



Guidelines and recommendations for integrating clinical data sources in the ACGT platform

Project Number: FP6-2005-IST-026996
Deliverable id: D 5.7
Deliverable name: Guidelines and recommendations for integrating clinical data
sources in the ACGT platform
Submission Date: 11/11/2009

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	ACGT
Project Full Name:	Advancing Clinico-Genomic Clinical Trials on Cancer: Open Grid Services for improving Medical Knowledge Discovery
Document id:	D 5.7
Document name:	Guidelines and recommendations for integrating clinical data sources in the ACGT platform
Document type (PU, INT, RE)	PU
Version:	0.40
Submission date:	11/11/2009
Editor: Organisation: Email:	Erwin Bonsma Philips Research erwin.bonsma@philips.com

Document type PU = public, INT = internal, RE = restricted

ABSTRACT:

This deliverable provides guidelines and recommendations for integrating clinical data sources into the ACGT platform. It considers two aspects. First it reviews available Open Source database solutions for storing the different types of clinical trial data after anonymisation. Next it describes how this data can be made accessible inside the ACGT platform by way of the data access services. Detailed instructions are provided for each data access service. The report closes with a discussion of our experiences so far, which illustrates what is possible but also describes the open issues.

KEYWORD LIST: clinical trials, databases, data integration, data access services

MODIFICATION CONTROL			
Version	Date	Status	Author
0.10	18/06/2009	Draft	E. Bonsma
0.20	25/06/2009	Draft – For internal review	E. Bonsma
0.30	10/07/2009	Draft – Updated after review	E. Bonsma
0.4	11/11/2009	Final – For review	J. van Leeuwen

List of Contributors

- Erwin Bonsma, Philips Research
- Jasper van Leeuwen, Philips Research
- Anca Bucur, Philips Research
- Mattias Ohlson, Lund University

Contents

1	INTRODUCTION	5
1.1	SCOPE	5
1.2	STRUCTURE.....	5
2	DATABASES	6
2.1	RELATIONAL DATA.....	6
2.1.1	<i>Clinical trial management systems</i>	6
2.2	DICOM IMAGE DATA	7
2.3	MICROARRAY DATA	8
3	DATA ACCESS SERVICES IN ACGT	11
3.1	FUNCTIONAL REQUIREMENTS.....	11
3.2	FUNCTIONALITY	11
3.3	SUPPORTED DATA SOURCES.....	12
4	INTEGRATING NEW DATA SOURCES	13
4.1	DATA ACCESS SERVICE FOR RELATIONAL DATABASES	14
4.1.1	<i>Functionality</i>	14
4.1.2	<i>Creating the service</i>	14
4.2	DATA ACCESS SERVICE FOR DICOM IMAGE DATA	22
4.2.1	<i>Functionality</i>	22
4.2.2	<i>Creating the service</i>	22
4.3	DATA ACCESS SERVICE FOR MICROARRAY DATA	23
4.3.1	<i>Functionality</i>	24
4.3.2	<i>Creating the service</i>	24
5	DISCUSSION	26
5.1	DATABASES	26
5.1.1	<i>Relational databases</i>	26
5.1.2	<i>DICOM databases</i>	26
5.1.3	<i>BASE database</i>	27
5.2	DATA ACCESS SERVICES	27
5.2.1	<i>Data access service for relational databases</i>	27
5.2.2	<i>Data access service for DICOM image data</i>	34
5.2.3	<i>Data access service for microarray data</i>	34
5.2.4	<i>Dynamic deployment of data access services</i>	34
5.2.5	<i>Open issues</i>	35
6	REFERENCES	37
	<i>Appendix 1 - Abbreviations and acronyms</i>	40
	<i>Appendix 2A – Initial example D2RQ mapping</i>	42
	<i>Appendix 2B – Final example D2RQ Mapping</i>	45
	<i>Appendix 3A – SQL query produced for initial Obtima SPARQL query</i>	48
	<i>Appendix 3B - SQL query produced for refined Obtima SPARQL query</i>	50

1 Introduction

One of the main challenges in carrying out post-genomic research is to efficiently manage and retrieve all relevant data. Carrying out a post-genomic clinical trial involves the collection and storage of a wide variety of data, including: clinical data collected on Case Report Forms (e.g. symptoms, histology, administered treatment, treatment response), imaging data (e.g. X-Ray, CT, MR, Ultrasound), genomic data (e.g. microarray data), pathology data and other lab data. Next to that there are many public biomedical databases that are relevant. These store information about gene and protein sequences, pathways, genomic variation, microarray experiments, medical literature, tumour antigens, protein domains, metabolites, etc. Biomedical researchers currently have to use many different tools and web interfaces to find and extract the data that is relevant to their clinical research. Providing seamless and integrated access to clinical, genetic and image databases would therefore greatly facilitate post-genomic research.

In order to provide seamless data access, syntactic and semantic integration needs to take place. Syntactic data integration handles differences in the formats and mechanisms of data access, whereas semantic integration deals with the meaning of information; it must handle the fact that information can be represented in different ways, using different terms and identifiers.

1.1 Scope

The focus of this report is, first of all, on data retrieval. It is assumed that clinical trials are running and that a data entry solution is in place. In order to analyse clinical trial data using the ACGT tools, the data needs to be integrated into the ACGT platform. This involves exporting the data from the hospital database, anonymising it, and storing it in a separate database (typically hosted outside the hospital firewall). This report examines what (Open Source) database solutions are available for storing the anonymized clinical trial data.

The scope of this report is further restricted towards resolving syntactic differences, which is the responsibility of the data access services. Semantic integration in ACGT is carried out by the semantic mediator with help of the ACGT Master Ontology, which are both being developed within WP7. Semantic integration falls outside the scope of this report. It will only be briefly touched upon where relevant.

1.2 Structure

The remainder of this document is structured as follows. Chapter 2 reviews available Open Source database management systems that can be used for storing clinical trial data, and subsequently gives guidelines and recommendations. Chapter 3 introduces the data access services that have been developed within ACGT, which can be used to integrate databases containing clinical trial data into the ACGT platform. Chapter 4 describes in detail how new services can be created for each type of database. Chapter 5 concludes by summarising the current state of data access in ACGT. It gives an overview of what has been achieved, but also discusses the open issues.

2 Databases

This chapter reviews available Open Source databases for storing different types of clinical trial data. Section 2.1 discusses database management systems for storing relational data. Section 2.2 reviews storage solutions for medical image data. Section 2.3 discusses the Open Source systems for storing microarray data.

2.1 Relational data

The integration of relational databases into the ACGT platform is very important for two reasons. Firstly, important clinical trial data is stored in these databases, for example data collected using CRF forms. Secondly, relational databases can also be used to make data available that may not yet be stored in a relational database, but that can be mapped to the relational data model. This holds for data collected in files of various formats, such as Excel files, plain text files (in CSV format for example), XML files, etc.

Suitable Open Source Relational DataBase Management Systems (RDBMS) exist for hosting relational data. The two main ones are PostgreSQL [20] and MySQL [18], which both have been successfully used within ACGT. There have not been attempts to integrate other RDBMS, as there simply has been no need. However, as long as there is a suitable JDBC database driver available, this should be straightforward. The choice of RDBMS depends on various factors, including the following:

- Is there a RDBMS installed already that can be used?
- What type of RDBMS (if any) is used to collect the trial-data? This may affect the ease by which data can be imported. Importing the data is typically easier when the exported data comes from the same type of RDBMS (see also Section 5.1.1).
- Are there any non-functional requirements that are especially important? Performance, reliability and scalability are all potential features on which to base the choice of RDBMS.
- What are the preferences of the system administrator responsible for managing the database?

The choice of RDBMS therefore needs to be made on a case-by-case basis. It may require installation of the RDBMS and the steps that subsequently need to be taken to import the data obviously depend on the RDBMS that is chosen. Instructions on how to install the RDBMS or how to import data are not documented here, as they are fully described in the documentation of the RDBMS itself, e.g. [21, 19]. Once the anonymised data has been successfully imported in a relational database, subsequent configuration of the data access service is identical for all RDBMS systems, except for the database driver that will have to be used.

2.1.1 Clinical trial management systems

As mentioned in Section 1.1 the focus of this report is on storing and providing access to anonymised clinical trial data that has already been collected. Data-entry falls outside the scope of this report. Nevertheless, given their importance within

clinical trials, it is worth briefly discussing clinical data management systems that store data collected on CRFs.

As reported in D5.1 [1], within the TOP trial Oracle Clinical is used for storing the clinical data recorded on CRF forms. Oracle Clinical [39] is the most popular commercial CTMS. In contrast, the SIOP-trial uses a home-grown, trial-specific CTMS, which uses a Postgres database for storage.

OpenClinica [40] is currently the only generic Open Source CRF system for storing clinical trial data. It is being developed by Akaza Research, who are providing the system for free and earning money through additional services, such as installation, hosting, maintenance and validation. Over the past four years OpenClinica has gathered a community of 6000 registered users with 16.000 downloads of the software. At this moment it is used in actual clinical trials at 200 locations worldwide. This is more than double compared to the year before. We have experimented with the use of OpenClinica and found installation to be quick and straightforward. Creation of new CRFs is done using a custom Microsoft Excel template. Learning to create new CRF takes a bit of time, but the fact that configuration of a CRF is stored in a single file has the advantage that it can be easily shared and reused. Whether the performance and functionality is satisfactory when used in practice we have yet to evaluate, but its wide adoption would suggest this is the case.

Within ACGT work is being carried out on an ontology-based CTMS, called Optima [3]. It has an ontology at the core of the data-entry process. All fields on the CRF forms are annotated with (paths of) terms from the ontology that concisely and unambiguously describe the data in a manner suitable for machine-processing. This greatly facilitates integration and comparison of data collected across different clinical trials. However, at the moment Optima is still less mature than OpenClinica and commonly used commercially available CTMS systems.

All the different CTMS provide their own front-ends to support entry of clinical trial data. The schemas they use for storing the data also differ, but what they have in common is that they all store the data in a relational database. So in order to integrate the data into the ACGT platform, the data can be exported and, after anonymisation, subsequently imported into a generic RDBMS. In other words, for storing clinical trial data for the purpose of retrieval and analysis in ACGT, a standard relational database is sufficient.

2.2 DICOM image data

Although there are a significant number of Open Source DICOM image viewers, the number of Open Source DICOM servers is much smaller. Furthermore, compared to commercial PACS solutions, their capabilities are limited. There are only a few suitable free PACS systems that we are aware of.

Conquest [12] is a public domain DICOM server. We have been using it within ACGT to host DICOM images and to test out our data access services. Some of its benefits are that it is easy to install and use, and that it supports DICOM Query/Retrieve (the generic mechanism used by DICOM clients to query and retrieve images). It is still actively being developed (last release April 2009). There is a Windows version and a Linux version. The Linux version that we are using crashes periodically. Although it can be used for testing, it does not offer reliable storage and access to image data. Windows is their main target platform, and we have found the Windows version to be

stable. It successfully coped with high load in the performance experiments that we have carried out [2]. These experiments, however, were designed to test the data access services, and not the underlying DICOM server. We also have not tested scalability w.r.t. the number of images stored; the number of images that we have been hosting using the Conquest DICOM server has been relatively small. Whether or not the use of Conquest is appropriate for a particular use case will require further investigation, and will also need to be decided on a case-by-case basis.

Another free PACS server is the OpenSourcePACS system [15], which is available under a LGPL licence. We have, however, no experience with using it. The main reason is that it lacks DICOM Query/Retrieve support. Although the DICOM protocol is used for storing images, subsequent browsing and viewing of these images is done by way of a dedicated client, using a proprietary format. Its lack of a standard interface for accessing images (i.e. by way of DICOM) meant it has been unsuitable for use within ACGT. Although support for DICOM Query/Retrieve is mentioned on their roadmap, the last release of the software has been more than two years ago (April 2007) and there are no signs of active development, so it is questionable whether it will eventually appear.

The National Biomedical Imaging Archive (NBIA)¹ is a DICOM image storage solution provided by caBIG [17]. It provides web-based access to de-identified DICOM images, markups, and annotations. Although it stores DICOM images, it does not use the DICOM protocol for importing, querying or viewing images. Images are imported using a dedicated FieldCenter tool, which also takes care of anonymisation. In other words, it cannot directly accept images from imaging modalities or other DICOM servers. Querying and retrieval of images can only be done through their web interface. So the NBIA does not provide the image storage and retrieval functionality that is required within a PACS system. It can only be used alongside existing PACS systems.

The dcm4che project [14] also offers a DICOM storage archive, named dcm4chee [13]. We have not used it within the ACGT project, as for purposes of developing and testing the data access services, the Conquest DICOM server was sufficient and easier to set-up. However, compared to the Conquest server, dcm4chee offers more functionality. Furthermore, it is still actively being developed and has an active user community.

Given the limited choice of free PACS servers, it is also worth mentioning the low-cost PACSOne solution [16]. It has good DICOM support, and provides an Apache-based web front-end for administrating the server. It is actively supported, with a support team that responds rapidly.

