Question Bank is an excellent candidate for this type of flexible, distributed architecture. Both the architectural framework described here, and the specific architecture for the European Question Bank, share this approach, which is felt to be the best one for this application.