2.3 Microarray data

For storing microarray data in ACGT we are using BASE (version 2, hereafter only called BASE) [24, 25]. BASE is an Open Source microarray database that is being developed by Lund University, one of the partners in ACGT. It is a web-based database solution for storing the data generated by microarray analysis. BASE is released under the GNU General Public License version 3. Some of its features are:

¹ It used to be called NCIA: National Clinical Imaging Archive.

- It is MIAME compliant
- It is compliant with ArrayExpress, using the Tab2MAGE import/export plug-in
- It uses the concept of projects for managing sharing of data and access control
- It is an array LIMS (Lab Information Management System)
- It is a biomaterial LIMS
- It supports many microarray data formats, including: Affymetrix, Agilent, GenePix, Illumina and custom formats (through customizable import plug-ins)
- It is extensible by way of an advanced plug-in API
- It provides a Web Service interface for data access
- It provides customization via Annotation Type/Categories mechanism
- It includes analysis tools:
 - Lowess Normalization
 - Integration with TIGR MEV through Java Web Start

There exist a number of other software solutions for handling and storing microarray data. The main ones are:

- caArray [41], an open-source management system for storing microarray data. It allows for programmatic access and web access. No analysis tools are available.
- Mimas 3.0 [42], a web-accessible multiomics information management and annotation system. It is designed to store manually annotated expression data from several research facilities that may be organized within a consortium. It can handle the recent, ultra high-throughput DNA sequencing (UHTS) data.
- Microarray Analysis and Retrieval System (MARS) [43], a MIAME-compliant software suite for storing, retrieving, and analyzing multi color microarray data. Not all kinds of platforms are supported (GeneChip and BeadArray).
- Systems Biology Experiment Analysis Management System (SBEAMS) [44], a framework for collecting, storing, and accessing data produced by microarray, proteomics, macroarray, and other experiments. Currently limited support for exporting data in MAGE-ML and MAGE-TAB formats.

The following comparison table is taken from Gattiker et al. [42]. It lists a few more databases and analysis tools than the ones listed above.

Feature	Database	MIMAS 3	BASE 2	MARS	SBEAMS	MiMiR	maxdownload2	MIAMExpress
Affymetrix GeneChip		•	•		•	•	•	•
Two-color Microarray			•	•	•	•	•	•

Illumina BeadArray	•	•					
UHTS	•						
MIAME compliant	•	•	•	•	•	•	•
MAGE-TAB export	•	•					•
MAGE-ML export		•	•		•	•	•
MGED Ontology support	•		•		•	•	•
Connectivity to analysis and visualization tools	•	•	•	•	•	•	
User and group management	•	•	•	•	•		
Object sharing and permissions management	•	•	•	•	•		
Web application	•	•	•	•			•
User Notifications/Messages	•	•	•				
Wizard-based annotation and experiment creation	•	•					

Why BASE?

Some of the advantages of using BASE as the storing and analysis suite for microarray data are listed below:

- Plugin-enabled: Users can contribute analysis tools using a plug-in API.
- Active development: The BASE software is under active development
- Web-services: Data can be retrieved using web-services.
- Array formats: Many arrays format are supported. Custom formats can be used using a customizable import plug-in.
- Multi-user environment: BASE is a multi-user environment that supports cooperation between users while protecting all data against unauthorized access or modification.

3 Data access services in ACGT

This chapter introduces the ACGT data access services. Section 3.1 briefly reviews the functional requirements. Section 3.2 summarises the functionality offered by the data access services. Section 3.3 lists the types of data source for which data access services have so far been developed.

3.1 Functional requirements

As described in D5.2 [2] the data access services need to fulfil the following functional requirements:

- They should provide a uniform data access interface. This includes uniformity of transport protocol, message syntax, query language, and data format.
- They should export the structure of the database, using a common data model, together with possible query limitations of the data source. Clients of the web service require this information for constructing queries.
- They should enforce the data source access policy, and audit access to data sources. For post-genomic clinical trial data, there exist strict legal and ethical requirements that need to be adhered to.

3.2 Functionality

The decisions taken with respect to the design of the data access services are documented in D5.2. Here, we only summarize the main features of the data access services. These can be considered as the benefits that are obtained when making databases accessible by way of the data access services when considered on its own. The largest benefit of integration of new data sources is of course that the data is available for analysis and mining using the suite of services and tools included in the ACGT platform.

The functionality provided by the data access services are:

- Support of a uniform mechanism for querying data. Data can be queried using SPARQL, thus hiding the different query mechanisms provided by the underlying databases.
- Export of the schema of data resources. An RDF Schema of a data resource is available on demand. This schema is, for example, used by the semantic mapping editor to provide the mapping to the ACGT Master Ontology.
- Retrieval and delivery of files. Large files in standardised formats that do not need to be processed by the semantic mediator can be retrieved by ID, and delivered to specified locations, amongst others DMS. Files can optionally be compressed and combined into a single archive to speed-up delivery.
- Credential-based authentication and authorization. The ACGT data access services are fully integrated into the ACGT security infrastructure to control access to the data.

- Dynamic deployment of new data resources. To speed-up integration of new data resources, integration into the ACGT infrastructure can be carried out by the owner of the database using the ACGT web portal.

3.3 Supported data sources

So far, we have implemented data access services for the following databases:

- Relational databases
- DICOM images
- BASE microarray databases

These have been chosen after careful review of the requirements; they are considered the most important in the context of post-genomic clinical trials given the data-mining scenarios that were identified during the requirements-gathering process [1, 4].

4 Integrating new data sources

The main steps needed to integrate new data sources with trial-specific patient data into the ACGT platform are the same for all data sources, irrespective of the type of data they store. The steps that need to be taken are:

1. Export the data from the data source that collects the data.
2. Anonymise and pseudonymise the data (using the CAT tool [7]).
3. Determine who should be allowed access, and if need be, create the appropriate contracts and have these signed by all parties.
4. Set-up a database for hosting the data, and import the anonymised data.
5. Set-up secure access to the database.
6. Create a data access service for the database.
7. Configure the GAS [6] so that authorized users can access the data by way of the data access services.
8. Optionally create the required semantic mapping so that the database can be queried using the ACGT Master Ontology.

From this it should be clear that integration of new data sources is not only a technical task. Successfully making new trial data available also involves clinical partners and legal partners in ACGT.

This report covers Step 4 to 6, as these are technical steps that fall under the responsibility of WP5. Step 6 is described in the most detail, as this step is specific to the ACGT. Step 4 and 5 are more generic, and specific details depend on the type of database that is used. For example, importing data into a postgresSQL database is done differently than importing data into a mySQL database, even though they are both relational databases. The same holds for configuring access. However, documentation on how this can be done is widely available both online, e.g. [19, 21], as in printed hardcopy, e.g. [22, 23], so it will not be described here.

It should be clear from the above that WP5 is not the only work package that is involved when new data sources are being integrated into the ACGT platform. Other work packages that are important are the following:

- WP7 (for semantic integration),
- WP10 (for arranging contract agreements and other legal issues),
- WP11 (for anonymising the data, and providing secure access), and
- WP12 (for providing the clinical trial data, as well as guidance on how to use and interpret the data).

The next three sections describe how to integrate each of the different types of clinical data sources. Section 4.1 describes the data access service for relational databases, Section 4.2 the DICOM data access service, and Section 4.3 the data access service for the BASE microarray database.

4.1 Data access service for relational databases

The functionality of the data access services for relational databases is briefly described in Section 4.1.1. Creating a data access service for a new relational database is relatively complex, as it requires creating a database-specific RDF mapping on top of the relational schema. Section 4.1.2 provides detailed instructions about the steps involved in setting up a relational data access service.

4.1.1 Functionality

The data access service for relational databases offers the generic functionality outlined in Section 3.2. Next to the uniform SPARQL-based query interface, however, it also offers SQL query support. The reason for also providing this interface is that SQL provides query functionality not supported by SPARQL, in particular aggregation. Queries that use aggregation can therefore be executed more efficiently using the SQL query interface. When the SPARQL query interface is used, aggregation will have to be carried at the client side, meaning that potentially much more data needs to be retrieved than when aggregation is carried out by the database.

4.1.2 Creating the service

Once the data has been anonymised, it needs to be imported in a relational database. For information on how to do so, please refer to the documentation of your RDBMS. Access to the database also needs to be set-up. For this it is recommended to create a dedicated account (and password) with read-only access (i.e. only *SELECT* privileges). Furthermore, depending on the location of the database in relation to the web services container and the data that is hosted, it may be necessary to set-up a secure connection between the two, e.g. by way of an ssh tunnel. Within ACGT we have been using the *stunnel* tool [26] to securely access remote databases over the Internet.

Once the relational database has been set up, creation of a data access service subsequently involves the following steps:

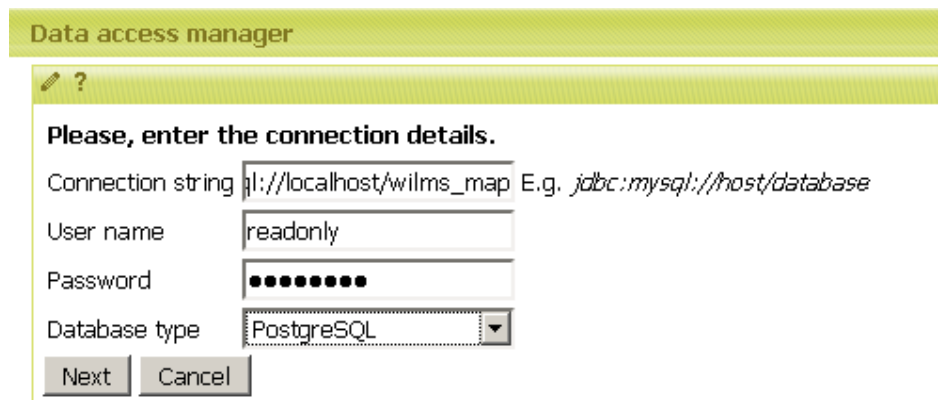
1. Create a default mapping to RDF. This automatically generated mapping is needed so that the database can be queried using SPARQL, the query language supported by the data access services.
2. Manually improve the mapping. This optional step can be used to simplify the mapping, which in turn may simplify the queries that can be used to extract data.
3. Create the data resource using the mapping.
4. Get the (automatically generated) RDF schema. The RDF schema indirectly documents the queries that are supported, and is also needed to create the semantic mapping to the ACGT master ontology.
5. Test the data access service.

These steps may be repeated as needed. The mapping could be improved iteratively, for example, in order to facilitate the semantic mapping of the schema of the data source to the Master Ontology.

The above process is supported by the Data Access portlet. The next subsections describe each step in more detail.

Creating the default RDF mapping

You can start the process from the Data Access portlet by clicking the “Add a new data resource” button. After having done so, you will need to provide the credentials for accessing the database. This is shown in Figure 1.



The screenshot shows a dialog box titled "Data access manager" with a green header. Below the header is a toolbar with an edit icon and a question mark. The main content area contains the text "Please, enter the connection details." followed by four input fields: "Connection string" (with a text input containing "j1://localhost/wilms_map" and a hint "E.g. jdbc:mysql://host/database"), "User name" (with a text input containing "readonly"), "Password" (with a masked input containing "●●●●●●●●"), and "Database type" (with a dropdown menu showing "PostgreSQL"). At the bottom are "Next" and "Cancel" buttons.

Figure 1 Providing database access details

After having done so, a default mapping will be created for the database. The format of the mapping is according to the D2RQ specification [11].

Improving the RDF mapping

Depending on the schema of the underlying database, the desired RDF schema, and the type of RDBMS that is used the default mapping may be used “as is” or may need to be further improved. Below we describe the typical cases where the default mapping can be improved, show how this can be done, and give examples of the results.

We use the database schema that is shown in Figure 2 in all examples. This schema is extremely simple and would normally be part of a larger one; it includes only tables and columns that are relevant to the examples. The *crf_postsurgery* table stores information stored on a “Post Surgery” CRF. The column *rdx* indicates whether the patient received radio therapy; the value “1” means “no”, and “0” means “yes”. The *entry* table stores for each CRF the patient it refers to, and the date that the form was filled in. The *patient* table stores for each patient the date of birth and the sex. The latter uses the *code_sex* code table, which stores the values that are allowed. In this example it contains two possible values: “Vrouw” and “Man”, which are Dutch terms (for purposes of illustration) respectively meaning “Female” and “Male”.

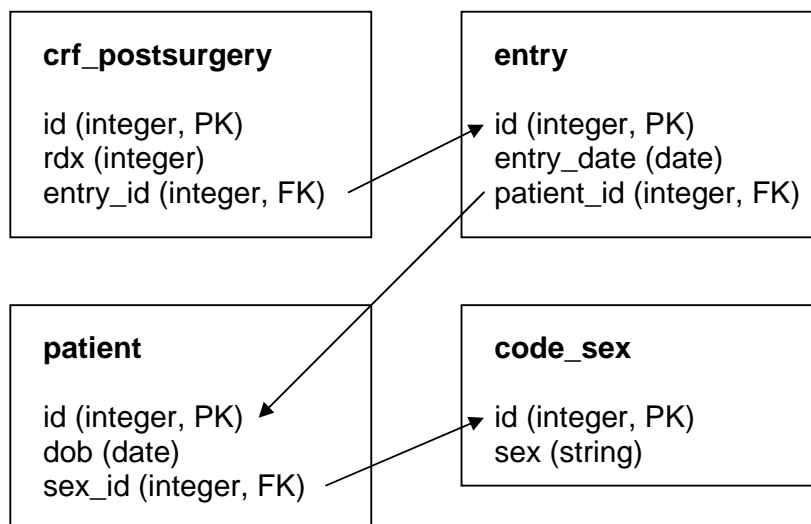


Figure 2 - Simple database schema used for the examples. Table names are in bold, followed by a list of their columns. Between brackets the data type is indicated. Primary keys are denoted by PK, and foreign keys by FK. Arrows indicate the key that is referred to by each foreign key.

Given this schema, we want to carry out the following query:

- “Find all female patient that received post-surgery radiotherapy; for each return the patient identifier, the date of birth, and the date that the post-surgery form was filled in”. The two dates can be used to infer the age of the patient at time of the radiotherapy treatment.

The corresponding SPARQL query, when a basic mapping from SQL to RDF is used, is shown in Figure 3. For a short introduction to SPARQL, please refer to [2]². The mapping that is used to translate this SPARQL query to SQL can be found in Appendix 2A. In the remainder of this section the mapping will be improved step-by-step. The effect will be illustrated by the corresponding change to the SPARQL query.

```

PREFIX vocab: <http://example.philips.com/vocab/>
SELECT ?patient_id ?dob ?entry_date
WHERE {
  ?crf vocab:crf_postsurgery_entry_id ?entry_id ;
       vocab:crf_postsurgery_rdx "1"^^xsd:int .
  ?entry vocab:entry_id ?entry_id ;
         vocab:entry_entry_date ?entry_date ;
         vocab:entry_patient_id ?patient_id .
  ?patient vocab:patient_id ?patient_id ;
           vocab:patient_sex_id ?sex_id ;
           vocab:patient_dob ?dob .
  ?code_sex vocab:codesex_id ?sex_id ;
            vocab:codesex_sex "Vrouw" .
}
  
```

² A slightly longer and more end-user friendly introduction to SPARQL will appear in the ACGT user manual that is currently being written.

Figure 3 Example query, initial version

The vocabulary terms for properties in the default mapping are derived from the table and column names in the underlying database. This is done as follows: `vocab:{tablename}_{columnname}`, where `vocab` denotes the configurable namespace prefix URI. This simple scheme will typically suffice, but there can be various reasons for changing these names. For example, when two databases have similar schema, renaming RDF properties may help to make the RDF schema more similar or even identical. This has been the case for the TOP scenario that we have implemented and demonstrated [8]. There were minor differences in the schema of the public database (which was used for testing), and the private database (which contained the actual data). We renamed properties to make both RDF schema identical. This way, the semantic mapping could be tested on the public database, yet the resulting mapping would be valid on the private database. Renaming a property is achieved by simply changing the value for the corresponding `d2rq:property` triple in the mapping.

Another simple change is adding information about datatypes. The default mapping will add a datatype to each property bridge where it can infer this. However, is not always possible for all properties, depending on the type of the RDBMS and the schema of this database. In these cases, it may be desirable to improve the mapping by manually adding the datatype information. This is achieved by adding a `d2rq:datatype` triple to the corresponding property bridge.

In our example we can improve the mapping by renaming the term `vocab:crf_postsurgery_rdx` to `vocab:crf_postsurgery_radiotherapy`, in order to be more descriptive. Also, given that it is a boolean value, we can change the mapping so that the type changes from `xsd:int` to `xsd:boolean`. Figure 4 shows what the query looks like after these changes have been made to the D2RQ mapping. The part of the query that is changed is shown in italics.

```
PREFIX vocab: <http://example.philips.com/vocab/>
SELECT ?patient_id ?dob ?entry_date
WHERE {
  ?crf vocab:crf_postsurgery_entry_id ?entry_id ;
       vocab:crf_postsurgery_radiotherapy "true"^^xsd:boolean .
  ?entry vocab:entry_id ?entry_id ;
         vocab:entry_entry_date ?entry_date ;
         vocab:entry_patient_id ?patient_id .
  ?patient vocab:patient_id ?patient_id ;
           vocab:patient_sex_id ?sex_id ;
           vocab:patient_dob ?dob .
  ?code_sex vocab:codesex_id ?sex_id ;
            vocab:codesex_sex "Vrouw" .
}
```

Figure 4 Example query, Version 1 – Reflecting change in radiotherapy property

More complex changes, which affect the structure of the RDF schema, can be achieved as well, for example to hide the use of foreign keys. The default mapping will hide the use of foreign keys when it can retrieve the foreign key information from the underlying RDBMS. In this case, all foreign key values in the relational database will be represented by an IRI in the RDF model that represents the row in the table that the foreign key refers to. If the RDBMS does not return information about foreign

keys, in particular the column and table that it refers to, the mapping will have to be edited manually to achieve the same effect.

In our example, we can improve the D2RQ mapping to hide the use of the *sex_id* foreign key in the patient table. The *vocab:patient_sex_id* property, which used to refer to this foreign key, can be replaced by a new *vocab:patient_sex* property that directly refers to the RDF node that represents the row in the *code_sex* table that is referred to by the foreign key. Figure 5 shows the new query after the mapping has been changed this way.

```
PREFIX vocab: <http://example.philips.com/vocab/>
SELECT ?patient_id ?dob ?entry_date
WHERE {
  ?crf vocab:crf_postsurgery_entry_id ?entry_id ;
       vocab:crf_postsurgery_radiotherapy "true"^^xsd:boolean .
  ?entry vocab:entry_id ?entry_id ;
         vocab:entry_entry_date ?entry_date ;
         vocab:entry_patient_id ?patient_id .
  ?patient vocab:patient_id ?patient_id ;
           vocab:patient_sex ?code_sex ;
           vocab:patient_dob ?dob .
  ?code_sex vocab:codesex_sex "Vrouw" .
}
```

Figure 5 Example query, Version 2 – Not using key for code_sex table anymore

When simple code tables are used that consist of two columns, a primary key and the corresponding code value, the mapping can be further simplified to remove another level of indirection. Foreign keys that refer to a row in such a code table can be represented in RDF not by the row in this table, but by the corresponding code value. The latter can be returned as a literal, e.g. “Vrouw”, or as a URI that represents the code, e.g. <http://example.philips.com/codes#Female>. Figure 6 shows the query after the mapping has been updated to translate each literal string value in the *code_sex* code tables to a URI. Also, the indirection by way of the *code_sex* table has been removed; the value of the patient’s sex is now directly available from the RDF node that represents the patient.

```
PREFIX vocab: <http://example.philips.com/vocab/>
PREFIX code: <http://example.philips.com/codes#>
SELECT ?patient_id ?dob ?entry_date
WHERE {
  ?crf vocab:crf_postsurgery_entry_id ?entry_id ;
       vocab:crf_postsurgery_radiotherapy "true"^^xsd:boolean .
  ?entry vocab:entry_id ?entry_id ;
         vocab:entry_entry_date ?entry_date ;
         vocab:entry_patient_id ?patient_id .
  ?patient vocab:patient_id ?patient_id ;
           vocab:patient_sex code:Female ;
           vocab:patient_dob ?dob .
}
```

Figure 6 Example query, Version 3 – Now using URIs to refer to code table entries

The use of foreign keys and code tables are relational artifacts that can be successfully hidden from the resulting RDF model. More complex relational artifacts, such as tables for storing n-to-m relations can be hidden as well. These tables will

show up as classes in the default mapping, but can be made to disappear by adding extra *d2rq:join* triples to the property bridges corresponding to the tables that are subject of these n-to-m relations.

In our example, we can for example hide the *entry* table from the RDF mapping. This can be done by adding the entry date as a property of the CRF class. Also, the patient is added as a property of this class. The latter requires two joins, which is supported by D2RQ. Figure 7 shows what the query looks like after these changes. The final D2RQ mapping can be found in Appendix 2B. By comparing it to the original D2RQ mapping you can see how each of these improvements has been achieved.

```
PREFIX vocab: <http://example.philips.com/vocab/>
PREFIX code: <http://example.philips.com/codes#>
SELECT ?patient_id ?dob ?entry_date
WHERE {
  ?crf vocab:crf_postsurgery_entry_date ?entry_date ;
       vocab:crf_postsurgery_radiotherapy "true"^^xsd:boolean ;
       vocab:crf_postsurgery_patient ?patient .
  ?patient vocab:patient_id ?patient_id ;
           vocab:patient_sex code:Female ;
           vocab:patient_dob ?dob .
}
```

Figure 7 Example Query, Version 4 – Not referring to entry table anymore

Creating the data resource

Once a satisfactory mapping has been obtained, the data resource can be created. This is achieved from the Data Access service by providing a service name and supplying the RDF mapping (presuming that you are not using the default one), and clicking the “Register resource” button. See Figure 8. The service name that you provide is used as a prefix. Each resource will have a unique but arbitrary identifier appended at the end. Upon successful creation, the name of the newly created resource is returned, as shown in Figure 9. When a dynamic data resource is not needed anymore, it can be removed by the user that created it using the link that is provided.

Data access manager


Service name

RDF Mapping

```
# Table wilms_assays
map:wilms_assays a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern
  "wilms_assays/@@wilms_assays.array_id|urlify@";
  d2rq:class vocab:wilms_assays;
.
map:wilms_assays__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:wilms_assays;
  d2rq:property rdfs:label;
  d2rq:pattern "wilms_assays
  #@@wilms_assays.array_id@";
.
map:wilms_assays_siopnr a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:wilms_assays;
  d2rq:property vocab:wilms_assays_siopnr;
  d2rq:column "wilms_assays.siopnr";
  d2rq:datatype xsd:int;
.
map:wilms_assays_scanname a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:wilms_assays;
  d2rq:property vocab:wilms_assays_scanname;
  d2rq:column "wilms_assays.scanname";
.
map:wilms_assays_molid a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:wilms_assays;
  d2rq:property vocab:wilms_assays_molid;
  d2rq:column "wilms_assays.molid";
.
map:wilms_assays_array_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:wilms_assays;
```

Figure 8 Providing a name prefix and RDF mapping for a new data resource

Data access manager

 **THE RESOURCE WAS REGISTERED SUCCESSFULLY**

Resource name	Resource owner	Creation date	Action
Dynamic-ogsadai-120b4d712d2	Jane Doe	Apr 20, 2009 11:29:01 AM	
Dynamic-ogsadai-120b4d712d7	Erwin Bonsma	Apr 20, 2009 11:30:03 AM	Delete

Figure 9 Successful creation of a new data resource

The functionality to restrict access to dynamically created resources is currently still being refined. The functionality can therefore be expected to change. Having said that, the current functionality is as follows: A dynamically created data resource that is accessible to anyone with valid ACGT credentials. So, although access to the dynamically created resource is managed by way of the GAS, the GAS does not have specific access rules for this resource. This means that you should only create dynamic resources for data that does not require more specific access restrictions.

Getting the RDF schema

****Await support in Data Access portlet****

Testing the service

****Await support in Data Access portlet****

Using the service

If all works fine, the data resource is ready for use. This can be done in either of two ways. The data resource can be queried directly from the workflow enactor, or the data resource can be queried by way of the semantic mediator. The former is appropriate for small, ad-hoc databases, that you want to integrate quickly into a data-mining workflow. It does not require any additional actions. Querying by way of the semantic mediator is appropriate for databases containing clinical data that falls inside the scope of the ACGT master ontology. It requires creating a semantic mapping, and possibly, modifying and extending the ontology. In order to justify the effort that this requires, this is typically only done for databases that will be queried by multiple users over a longer period of time. A good example is the clinical trial databases that store the collected CRF data. Semantic mappings can be created by ontology experts using the semantic mapping tool. This tool will dynamically retrieve the RDF schema of the data resource, which the user will then have to map to the ACGT Master Ontology.

Creating a static data resource

The preceding steps described how to create a dynamic data resource using the ACGT portal. This process has the advantage that it is convenient and straightforward. Also, it can be carried out by anyone with a valid ACGT certificate. Dynamically created data resources are, however, not ideally suited for all use cases. They have the following drawbacks: Firstly, they are not persistent; they will be gone when the web services container is restarted. The latter happens fairly rarely, but can be needed when there is a problem, or when the functionality of the data access services needs to be upgraded. Secondly, it is currently not possible to change their default access control rules. Thirdly, their names are slightly verbose, which makes them more difficult to remember and enter. Fourthly, they will always return the RDF schema that is automatically generated. This is appropriate for most cases, but it can sometimes be helpful to extend the schema with further information that cannot be easily inferred from the D2RQ mapping file, such as class inheritance relations.

When any of these drawbacks is problematic in a particular use case, a static data resource can be created. A static data resource does not suffer from these drawbacks; however, it is slightly more cumbersome to create. It requires changing the configuration of the web service container, which can only be done by an administrator of the web services container. The data source specific information that

is required is: the D2RQ mapping file, and an RDF schema file. These can, however, conveniently be generated by first creating a dynamic version of the data resource, which can be carried out by a user that is familiar with the database. Once this is done, this configuration information can be given to the administrator of the data access services, who can then quickly create the corresponding static data resource. Subsequently the GAS can be configured to create dedicated and specific access rules for the new data resource.

4.2 Data access service for DICOM image data

Section 4.2.1 describes the functionality of the data access service for DICOM images. Section 4.2.2 subsequently documents how to create the service. For more details about the DICOM data access service, including a detailed description of its implementation, please refer to D5.2 [2].

4.2.1 Functionality

The DICOM data access service provides all the generic functionality described in Section 3.2. The SPARQL query interface can be used to carry out queries about the images that are stored. Next to that, the service offers additional retrieval functionality. The (unique) identifiers for one or more images can be specified, and these images will be retrieved from the DICOM server. After optional further processing, e.g. compression and archival into a ZIP file, the images can be delivered to a chosen location. The OGSA-DAI image retrieval activity has been designed so that it takes the results of a query activity as input. This means that image query and retrieval can be executed using a single request to the data access service.

The data access services are not specific to a particular type of PACS. The data access services for DICOM images use DICOM Query/Retrieve to interact with the image databases, as this is the widely adopted standard for querying and retrieving medical images.

4.2.2 Creating the service

Unlike the data access services for relational databases, which can be created dynamically and statically, the data access services for DICOM can only be created statically. In other words, it cannot be done by way of the ACGT portal but requires configuration of the web services container by an administrator. Functionality could be added to the portal to also support dynamic creation of DICOM data resources. However, this has not been implemented as there has not been a need yet for this functionality. See also Section 5.2.2.

The RDF schema for all DICOM image resources is identical, as the classes and their attributes that can be queried are fully specified by the DICOM protocol. This means that a single semantic mapping suffices for all DICOM data resources, which in turn implies that any new image resource, can be queried using the semantic mediator automatically, without any extra effort. This difference with respect to relational data resources also means that configuration of data access services for DICOM image data is much simpler. The following subsection document how a new DICOM data access service can be configured, and how the new data access service can subsequently be tested.

Configuring the service

The template for the DICOM configuration file can be found in the software archive³. It contains more than a dozen parameters that need to be set, but as these are documented in the template configuration file itself, only a high-level overview is given here.

- The *queryRetrieveSCP* element is used to specify the name and address of the database server that stores the images.
- The *queryRetrieveSCU* element specifies the name and address of the DICOM Service Class User (SCU) *at the web service container* that interacts with the DICOM server to retrieve the image meta-data. Each DICOM data resource will create its own SCU, so you should ensure that the name and address are unique. You should also configure the DICOM server so that it allows access to the SCU that will be created⁴.
- The *storageSCP* element specifies the names and addresses of the DICOM Service Class Providers (SCP) *at the web service container* that are used to receive images from the DICOM server. You can create more than one, so that multiple images can be retrieved in parallel. As was the case for the Query/Retrieve SCU, you need to make sure that the names and addresses are unique.

Testing the service

After the data resource has been created and the GAS has been configured to allow access, the data resource can be tested. The best way of doing so is by way of the command-line OGSA-DAI client. You can use the one that is provided as part of the OGSA-DAI WSRF 2.2 distribution, or use the extended OGSA-DAI client that is available from the ACGT source code repository. Details on how to use the client can be found on the ACGT wiki⁵. Example OGSA-DAI perform documents are available in the ACGT source code repository for both querying what images are available and for retrieving them. Further details can also be found on the ACGT wiki⁶.

A DICOM data resource can also be tested by creating an executing a workflow that queries it and that retrieves images. It has the benefit that it has a more user-friendly interface (a GUI is used instead of a command line environment). However, a drawback is that it involves more components, which increases the chance of failure and may require more effort to troubleshoot any problems.

4.3 Data access service for microarray data

The BASE data access service has been implemented after D5.2 was written, so unlike the preceding two data access services, it is not described there. However,

³

<https://iapetus.ics.forth.gr/svn/ACGT/WrapperServices/trunk/etc/resource/Templates/DICOM/dataResourceConfig.xml>

⁴ In the case of the ConQuest DICOM server, you will have to edit the *acname.map* to add the so-called DICOM Application Entity.

⁵ http://wiki.healthgrid.org/ACGT:Wrappers_-_Invocation

⁶ http://wiki.healthgrid.org/ACGT:Wrappers_-_Data_Resources

conceptually it is a very simple service, similar to the data access service for DICOM images. Section 4.3.1 describes the functionality in more detail. It is followed by Section 4.3.2, which describes how to create new services.

4.3.1 Functionality

The BASE data access service delivers all functionality described in Section 3.2, except that it lacks a SPARQL-query interface. The data access service offers retrieval functionality similar to the DICOM data access service: one or more unique assay identifiers can be specified, and the service will retrieve the corresponding microarray files. It also offers very basic query functionality: it can return the identifiers of all assays that can be retrieved. The reason that more advanced query functionality is not (yet) provided is that there has not been a need. In all the use cases so far, the metadata about the microarray data has always been stored in a separate relational database. This relational database stores for each microarray file all relevant information for the purposes of the clinical trial, including the pseudonymised identifier of the patient it corresponds to. When there is a specific need to query the BASE database directly, a SPARQL query interface can be added to support this. However, as was the case for the DICOM SPARQL query interface, adding such an interface will require significant effort, so this will need to be justified by the benefits of doing so.

The current interface of the BASE data access services is generic and applicable to other microarray storage systems. However, the data access services use a BASE-specific web-service interface to interact with BASE, so the implementation is specific to BASE. Given additional effort, however, additional implementations could be provided should there be a need, so that the same interface can be used to retrieve microarray files from different microarray storage systems.

4.3.2 Creating the service

BASE data access services can currently only be deployed statically. The subsections below document how to configure new services at the web service container and how they can subsequently be tested.

Configuring the service

Configuration of the BASE data access service is very straightforward. It only requires setting three parameters, which are all self-explanatory: the URL of the BASE web service interface, the password and the username.

The web service interface of the BASE database, not to be confused with the BASE data access service, is not integrated into the ACGT security infrastructure. As a result, communication with this service is not encrypted by default. Unless the data access service is located on the same server as the BASE database, it is therefore recommended to use an encrypted connection, e.g. by way of the stunnel tool [26].

Testing the service

Testing a BASE data access service is very similar to testing a DICOM data access service, as is described at the end of Section 4.2.2. When you use the OGSA-DAI command-line client, you can first retrieve the list of all available assays, for which

you can use the perform document GetAllAssayIDs.xml available in SVN⁷. You can subsequently retrieve and deliver some or all assays to DMS. Example perform documents for doing so can also be found in SVN, for example in the *examples/perform/acgt_assays* directory. You will, however, need to modify the perform documents to include the identifiers of the assays that you want to retrieve, as well as the desired destination directory on DMS.

7

<https://iapetus.ics.forth.gr/svn/ACGT/WrapperServices/trunk/examples/perform/generic/assays/GetAllAssayIDs.xml>

5 Discussion

This deliverable closes with a discussion of the current state of data access within ACGT. More specifically, Section 5.1 relates our experiences with the databases that store the various types clinical trial data. Section 5.2 discusses the data access services that provide access to each of these databases in the ACGT platform.

5.1 Databases

5.1.1 Relational databases

There are good free software solutions available for storing relational data. Within the ACGT project, we have successfully used MySQL and PostgreSQL. We experienced the biggest problems, which were still only minor, when importing data into the databases. For example:

- We had difficulty importing a dump from a PostgreSQL database into the MySQL database that we had installed already. Although feasible, it would require substantial effort so we decided to also deploy a PostgreSQL database on our gridnode server, to import the data there instead.
- Subsequently we had some difficulty importing the dumped data into our PostgreSQL database, as it was an older version than the one that generated the dump. Upgrading the database on our gridnode server was not really feasible, given the version of the operating system it used. So we created scripts for stripping non-supported parts from the dump. After running the dump through these scripts, the import subsequently succeeded.
- We also experienced minor problems preserving special characters in the data (e.g. umlauts and other German characters). To solve these we needed to ensure that appropriate text-encodings were used everywhere (i.e. in the database dump file, the encoding of database, the settings of the client viewing the data, and the code page settings of the terminal where the client executes). Furthermore, explicit conversions needed to be carried out where encodings differed. For example, in order for the scripts to process the dump correctly, the dump was using the Latin1 encoding, whereas the encoding used in the database is UTF8.

These issues could be solved without much problem. They are documented here because it is useful to be aware of them. In general though, both MySQL and PostgreSQL met our requirements and have been used without problems so far.

5.1.2 DICOM databases

Within the ACGT project there has so far been little need to share DICOM image data as there have been few use-cases for mining of image data within ACGT. The focus within the project with respect to data-mining has been on finding bio-markers in gene expression data. This is not surprising, given that the latter is more amenable for data-mining. Extracting bio-markers from image data requires advanced image processing algorithms to extract potential bio-markers, which is not an expertise present within the consortium.

For our needs so far within the project the Linux-based Conquest DICOM server has been sufficient. Should the image storage needs, the required reliability, or query load increase significantly, we will need to evaluate if it is still appropriate, or whether

an alternative solution needs to be found. In that case, the dcm4chee DICOM archive is the most promising Open Source alternative to consider.

5.1.3 BASE database

For storing microarray files we have used the BASE database hosted by Lund University. There have been some issues with BASE and the server hosting the BASE environment:

- Since BASE currently is not supporting “https” access a workaround was chosen which uses the stunnel SSL wrapper. We have experienced some stability problems with stunnel running on the BASE server. It sometimes requires a restart. BASE will eventually support SSL and then there will not be a need anymore to use stunnel for providing secure access.
- Importing array data to BASE is not always straightforward. We have had some problems importing array designs into BASE. If the documentation of the array design is missing or limited then it is problematic to import it into BASE, so good documentation should always be provided.

5.2 Data access services

Each of the data access services will be discussed separately in the sections below. Section 5.2.1 discusses the current implementation of the data access services for relational database. The data access service for DICOM images is subsequently discussed in Section 5.2.2. Section 5.2.3 briefly discusses the BASE data access service. Dynamic deployment of data access services is the subject of Section 5.2.4. Section 5.2.5 concludes with a discussion of the open issues.

5.2.1 Data access service for relational databases

Out of the data access services that have been implemented so far, the one for relational databases is the most complex with respect to the functionality it provides and the range of interactions it supports. More specifically, the types of queries that can be carried out as well as the queries that users would like to carry out vary per database. The queries depend on database schema, on the data that is stored, as well as the research question that is being tackled. A good way of finding out how useful the data access services are in practice is therefore to use them on a range of databases in different scenarios, which is what we have done over the last two years. The next subsection uses the most challenging scenario (from the point of view of the relational data access service) that we have addressed so far to illustrate the capabilities and limitations of the data access service. It is followed by another subsection that concludes by describing the current state of the relational data access services and by providing our suggestions for handling existing limitations.

Capabilities and limitations

The scenario we consider in this subsection is the Wilms tumor analysis scenario that was demonstrated at the 3rd ACGT review in Saarbrücken [9]. The aim of this scenario was to reproduce the results published in [28]. This scenario is interesting from the perspective of data access for several reasons. Firstly, it provided actual data (with typical problems such as missing and incorrect values). Secondly, the data came from an actual CRF database, with a schema whose complexity reflected that of a typical clinical trial. Thirdly, the queries to be carried out were good examples of

actual queries that end-users are interested in when carrying out biomedical research (as opposed to example queries crafted by developers to show off features of the framework they have developed).

One query that needed to be carried out was to obtain the histological classification of all tumor samples that had undergone microarray analysis. There were several complicating factors in extracting the data:

- The microarray analysis was not part of the SIOP trial as it had originally been defined, which meant that the SIOP CRF database, from which the classification was to be extracted, does not store any information specific to the microarray analysis that was carried out. I.e. the samples that were analysed using microarrays were not marked as such in the database. The subset of patients whose tumors had undergone microarray analysis were only stored in a separate database, which still needed to be queried separately. This database is, for sake of simplicity, excluded from consideration in the remainder of this section.
- Some patients have bilateral tumors, i.e. affecting both kidneys. In this case, two classifications should be returned, one for each kidney.
- Samples are classified multiple times, with increasing quality of the classification results. First a local pathologist classified the sample, then a pathologist at the reference center, and finally a panel of pathologists at the reference center. The classifications that were available for each tumor varied. The classification by the reference panel was for example not (yet) available for all samples. Selecting the most reliable classification therefore had to be done for each tumor independently.
- There were missing values in the database for some fields needed in the query. E.g. the date of the pathology examination was sometimes missing, as well as the type of assessment.
- For some patients multiple pathology examinations were carried out. The query should only return the classification of the first pathology sample taken after main surgery. However, the "main surgery sample" was not explicitly marked as such. It had to be inferred by comparing the dates of the samples. The earliest sample was that taken after main surgery.

Figure 10 shows the resulting query in SQL on the schema used by the original SIOP database. The query uses aggregation at various places (to select the most reliable classification of the sample taken after main surgery) so that for each patient only one classification is returned, or two for bilateral patients.

```
select distinct
  tmp_wilms_assays.molid as "ID",
  f4.pdate - patient.gebdat as "Histology",
  best_histology.max_pathtyp as "PathType",
  code_pathtyp.name as "PathType_",
  f4_sub.histotyp as "HistoType",
  code_histotyp.histotyp as "HistoType_",
  f4.lokal_prob as "SampleLoc",
  code_lokal_prob.name as "SampleLoc_"
from patient
join tmp_wilms_assays using (siopnr)
join f4 using (pnr)
join code_lokal_prob on code_lokal_prob.idx = f4.lokal_prob
join f4_sub on f4.idx = f4_sub.f4idx
join code_pathtyp on code_pathtyp.idx = f4_sub.pathtyp
```

```

join code_histotyp on code_histotyp.idx = f4_sub.histotyp
join (select pnr, lokal_prob,
          min( coalesce(pdate, date '2100-01-01') ) as event_date
      from f4
      group by pnr, lokal_prob
    ) as first_histology using (pnr, lokal_prob)
join (select pnr, lokal_prob,
          coalesce(pdate, date '2100-01-01') as event_date,
          max(pathtyp) as max_pathtyp
      from f4
      join f4_sub on f4.idx = f4_sub.f4idx
      where (f4_sub.stud_ass is null
            or f4_sub.stud_ass in (2, 4))
      group by pnr, lokal_prob, pdate
    ) as best_histology
  on first_histology.pnr = best_histology.pnr and
  first_histology.lokal_prob = best_histology.lokal_prob and
  first_histology.event_date = best_histology.event_date
where (first_histology.event_date =
       coalesce(f4.pdate, date '2100-01-01'))
  and first_histology.lokal_prob = f4.lokal_prob
  and f4_sub.pathtyp = best_histology.max_pathtyp
order by tmp_wilms_assays.molid;

```

Figure 10 - The histology query for the Wilms scenario, in SQL for the original database.

The above query can be carried out using the SQL interface of the relational data access service. However, if the semantic mediator is to be used, it needs to be expressible in SPARQL as well. Given that SPARQL does not support aggregation, it is not possible to create a SPARQL query that returns the same results as the SQL query in Figure 10. Instead, the SPARQL query will have to return more results, and aggregation will have to take place client-side. In other words, the SPARQL query will for each patient return all classifications of all histology samples that have been taken. It is up to the client to select the relevant one for each patient.

Figure 11 shows what the histology query looks like when it is expressed in SPARQL on a Wilms database that uses the default D2RQ mapping to support the query. It is shorter than the SQL query in Figure 10 only because it is lacking aggregation. Unfortunately, this query is not handled correctly by D2RQ. There are two problems with it. Firstly, there is a problem handling the second OPTIONAL block. This block should ensure that results are also returned for samples for which there is no value for the *vocab:f4_sub_stud_ass* property. However, there is a bug in D2RQ 0.6, the version that we are currently using, that stops these results from being returned anyway [30]. Secondly, the filter condition is also not handled efficiently. For each result it will send a separate SQL query for the *stud_ass* field, as opposed to just filtering the returned results and/or including the filter as part of the SQL statement. As a result, the query executes unacceptably slowly.

```

SELECT ?siopnr ?gebdat ?pdate ?lokal_prob_idx
       ?pathtyp_idx ?histotyp_idx ?stud_ass_idx
WHERE {
  ?patient vocab:patient_siopnr ?siopnr ;
           vocab:patient_gebdat ?gebdat .
  ?f4 vocab:f4_pnr ?patient ;
     vocab:f4_lokal_prob ?code_lokal_prob .
  ?f4_sub vocab:f4_sub_f4idx ?f4 ;
         vocab:f4_sub_pathtyp ?code_pathtyp ;

```

```

        vocab:f4_sub_histotyp ?code_histotyp .
?code_lokal_prob vocab:code_lokal_prob_idx ?lokal_prob_idx .
?code_pathtyp vocab:code_pathtyp_idx ?pathtyp_idx .
?code_histotyp vocab:code_histotyp_idx ?histotyp_idx .
OPTIONAL {
    ?patient vocab:f4_pdate ?pdate .
}
OPTIONAL {
    ?f4_sub vocab:f4_sub_stud_ass ?code_stud_ass .
    ?code_stud_ass vocab:code_asse_idx ?stud_ass_idx .
}
FILTER (
    !BOUND(?stud_ass_idx) || ?stud_ass_idx = 2 || ?stud_ass_idx = 4
)
}

```

Figure 11 - The histology query in SPARQL, when the default mapping is used for the Wilms database.

To address these two problems, we changed the D2RQ mapping and the query slightly. The improved query is shown in Figure 12. We changed the D2RQ mapping to address the first problem. The use of the *stud_ass* foreign key in the *f4_sub* table, which caused problems because it could be null, was hidden in the original D2RQ mapping (the property directly linked to an entry in the *code_asse* code table). We extended the mapping with a *vocab:f4_sub_stud_ass_fix* property so that the foreign key could be retrieved directly and used instead in the OPTIONAL block. The problem with the inefficient handling of the FILTER condition was handled by simply removing the filter. All values are returned, and it is up to the client to discard the results with inappropriate values for the *?stud_ass_idx* variable.

```

SELECT ?siopnr ?gebdat ?pdate ?lokal_prob_idx
        ?pathtyp_idx ?histotyp_idx ?stud_ass_idx
WHERE {
    ?patient vocab:patient_siopnr ?siopnr ;
            vocab:patient_gebdat ?gebdat .
    ?f4 vocab:f4_pnr ?patient ;
        vocab:f4_lokal_prob ?code_lokal_prob .
    ?f4_sub vocab:f4_sub_f4idx ?f4 ;
            vocab:f4_sub_pathtyp ?code_pathtyp ;
            vocab:f4_sub_histotyp ?code_histotyp .
    ?code_lokal_prob vocab:code_lokal_prob_idx ?lokal_prob_idx .
    ?code_pathtyp vocab:code_pathtyp_idx ?pathtyp_idx .
    ?code_histotyp vocab:code_histotyp_idx ?histotyp_idx .
    OPTIONAL {
        ?patient vocab:f4_pdate ?pdate .
    }
    OPTIONAL {
        ?f4_sub vocab:f4_sub_stud_ass_fix ?stud_ass_idx .
    }
}

```

Figure 12 - Improved SPARQL query that works around problems of D2RQ.

The Wilms data was also imported into the Obtima database. We also tried to extract the histology data from there, using the SPARQL-interface of the relational data access service. The schema of the Obtima database is entirely different from the schema of the Wilms database. The latter closely reflects the CRF forms of the SIOP trial. There are specific tables corresponding to each (part of) a CRF, and each entry on a CRF is represented by a specific column in the database. The Obtima schema, on the other hand, is generic; it can be applied to any clinical trial. It can be thought of as a meta-schema. There are no tables or columns corresponding to specific CRF forms, or entries on those forms. Instead, these will be represented by values stored inside generic tables. As a result, the query looks completely different.

Figure 13 shows the SPARQL query for the Obtima database, as it was initially created. The query is impossible to understand on its own, due to the generic nature of the Obtima schema. This, however, is not a problem. SPARQL queries for Obtima are not intended to be created manually. Instead, data will be retrieved from Obtima by the semantic mediator using queries expressed in the Master Ontology. These queries can be understood on their own.

```

SELECT ?patient ?v ?v1 ?v3 ?v4 ?v5
WHERE {
  ?patient vocab:Patient_CRF ?crf .
  ?crf vocab:CRF_ItemGroup ?itemgroup ;
       vocab:CRF_ItemGroup ?itemgroup2 ;
       vocab:CRF_ItemGroup ?itemgroup3 ;
       vocab:CRF_ItemGroup ?itemgroup4 ;
       vocab:CRF_ItemGroup ?itemgroup5 ;
       vocab:ci_crf_template_fk ?rest0 .

  ?itemgroup vocab:Itemgroup_Item ?item ;
             vocab:ige_itemgroup_template_fk ?rest1 .
  ?item vocab:item_entry_value ?v ;
        vocab:ie_item_template_fk ?rest2 .

  ?itemgroup2 vocab:Itemgroup_Item ?item2 ;
             vocab:ige_itemgroup_template_fk ?rest4 .
  ?item2 vocab:item_entry_value ?v2 ;
         vocab:ie_item_template_fk ?rest5 .

  ?itemgroup3 vocab:Itemgroup_Item ?item3 ;
             vocab:ige_itemgroup_template_fk ?rest7 .
  ?item3 vocab:item_entry_value ?v3 ;
         vocab:ie_item_template_fk ?rest8 .

  ?itemgroup4 vocab:Itemgroup_Item ?item4 ;
             vocab:ige_itemgroup_template_fk ?rest9 .
  ?item4 vocab:item_entry_value ?v4 ;
         vocab:ie_item_template_fk ?rest10 .

  ?itemgroup5 vocab:Itemgroup_Item ?item5 ;
             vocab:ige_itemgroup_template_fk ?rest11 .
  ?item5 vocab:item_entry_value ?v5 ;
         vocab:ie_item_template_fk ?rest12 .

  FILTER ( ?rest0 = "512"^^xsd:integer )
  FILTER ( ?rest1 = "1756"^^xsd:integer )

```

```

FILTER ( ?rest2 = "3432"^^xsd:integer )
FILTER ( ?rest4 = "1757"^^xsd:integer )
FILTER ( ?rest5 = "3433"^^xsd:integer )
FILTER ( ?rest7 = "1758"^^xsd:integer )
FILTER ( ?rest8 = "3434"^^xsd:integer )
FILTER ( ?rest9 = "1690"^^xsd:integer )
FILTER ( ?rest10 = "3366"^^xsd:integer )
FILTER ( ?rest11 = "1603"^^xsd:integer )
FILTER ( ?rest12 = "3279"^^xsd:integer )
}

```

Figure 13 - The initial SPARQL query when carried out on the Obtima database.

A problem, however, is that this query cannot be executed efficiently by D2RQ. There is currently a limitation in D2RQ's query translation engine that introduces too many unnecessary joins in the SQL query that is produced [31]. These joins are basically null-operations that do not affect the results. A smart SQL query handling engine at the RDBMS may realize this and avoid carrying these out. However, when this is not the case, the SPARQL query is sufficiently complex, and the tables in the underlying database are sufficiently large, performance can become unacceptable. Queries can take too long to execute (several minutes or more), or may even crash the database (e.g. when they require more temporary storage than is available).

In an attempt to work around this problem we extended the D2RQ mapping with additional properties, one for each item group instance and one for each item instance. This can simplify the SPARQL queries, and also results in a faster execution. The SPARQL query that uses the extended D2RQ mapping is shown in Figure 14.

```

SELECT ?patient ?v ?v2 ?v3 ?v4
WHERE {
  ?patient vocab:Patient_CRF512 ?crf ;
           vocab:Patient_CRF511 ?crf2 .

  ?crf vocab:CRF_ItemGroup1756 ?itemgroup .
  ?itemgroup vocab:Itemgroup_Item3432 ?item .
  ?item vocab:item_entry_value ?v .

  ?crf vocab:CRF_ItemGroup1757 ?itemgroup2 .
  ?itemgroup2 vocab:Itemgroup_Item3433 ?item2 .
  ?item2 vocab:item_entry_value ?v2 .

  ?crf vocab:CRF_ItemGroup1758 ?itemgroup3 .
  ?itemgroup3 vocab:Itemgroup_Item3434 ?item3 .
  ?item3 vocab:item_entry_value ?v3 .

  ?crf vocab:CRF_ItemGroup1690 ?itemgroup4 .
  ?itemgroup4 vocab:Itemgroup_Item3366 ?item4 .
  ?item4 vocab:item_entry_value ?v4 .

  ?crf2 vocab:CRF_ItemGroup1603 ?itemgroup5 .
  ?itemgroup5 vocab:Itemgroup_Item3279 ?item5 .
  ?item5 vocab:item_entry_value ?v5 .
}

```

Figure 14 The Obtima query in SPARQL after changing the D2RQ schema.

The SQL query produced for the SPARQL query that uses the optimised D2RQ mapping is smaller, but still more complex than need be. Appendix 3A and 3B show the SQL queries produced by D2RQ for respectively the SPARQL query in Figure 13 and Figure 14. Unfortunately, the performance problems are not entirely resolved, but performance seems to have improved. The query occasionally completes within seconds. However, the query does not always complete successfully; the database can still spend minutes at 100% CPU usage without completing the query for reasons that are still unclear. Also, a drawback of the extended D2RQ mapping that leads to improved performance is that it is specific to a particular trial for which Obtima has been used. The generic D2RQ mapping can be used for all trials, but each trial will need its own extended mapping. This is, however, not a serious drawback because the extended mapping can be generated fully automatically; no human involvement is required.

Summary

The relational data access services have been running for two years now. They have successfully been used in various scenarios. Nevertheless, there are still some issues. One set of problems is due to limitations of the D2RQ library that is used by the data access services to translate queries from SPARQL to SQL. There are two important classes of limitations.

First, there are limitations that affect the query results. These problems are the most serious, as these may lead to incorrect conclusions. The only known problem of this type is the incorrect handling of some OPTIONAL blocks, which means that fewer results are returned than should be [30]. This bug, however, has been fixed in the repository, and the fix will be included in the next release of D2RQ.

A second class of limitations are those that cause poor query execution performance, e.g. [31]. These prevent the execution of certain complex queries. This problem is less severe, as it is apparent when it occurs and can be worked around by splitting a complex query into smaller subqueries. Nevertheless, these limitations are troublesome and they will hopefully be addressed. However, due to the inherent complexity associated with translating SPARQL queries into efficient SQL queries [27], this may take a while.

There are also other, more minor, limitations, for example, ones for which good work arounds are available, e.g. [33] and [32]. All known problems in third-party libraries that affect the data access services are listed on the ACGT wiki⁸, so that we can easily track the outstanding issues. They have also been filed as bug reports, so that the developers of these third-party libraries are aware of them and can provide the corresponding fixes.

A more fundamental limitation of the SPARQL-query interface of the relational data access service is caused by limitations of the SPARQL query language. In particular, SPARQL lacks support for aggregation of data. That aggregation can be useful in practice can be seen in the Wilms microarray scenario. Out of the three queries on the Wilms CRF database that are needed to extract the required data, two use aggregation⁹. This limitation can be worked around by carrying out aggregation

⁸ http://wiki.healthgrid.org/ACGT:Wrappers_-_Known_problems

⁹ Next to the tumor histology query discussed in Section 5.2.1, a query is needed for extracting basic patient info, and another query for extracting survival information obtained during follow-ups. These

client-side (e.g. in the R scripts that carry out the data-mining algorithms). However, this does require additional effort by the bio-informatician that carries out the analysis and has to create the R scripts.

5.2.2 Data access service for DICOM image data

The current functionality of the DICOM data access services lets users retrieve images associated with given patients, which is a common need when accessing imaging data. Furthermore, the SPARQL-query interface provides a uniform interface for carrying out simple queries within a given trial. The query interface is, however, not intended for carrying out complex queries across images from many different trials. If this functionality is needed, an approach similar to that of NCIA, an Open Source solution provided by caBIG, is needed. Such a system could be extended and integrated into the ACGT platform, for example to use ACGT's authorization and authentication approach for controlling access and to support delivery of images to DMS.

Implementing the data access services for DICOM data alongside those for relational data has been a good test-case for the feasibility of providing a uniform query interface. DICOM query functionality is very basic, and significantly more limited than both SPARQL and SQL. Resolving the differences in query functionality and query capabilities of databases is always a big challenge when attempting to provide a uniform interface over a set of heterogeneous databases. The particular problems that we have encountered with the DICOM data access service have been documented and discussed in [2] and [29].

5.2.3 Data access service for microarray data

The functionality offered by the data access services for the BASE databases is relatively simple, yet sufficient for all the data-mining use cases we have considered so far. Most importantly, it offers secure access to the microarray data as the data access services have been fully integrated into the ACGT security infrastructure. The files can be delivered to DMS, after optional compression and archiving into a single file for efficiency, where they can subsequently be processed by the data-mining scripts.

5.2.4 Dynamic deployment of data access services

We have recently provided users with the ability to dynamically deploy new data sources into the ACGT platform. Users can integrate new data sources from the ACGT portal and can subsequently query these directly from the workflow enactor.

So far, we have only done so for the relational data access services, as this is the most useful. Similar functionality could be provided for the other data access services, but there is less of a need. Dynamic creation of relational data access services is particularly useful because a lot of different types of data can be made available this way, including data from Excel spreadsheets and text files in CSV format. Furthermore, integration of new data sources requires creating a mapping

queries are available in SVN, at respectively
<https://iapetus.ics.forth.gr/svn/ACGT/WrapperServices/trunk/examples/perform/wilms/PatientInfo.xml>
and
<https://iapetus.ics.forth.gr/svn/ACGT/WrapperServices/trunk/examples/perform/wilms/FirstSurvivalEvent.xml>

from SPARQL to SQL, which is best done by someone with good familiarity of the schema of the database and the data contained in it. In contrast, there is no content-specific knowledge required to integrate new DICOM and microarray databases. Furthermore, the data they store is much more specific and new databases appear less frequently (generation of new data in both cases requires expensive equipment, is time-consuming and always involves patients or tissue samples), which means that static deployment of these data access services is in practice sufficient.

Currently the access rules for all dynamically created data resources are the same. Each resource is accessible by anybody with valid ACGT credentials. We plan to extend the functionality so that by default newly created data resources are only accessible to the owner (i.e. the person that created the resource). Subsequently the owner can change the access rules to also grant access to other persons (either one-by-one by explicitly naming each person that is allowed access, or in batches by granting access to anyone that belongs to a particular Virtual Organization). This improved functionality would have the benefit that dynamic data resources can also be created for databases that contain data with more restrictions on access.

5.2.5 Open issues

We have implemented OGSA-DAI data access services for three types of data sources: relational databases, medical image databases and a micro-array database. The main research question is how to best provide a syntactically homogeneous interface, and a key question is the query language that is used. We have chosen SPARQL as the common query language and have demonstrated that it can be successfully applied to relational databases and DICOM image databases.

For the relational databases, the SPARQL language does not support all features offered by the query language of the data source, SQL. For instance, it does not support aggregation of data (averaging, summation, counting, etc). So aggregation needs to be performed at the client-side, even though the underlying database supports it directly, which negatively affects performance. The actual use of the system by the end users will clarify whether this is a problem that needs to be addressed.

For medical image databases, SPARQL is more expressive than the query support provided by the DICOM protocol. For this reason, the data access service does not support all queries. These limitations are currently described as text, but should be expressed in a more formal manner, so that other services and applications can interpret these and handle accordingly. In order to select a suitable formal framework for this, we need to thoroughly review the capabilities and limitations of all relevant data sources.

A capability-restricted data access architecture has the advantage that it is easier to develop data access services for data sources; as a consequence, new data sources can be integrated much more quickly. It may, however, complicate applications and services that use the data access services. A higher level data access service may therefore be introduced that hides query restrictions of the underlying services. This generic service would decompose queries for a specific data access service as needed, store the intermediate results, and join these to produce the final answer. This would facilitate implementation of the semantic mediator, while incurring a slight performance penalty. However, this higher-level data access service may also carry

out generic optimizations such as caching of query results, resulting in overall performance gains.

Another open issue is how to provide text-based query functionality. There are many public biomedical databases where part of the data is free text. Examples are descriptions of microarray experiments (e.g. in GEO [34] and ArrayExpress [35]), descriptions of gene and protein functions (e.g. in UniProt [36] and EntrezGene [37]), and abstracts and titles of medical publications (e.g. in PubMed [38]). Although most databases provide keyword-based functionality for querying data, this method of searching is not directly supported by SPARQL, so it is not immediately obvious how to extend the current data access services interface to support this functionality. One approach would be to add a separate text-based query interface for data sources that support text-based queries. This exposes more details of the underlying data source, resulting in a less homogeneous interface. This is undesirable but may be unavoidable in practice. However, there is a more important question that needs to be answered first: how should querying of text data be handled by the semantic layer? This is an important question as it determines the query interface that is available to end-users, but answering it falls outside the scope of this report.

6 References

- [1] Erwin Bonsma and Anca Bucur (Editors), Deliverable 5.1: "Consolidated requirements and specifications for data access", 15 January 2007
- [2] Erwin Bonsma (Editor), Deliverable 5.2: "Heterogeneous data access services", 19 December 2007
- [3] Fatima Schera and Gabriele Weiler (Editors), Deliverable 5.4, "Conceptual specification and a first prototype for an ontology based Clinical Data Management System and for the Trial Builder", 26 February 2008
- [4] Manolis Tsiknakis (Editor), Deliverable 2.1: "User requirements and specification of the ACGT internal clinical trial", 13 September 2006
- [5] Luis Martín (Editor), Deliverable 7.5: "Demonstration of final mediation access tools and services", 27 November 2008
- [6] Brecht Claerhout and Stefan Castille (Editors), Deliverable 11.1: "Consolidation of security requirements of ACGT and initial security architecture", 1 February 2007
- [7] Brecht Claerhout and Stefan Castille (Editors), Deliverable 11.2: "Implementation of the ACGT core security services & initial implementation of the Pseudonymisation tool", 17 December 2007
- [8] Thierry Sengstag (Editor), Deliverable 13.3: "Specification of scenarios for the first integrated demonstrator of the ACGT platform", 15 May 2008
- [9] Thierry Sengstag (Editor), Deliverable 13.4: "April 2009 Demonstrator Specifications", 14 April 2009
- [10] Bizer, C., Seaborne, A.: D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In: Proc. of the 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, 2004. Poster presentation
- [11] Chris Bizer, Richard Cyganiak, Jörg Garbers, Oliver Maresch and Christian Becker. "The D2RQ Platform v0.6 - Treating Non-RDF Relational Databases as Virtual RDF Graphs, User Manual and Language Specification", <http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/>
- [12] Marcel van Herk, "Conquest DICOM software", <http://www.xs4all.nl/~ingenium/dicom.html>
- [13] Max J. Warnock, Christopher Toland, Damien Evans, Bill Wallace and Paul Nagy. "Benefits of Using the DCM4CHE DICOM Archive". J Digit Imaging. 2007 November; 20(Suppl 1): 125–129. doi: 10.1007/s10278-007-9064-1.
- [14] dcm4che.org, "Open Source Clinical Image and Object Management", <http://www.dcm4che.org/>
- [15] UCLA Medical Imaging Informatics, "OpenSourcePACS -An Open Source PACS Framework", <http://www.mii.ucla.edu/opensourcepacs/>
- [16] RainbowFish Software, "PacsOne server", <http://www.pacsone.net/>
- [17] caBIG, "National Biomedical Imaging Archive", <https://cabig.nci.nih.gov/tools/NCIA>

- [18] Sun Microsystems, MySQL, <http://www.mysql.com/>
- [19] Sun Microsystems, "MySQL Documentation", <http://dev.mysql.com/doc/>
- [20] PostgreSQL Global Development Group, PostgreSQL, <http://www.postgresql.org/>
- [21] PostgreSQL Global Development Group, "PostgreSQL Documentation", <http://www.postgresql.org/docs/>
- [22] Russell J.T. Dyer, "MySQL in a Nutshell", O'Reilly, 2nd Edition, April 2008
- [23] John C. Worsley, Joshua D. Drake, "Practical PostgreSQL", O'Reilly, 1st Edition, January 2002
- [24] Saal LH, Troein C, Vallon-Christersson J, et al (2002) BioArray Software Environment: A Platform for Comprehensive Management and Analysis of Microarray Data. Genome Biol-ogy 2002 3(8): software0003.1-0003.6
- [25] Lund University, "BASE - BioArray Software Environment" <http://base.thep.lu.se/>
- [26] Brian Hatch, "Stunnel - Universal SSL Wrapper", <http://www.stunnel.org/>
- [27] Artem Chebotko, Shiyong Lu, Hasan M. Jamil and Farshad Fotouhi, "Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns", Technical Report TR-DB-052006-CLJF, May 2006. Revised November 2006.
- [28] B. Zirn et al., "Expression profiling of Wilms tumors reveals new candidate genes for different parameters", Int. J. Cancer: 118, 1954-1962 (2006)
- [29] Erwin Bonsma and Jeroen Vrijnsen, "Homogenising access to heterogeneous biomedical data sources", Proc. of the BMIINT AIAI 2009 Workshop, April 2009
- [30] D2RQ Bug 2808874, "Queries with OPTIONAL blocks returning too few results", https://sourceforge.net/tracker/?func=detail&aid=2808874&group_id=111002&atid=657968
- [31] D2RQ Bug 2798308, "Unnecessary self-joins in generated SQL queries", https://sourceforge.net/tracker/?func=detail&aid=2798308&group_id=111002&atid=657968
- [32] D2RQ Bug 1801864: "EOFException on database connection not handled", https://sourceforge.net/tracker/index.php?func=detail&aid=1801864&group_id=111002&atid=657968
- [33] D2RQ Bug 1794042: "Bug in D2RQ for properties that use more than one join", https://sourceforge.net/tracker/index.php?func=detail&aid=1794042&group_id=111002&atid=657968
- [34] Gene Expression Omnibus (GEO), <http://www.ncbi.nlm.nih.gov/geo/>
- [35] ArrayExpress, <http://www.ebi.ac.uk/microarray-as/ae/>
- [36] Uniprot, <http://www.uniprot.org/>
- [37] Entrez Gene, <http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene>.
- [38] PubMed, <http://www.ncbi.nlm.nih.gov/pubmed/>.

- [39] Oracle Clinical, http://www.oracle.com/industries/life_sciences/oracle-clinical.html
- [40] OpenClinica, "Open Source for Clinical Research", <http://www.openclinica.org/>
- [41] caArray, <http://caarray.nci.nih.gov/caARRAY>
- [42] Gattiker et al, "MIMAS 3.0 is a Multiomics Information Management and Annotation System", BMC Bioinformatics 2009, 10:151, doi:10.1186/1471-2105-10-151
- [43] Maurer et al, "MARS: Microarray analysis, retrieval, and storage system", BMC Bioinformatics 2005, 6:101 doi:10.1186/1471-2105-6-101
- [44] Institute for Systems Biology, Systems Biology Experiment Analysis Management System (SBEAMS), <http://www.sbeams.org/>

Appendix 1 - Abbreviations and acronyms

This glossary lists various acronyms that are used throughout the deliverable. It does, however, not include all acronyms that are used. Acronyms that are only introduced and used in a particular section, and not referred to subsequently, are typically excluded.

<i>API</i>	Application Programming Interface. The public interface provided by libraries and services.
<i>BASE</i>	BioArray Software Environment. The database for storing microarray data that is used within ACGT.
<i>CAT</i>	Custodix Anonymisation Tool. The tool used within ACGT to anonymise and pseudonymise clinical-trial data.
<i>CTMS</i>	Clinical Trial Management System.
<i>CRF</i>	Case Report Form.
<i>CSV</i>	Comma Separated Values. A simple textfile format for structured, tabular data.
<i>D2RQ</i>	A platform for accessing non-RDF, relational databases as virtual, read-only RDF graphs
<i>DICOM</i>	Digital Imaging and Communications in Medicine. A standard for exchanging medical data.
<i>DMS</i>	Data Management System. The grid-based file storage system that is used in ACGT.
<i>GAS</i>	Grid Authentication Server. The authentication server that is used within ACGT
<i>JDBC</i>	Java Database Connection. A Java API for database access
<i>LIMS</i>	Laboratory Information Management System.
<i>OGSA</i>	Open Grid Services Architecture
<i>OGSA-DAI</i>	OGSA standard for Data Access and Integration. A middleware product that supports the exposure of data sources onto the grid.
<i>PACS</i>	Picture Archiving and Communication System. Generic term for medical imaging databases.
<i>RDBMS</i>	Relational Database Management System.
<i>RDF</i>	Resource Description Framework. A language for representing information about resources.

<i>SCU</i>	Service Class User. DICOM term for the client of a service
<i>SCP</i>	Service Class Provider. DICOM term for the provider of a service.
<i>SPARQL</i>	A query language for RDF.
<i>SQL</i>	Standard Query Language. The most popular query language for relational databases.
<i>SVN</i>	Subversion, the version control system used within ACGT.
<i>URI</i>	Uniform Resource Identifier. A string of characters that identifies or names an object on the Internet. It is a generalisation of URL.
<i>URL</i>	Uniform Resource Locator. A type of URI that specifies where a resource is available, and the mechanism for retrieving it.
<i>XML</i>	Extensible Markup Language. The format that is used by web services to exchange data.

Appendix 2A – Initial example D2RQ mapping

The initial D2RQ mapping from RDF to SQL for the example used in Section 4.1.2.

```

@prefix map: <file:/C:/d2rq/d2rq-0.6/d2rq_example.n3#> .
@prefix db: <> .
@prefix vocab: <http://example.philips.com/vocab/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
.
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

map:database a d2rq:Database;
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
  d2rq:jdbcDSN "jdbc:mysql://localhost/d2rq_example";
  d2rq:username "readonly";
  d2rq:password "readonly";
  jdbc:autoReconnect "true";
  jdbc:zeroDateTimeBehavior "convertToNull";
.

# Table codesex
map:codesex a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "codesex/@@codesex.id@";
  d2rq:class vocab:codesex;
  d2rq:classDefinitionLabel "codesex";
.

map:codesex__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:codesex;
  d2rq:property rdfs:label;
  d2rq:pattern "codesex #@@codesex.id@";
.

map:codesex_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:codesex;
  d2rq:property vocab:codesex_id;
  d2rq:propertyDefinitionLabel "codesex id";
  d2rq:column "codesex.id";
  d2rq:datatype xsd:int;
.

map:codesex_sex a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:codesex;
  d2rq:property vocab:codesex_sex;
  d2rq:propertyDefinitionLabel "codesex sex";
  d2rq:column "codesex.sex";
.

# Table crf_postsurgery
map:crf_postsurgery a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "crf_postsurgery/@@crf_postsurgery.id@";
  d2rq:class vocab:crf_postsurgery;
  d2rq:classDefinitionLabel "crf_postsurgery";
.

```

```
map:crf_postsurgery__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:crf_postsurgery;
  d2rq:property rdfs:label;
  d2rq:pattern "crf_postsurgery #@@crf_postsurgery.id@";
.
map:crf_postsurgery_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:crf_postsurgery;
  d2rq:property vocab:crf_postsurgery_id;
  d2rq:propertyDefinitionLabel "crf_postsurgery id";
  d2rq:column "crf_postsurgery.id";
  d2rq:datatype xsd:int;
.
map:crf_postsurgery_entry_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:crf_postsurgery;
  d2rq:property vocab:crf_postsurgery_entry_id;
  d2rq:propertyDefinitionLabel "crf_postsurgery entry_id";
  d2rq:column "crf_postsurgery.entry_id";
  d2rq:datatype xsd:int;
.
map:crf_postsurgery_rdx a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:crf_postsurgery;
  d2rq:property vocab:crf_postsurgery_rdx;
  d2rq:propertyDefinitionLabel "crf_postsurgery rdx";
  d2rq:column "crf_postsurgery.rdx";
  d2rq:datatype xsd:int;
.
# Table entry
map:entry a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "entry/@@entry.id@";
  d2rq:class vocab:entry;
  d2rq:classDefinitionLabel "entry";
.
map:entry__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:entry;
  d2rq:property rdfs:label;
  d2rq:pattern "entry #@@entry.id@";
.
map:entry_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:entry;
  d2rq:property vocab:entry_id;
  d2rq:propertyDefinitionLabel "entry id";
  d2rq:column "entry.id";
  d2rq:datatype xsd:int;
.
map:entry_patient_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:entry;
  d2rq:property vocab:entry_patient_id;
  d2rq:propertyDefinitionLabel "entry patient_id";
  d2rq:column "entry.patient_id";
  d2rq:datatype xsd:int;
.
map:entry_entry_date a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:entry;
  d2rq:property vocab:entry_entry_date;
```

```
d2rq:propertyDefinitionLabel "entry entry_date";
d2rq:column "entry.entry_date";
d2rq:datatype xsd:date;
.

# Table patient
map:patient a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "patient/@@patient.id@";
  d2rq:class vocab:patient;
  d2rq:classDefinitionLabel "patient";
.
map:patient__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:patient;
  d2rq:property rdfs:label;
  d2rq:pattern "patient #@@patient.id@";
.
map:patient_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:patient;
  d2rq:property vocab:patient_id;
  d2rq:propertyDefinitionLabel "patient id";
  d2rq:column "patient.id";
  d2rq:datatype xsd:int;
.
map:patient_sex_id a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:patient;
  d2rq:property vocab:patient_sex_id;
  d2rq:propertyDefinitionLabel "patient sex_id";
  d2rq:column "patient.sex_id";
  d2rq:datatype xsd:int;
.
map:patient_dob a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:patient;
  d2rq:property vocab:patient_dob;
  d2rq:propertyDefinitionLabel "patient dob";
  d2rq:column "patient.dob";
  d2rq:datatype xsd:date;
.
```

Appendix 2B – Final example D2RQ Mapping

The final D2RQ mapping from RDF to SQL after applying all changes described in Section 4.1.2.

```

@prefix map: <file:/C:/d2rq/d2rq-0.6/d2rq_example.n3#> .
@prefix db: <> .
@prefix vocab: <http://example.philips.com/vocab/> .
@prefix code: <http://example.philips.com/codes#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
.
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

map:database a d2rq:Database;
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
  d2rq:jdbcDSN "jdbc:mysql://localhost/d2rq_example";
  d2rq:username "readonly";
  d2rq:password "readonly";
  jdbc:autoReconnect "true";
  jdbc:zeroDateTimeBehavior "convertToNull";
.

# Table codesex
code:Female a vocab:code_sex .
code:Male a vocab:code_sex .

map:SexCodes a d2rq:TranslationTable;
  d2rq:translation [ d2rq:databaseValue "Vrouw";
                    d2rq:rdfValue code:Female; ];
  d2rq:translation [ d2rq:databaseValue "Man";
                    d2rq:rdfValue code:Male; ];
.

map:code_sex a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriColumn "codesex.sex";
  d2rq:translateWith map:SexCodes;
  d2rq:class vocab:code_sex;
.

# Table crf_postsurgery
map:crf_postsurgery a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "crf_postsurgery/@@crf_postsurgery.id@";
  d2rq:class vocab:crf_postsurgery;
  d2rq:classDefinitionLabel "crf_postsurgery";
.

map:crf_postsurgery__label a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:crf_postsurgery;
  d2rq:property rdfs:label;
  d2rq:pattern "crf_postsurgery #@@crf_postsurgery.id@";
.

map:crf_postsurgery_id a d2rq:PropertyBridge;

```

```
d2rq:belongsToClassMap map:crf_postsurgery;
d2rq:property vocab:crf_postsurgery_id;
d2rq:propertyDefinitionLabel "crf_postsurgery id";
d2rq:column "crf_postsurgery.id";
d2rq:datatype xsd:int;
.
map:crf_postsurgery_entry_date a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:crf_postsurgery;
d2rq:property vocab:crf_postsurgery_entry_date;
d2rq:propertyDefinitionLabel "crf_postsurgery entry_date";
d2rq:column "entry.entry_date";
d2rq:join "crf_postsurgery.entry_id = entry.id";
d2rq:datatype xsd:date;
.
map:crf_postsurgery_patient a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:crf_postsurgery;
d2rq:property vocab:crf_postsurgery_patient;
d2rq:propertyDefinitionLabel "crf_postsurgery patient";
d2rq:refersToClassMap map:patient;
d2rq:join "crf_postsurgery.entry_id = entry.id";
d2rq:join "entry.patient_id = patient.id";
.
map:crf_postsurgery_rdx a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:crf_postsurgery;
d2rq:property vocab:crf_postsurgery_radiotherapy;
d2rq:propertyDefinitionLabel "crf_postsurgery rdx";
d2rq:column "crf_postsurgery.rdx";
d2rq:datatype xsd:boolean;
.
# Table patient
map:patient a d2rq:ClassMap;
d2rq:dataStorage map:database;
d2rq:uriPattern "patient/@@patient.id@";
d2rq:class vocab:patient;
d2rq:classDefinitionLabel "patient";
.
map:patient__label a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:patient;
d2rq:property rdfs:label;
d2rq:pattern "patient #@@patient.id@";
.
map:patient_id a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:patient;
d2rq:property vocab:patient_id;
d2rq:propertyDefinitionLabel "patient id";
d2rq:column "patient.id";
d2rq:datatype xsd:int;
.
map:patient_sex a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:patient;
d2rq:property vocab:patient_sex;
d2rq:propertyDefinitionLabel "patient sex";
d2rq:refersToClassMap map:code_sex;
d2rq:join "patient.sex_id = codesex.id";
.
```

```
map:patient_dob a d2rq:PropertyBridge;  
  d2rq:belongsToClassMap map:patient;  
  d2rq:property vocab:patient_dob;  
  d2rq:propertyDefinitionLabel "patient dob";  
  d2rq:column "patient.dob";  
  d2rq:datatype xsd:date;  
  .
```

Appendix 3A – SQL query produced for initial Obtima SPARQL query

Below is the SQL query that is produced by D2RQ for the SPARQL query in Figure 13

```

SELECT DISTINCT "T4_itemgroup_entry"."itemgroupentry_id",
"T18_item_entry"."value", "T21_itemgroup_entry"."itemgroupentry_id",
"T10_item_entry"."value", "T1_patient"."patient_id",
"T26_item_entry"."value", "T2_itemgroup_entry"."itemgroupentry_id",
"T9_itemgroup_entry"."itemgroupentry_id",
"T6_itemgroup_entry"."itemgroupentry_id",
"T25_itemgroup_entry"."itemgroupentry_id",
"T27_item_entry"."item_template_fk", "T14_item_entry"."value",
"T20_item_entry"."itementry_id",
"T17_itemgroup_entry"."itemgroupentry_id",
"T23_item_entry"."item_template_fk",
"T13_itemgroup_entry"."itemgroupentry_id",
"T15_item_entry"."item_template_fk", "T22_item_entry"."value",
"T5_itemgroup_entry"."itemgroupentry_id",
"T7_crf_instance"."crf_instance_fk", "T11_item_entry"."item_template_fk",
"T1_crf_instance"."crf_instance_id", "T8_item_entry"."itementry_id",
"T24_item_entry"."itementry_id", "T19_item_entry"."item_template_fk",
"T16_item_entry"."itementry_id", "T3_itemgroup_entry"."itemgroupentry_id",
"T12_item_entry"."itementry_id" FROM "itemgroup_entry" AS
"T21_itemgroup_entry", "patient" AS "T1_patient", "itemgroup_entry" AS
"T3_itemgroup_entry", "crf_instance" AS "T7_crf_instance", "crf_instance"
AS "T5_crf_instance", "crf_instance" AS "T4_crf_instance",
"itemgroup_entry" AS "T25_itemgroup_entry", "itemgroup_entry" AS
"T24_itemgroup_entry", "item_entry" AS "T22_item_entry", "item_entry" AS
"T16_item_entry", "itemgroup_entry" AS "T5_itemgroup_entry",
"itemgroup_entry" AS "T17_itemgroup_entry", "item_entry" AS
"T12_item_entry", "item_entry" AS "T11_item_entry", "itemgroup_entry" AS
"T20_itemgroup_entry", "item_entry" AS "T23_item_entry", "crf_instance" AS
"T3_crf_instance", "item_entry" AS "T8_item_entry", "itemgroup_entry" AS
"T2_itemgroup_entry", "itemgroup_entry" AS "T13_itemgroup_entry",
"itemgroup_entry" AS "T16_itemgroup_entry", "item_entry" AS
"T15_item_entry", "itemgroup_entry" AS "T6_itemgroup_entry", "item_entry"
AS "T18_item_entry", "itemgroup_entry" AS "T8_itemgroup_entry",
"item_entry" AS "T14_item_entry", "item_entry" AS "T27_item_entry",
"item_entry" AS "T26_item_entry", "item_entry" AS "T19_item_entry",
"crf_instance" AS "T1_crf_instance", "itemgroup_entry" AS
"T4_itemgroup_entry", "itemgroup_entry" AS "T9_itemgroup_entry",
"itemgroup_entry" AS "T12_itemgroup_entry", "item_entry" AS
"T10_item_entry", "item_entry" AS "T24_item_entry", "crf_instance" AS
"T2_crf_instance", "crf_instance" AS "T6_crf_instance", "item_entry" AS
"T20_item_entry" WHERE ("T10_item_entry"."itementry_id" =
"T11_item_entry"."itementry_id" AND "T11_item_entry"."itementry_id" =
"T8_item_entry"."itementry_id" AND "T12_item_entry"."itementry_id" =
"T15_item_entry"."itementry_id" AND "T12_item_entry"."itemgroupentry_fk" =
"T12_itemgroup_entry"."itemgroupentry_id" AND
"T12_itemgroup_entry"."itemgroupentry_id" =
"T13_itemgroup_entry"."itemgroupentry_id" AND
"T13_itemgroup_entry"."itemgroupentry_id" =
"T3_itemgroup_entry"."itemgroupentry_id" AND
"T14_item_entry"."itementry_id" = "T15_item_entry"."itementry_id" AND
"T16_item_entry"."itementry_id" = "T18_item_entry"."itementry_id" AND
"T16_item_entry"."itementry_id" = "T19_item_entry"."itementry_id" AND
"T16_item_entry"."itemgroupentry_fk" =
"T16_itemgroup_entry"."itemgroupentry_id" AND
"T16_itemgroup_entry"."itemgroupentry_id" =
"T17_itemgroup_entry"."itemgroupentry_id" AND

```



```
"T16_itemgroup_entry"."itemgroupentry_id" =
"T4_itemgroup_entry"."itemgroupentry_id" AND
"T1_crf_instance"."crfinstance_id" = "T3_crf_instance"."crfinstance_id" AND
"T1_crf_instance"."patient_fk" = "T1_patient"."patient_id" AND
"T20_item_entry"."itementry_id" = "T22_item_entry"."itementry_id" AND
"T20_item_entry"."itementry_id" = "T23_item_entry"."itementry_id" AND
"T20_item_entry"."itemgroupentry_fk" =
"T20_itemgroup_entry"."itemgroupentry_id"AND
"T20_itemgroup_entry"."itemgroupentry_id" =
"T21_itemgroup_entry"."itemgroupentry_id" AND
"T20_itemgroup_entry"."itemgroupentry_id" =
"T5_itemgroup_entry"."itemgroupentry_id" AND
"T24_item_entry"."itementry_id" = "T27_item_entry"."itementry_id" AND
"T24_item_entry"."itemgroupentry_fk" =
"T24_itemgroup_entry"."itemgroupentry_id" AND
"T24_itemgroup_entry"."itemgroupentry_id" =
"T25_itemgroup_entry"."itemgroupentry_id" AND
"T25_itemgroup_entry"."itemgroupentry_id" =
"T6_itemgroup_entry"."itemgroupentry_id" AND
"T26_item_entry"."itementry_id" = "T27_item_entry"."itementry_id" AND
"T2_crf_instance"."crfinstance_id" = "T2_itemgroup_entry"."crf_instance_fk"
AND "T2_crf_instance"."crfinstance_id" = "T3_crf_instance"."crfinstance_id"
AND "T2_itemgroup_entry"."itemgroupentry_id" =
"T9_itemgroup_entry"."itemgroupentry_id" AND
"T3_crf_instance"."crfinstance_id" = "T3_itemgroup_entry"."crf_instance_fk"
AND "T3_crf_instance"."crfinstance_id" = "T4_crf_instance"."crfinstance_id"
AND "T3_crf_instance"."crfinstance_id" = "T5_crf_instance"."crfinstance_id"
AND "T3_crf_instance"."crfinstance_id" = "T6_crf_instance"."crfinstance_id"
AND "T3_crf_instance"."crfinstance_id" = "T7_crf_instance"."crfinstance_id"
AND "T4_crf_instance"."crfinstance_id" =
"T4_itemgroup_entry"."crf_instance_fk" AND
"T5_crf_instance"."crfinstance_id" = "T5_itemgroup_entry"."crf_instance_fk"
AND "T6_crf_instance"."crfinstance_id" =
"T6_itemgroup_entry"."crf_instance_fk" AND
"T8_item_entry"."itemgroupentry_fk" =
"T8_itemgroup_entry"."itemgroupentry_id" AND
"T8_itemgroup_entry"."itemgroupentry_id" =
"T9_itemgroup_entry"."itemgroupentry_id" AND
("T11_item_entry"."item_template_fk" = 3432) AND
("T13_itemgroup_entry"."itemgrouptemplate_fk" = 1757) AND
("T15_item_entry"."item_template_fk" = 3433) AND
("T17_itemgroup_entry"."itemgrouptemplate_fk" = 1758)
AND("T19_item_entry"."item_template_fk" = 3434) AND
("T21_itemgroup_entry"."itemgrouptemplate_fk" = 1690) AND
("T23_item_entry"."item_template_fk" = 3366) AND
("T25_itemgroup_entry"."itemgrouptemplate_fk" = 1603) AND
("T27_item_entry"."item_template_fk" = 3279) AND
("T7_crf_instance"."crf_template_fk" = 512) AND
("T9_itemgroup_entry"."itemgrouptemplate_fk" = 1756))
```

Appendix 3B - SQL query produced for refined Obtima SPARQL query

Below is the SQL query that is produced by D2RQ for the SPARQL query in Figure 14

```

SELECT DISTINCT "T9_itemgroup_entry"."itemgroupentry_id",
"T14_item_entry"."value", "T13_item_entry"."itementry_id",
"T17_item_entry"."value", "T4_item_entry"."itementry_id",
"T2_crf_instance"."crfinstance_id", "T1_patient"."patient_id",
"T11_item_entry"."value", "T1_crf_instance"."crfinstance_id",
"T10_item_entry"."itementry_id", "T15_itemgroup_entry"."itemgroupentry_id",
"T8_item_entry"."value", "T6_itemgroup_entry"."itemgroupentry_id",
"T5_item_entry"."value", "T7_item_entry"."itementry_id",
"T3_itemgroup_entry"."itemgroupentry_id", "T16_item_entry"."itementry_id",
"T12_itemgroup_entry"."itemgroupentry_id" FROM "itemgroup_entry" AS
"T7_itemgroup_entry", "patient" AS "T1_patient", "patient" AS "T2_patient",
"item_entry" AS "T7_item_entry", "itemgroup_entry" AS "T3_itemgroup_entry",
"item_entry" AS "T5_item_entry", "item_entry" AS "T16_item_entry",
"item_entry" AS "T11_item_entry", "crf_instance" AS "T3_crf_instance",
"item_entry" AS "T8_item_entry", "itemgroup_entry" AS
"T13_itemgroup_entry", "itemgroup_entry" AS "T16_itemgroup_entry",
"itemgroup_entry" AS "T6_itemgroup_entry", "item_entry" AS
"T13_item_entry", "item_entry" AS "T14_item_entry", "item_entry" AS
"T17_item_entry", "crf_instance" AS "T1_crf_instance", "itemgroup_entry" AS
"T4_itemgroup_entry", "item_entry" AS "T4_item_entry", "itemgroup_entry" AS
"T9_itemgroup_entry", "crf_instance" AS "T15_crf_instance",
"itemgroup_entry" AS "T15_itemgroup_entry", "itemgroup_entry" AS
"T12_itemgroup_entry", "item_entry" AS "T10_item_entry", "crf_instance" AS
"T2_crf_instance", "itemgroup_entry" AS "T10_itemgroup_entry",
"crf_instance" AS "T6_crf_instance", "crf_instance" AS "T9_crf_instance",
"crf_instance" AS "T12_crf_instance" WHERE ("T10_item_entry"."itementry_id"
= "T11_item_entry"."itementry_id" AND "T10_item_entry"."itemgroupentry_fk"
= "T10_itemgroup_entry"."itemgroupentry_id" AND
"T10_itemgroup_entry"."itemgroupentry_id" =
"T9_itemgroup_entry"."itemgroupentry_id" AND
"T12_crf_instance"."crfinstance_id" =
"T12_itemgroup_entry"."crf_instance_fk" AND
"T12_crf_instance"."crfinstance_id" = "T3_crf_instance"."crfinstance_id"
AND "T12_itemgroup_entry"."itemgroupentry_id" =
"T13_itemgroup_entry"."itemgroupentry_id" AND
"T13_item_entry"."itementry_id" = "T14_item_entry"."itementry_id" AND
"T13_item_entry"."itemgroupentry_fk" =
"T13_itemgroup_entry"."itemgroupentry_id" AND
"T15_crf_instance"."crfinstance_id" =
"T15_itemgroup_entry"."crf_instance_fk" AND
"T15_crf_instance"."crfinstance_id" = "T2_crf_instance"."crfinstance_id"
AND "T15_itemgroup_entry"."itemgroupentry_id" =
"T16_itemgroup_entry"."itemgroupentry_id" AND
"T16_item_entry"."itementry_id" = "T17_item_entry"."itementry_id" AND
"T16_item_entry"."itemgroupentry_fk" =
"T16_itemgroup_entry"."itemgroupentry_id" AND
"T1_crf_instance"."crfinstance_id" = "T3_crf_instance"."crfinstance_id" AND
"T1_crf_instance"."patient_fk" = "T1_patient"."patient_id" AND
"T1_patient"."patient_id" = "T2_patient"."patient_id" AND
"T2_crf_instance"."patient_fk" = "T2_patient"."patient_id" AND
"T3_crf_instance"."crfinstance_id" = "T3_itemgroup_entry"."crf_instance_fk"
AND "T3_crf_instance"."crfinstance_id" = "T6_crf_instance"."crfinstance_id"
AND "T3_crf_instance"."crfinstance_id" = "T9_crf_instance"."crfinstance_id"
AND "T3_itemgroup_entry"."itemgroupentry_id" =
"T4_itemgroup_entry"."itemgroupentry_id" AND "T4_item_entry"."itementry_id"
= "T5_item_entry"."itementry_id" AND "T4_item_entry"."itemgroupentry_fk" =

```

```
"T4_itemgroup_entry"."itemgroupentry_id" AND
"T6_crf_instance"."crfinstance_id" = "T6_itemgroup_entry"."crf_instance_fk"
AND "T6_itemgroup_entry"."itemgroupentry_id" =
"T7_itemgroup_entry"."itemgroupentry_id" AND "T7_item_entry"."itementry_id"
= "T8_item_entry"."itementry_id" AND "T7_item_entry"."itemgroupentry_fk" =
"T7_itemgroup_entry"."itemgroupentry_id" AND
"T9_crf_instance"."crfinstance_id" = "T9_itemgroup_entry"."crf_instance_fk"
AND ("T10_item_entry"."item_template_fk" = 3434) AND
("T12_itemgroup_entry"."itemgrouptemplate_fk" = 1690) AND
("T13_item_entry"."item_template_fk" = 3366) AND
("T15_itemgroup_entry"."itemgrouptemplate_fk" = 1603) AND
("T16_item_entry"."item_template_fk" = 3279) AND
("T1_crf_instance"."crf_template_fk" = 512) AND
("T2_crf_instance"."crf_template_fk" = 511) AND
("T3_itemgroup_entry"."itemgrouptemplate_fk" = 1756) AND
("T4_item_entry"."item_template_fk" = 3432) AND
("T6_itemgroup_entry"."itemgrouptemplate_fk" = 1757) AND
("T7_item_entry"."item_template_fk" = 3433) AND
("T9_itemgroup_entry"."itemgrouptemplate_fk" = 1758))
```