Demonstration and Report of the Ontology Mediation Services

Project Number:	FP6-2005-IST-026996
Deliverable id:	D7.3
Deliverable name:	Demonstration and Report of the Ontology Mediation Services
Submission Date:	18/04/2008

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:	D7.3
Document name:	Demonstration and Report of the Ontology Mediation Services
Document type (PU, INT, RE)	RE
Version:	1.0
Submission date:	18/04/2008
Editor: Organisation: Email:	Luis Martín Universidad Politécnica de Madrid lmartin@infomed.dia.fi.upm.es

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

ABSTRACT:

This deliverable gives a description of the ontology mediation tools developed within WP7, that aim at solving semantic heterogeneities in data access, at achieving database integration, at supporting the clinical trial building process, and at navigating the ontology. This is a technical document focused in describing the services offered by the different tools that have been developed, and in identifying their user base. The selected scientific approaches for the implementation of these technologies are described as well.

Together with the ACGT Master Ontology on Cancer, a collection of software tools have been developed. These tools have the goal of solving semantic heterogeneity and providing database integration. The ACGT Semantic Mediator acts as the coordinator of all the tools, and offer the main querying services to the clients.

At the end of the document, a section regarding the main usages of the ACGT Master Ontology on Cancer has been included. This section also describes the Obtima tool.

KEYWORD LIST: clinical trials, database integration, web services, semantic mediation, ontologies

MODIFICATION CONTROL			
Version	**Date**	**Status**	**Editor**
0.1	26/11/2007	Internal Draft Version	Luis Martín
0.2	30/11/2007	Draft Version	Luis Martín
0.9	12/02/2008	Pre-final	Luis Martín
1.0	18/04/2008	Final	Luis Martín

List of contributors:

- Luis Martín, UPM
- Alberto Anguita, UPM
- Stefano Chiesa, UPM
- Cristian Cocos, IFOMIS
- Norbert Graf, USAAR
- Gabriele Weiler, FhG
- Haridimos Kondylakis, FORTH
- Andreas Persidis, BIOVISTA

Contents

1. Introduction

One of the main goals of the ACGT project is giving access to heterogeneous, disparate sources of information in a seamless, transparent way. For that purpose, a Data Access Infrastructure is being developed. This infrastructure is comprised by a set of resources of the ACGT Platform, namely the ACGT Master Ontology on Cancer (ACGT-MO), the ACGT Data Access Services (ACGT-DAS), and the ACGT Semantic Mediation Layer (ACGT-SM). Figure 1 shows the architecture of this infrastructure.

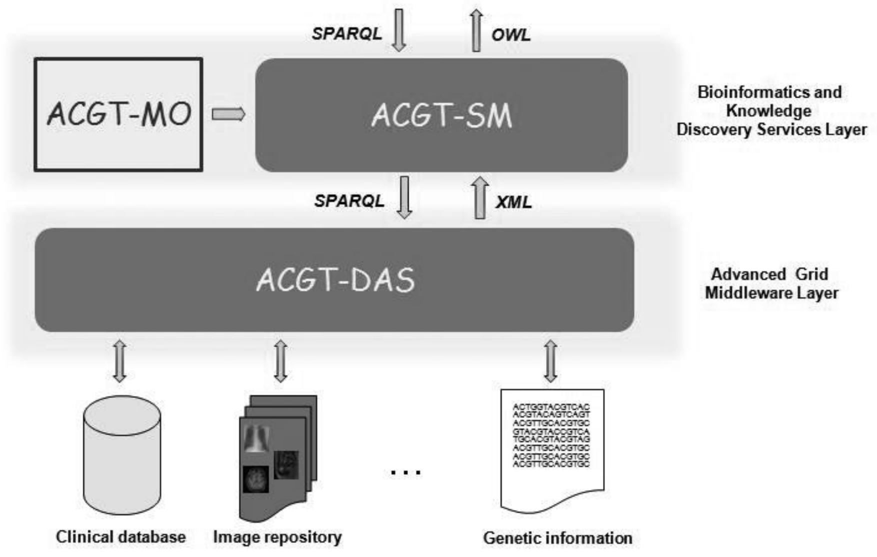


Figure 1: The ACGT Data Access Infrastructure

As can be seen, the ACGT-DAS have the responsibility of accessing the actual data sources, and to overcome syntactic heterogeneities. The client of the ACGT-DAS is the ACGT-SM, which uses the interface provided by the ACGT-DAS—i.e. SPARQL[1] language for querying, SPARQL Result Format [2] for the retrieved results—, and that makes use of the ACGT-MO to overcome semantic issues. This document is focused on the set of tools that comprise the ACGT-SM.

1.1. The ACGT Semantic Mediation Layer

The ACGT-SM is comprised by a set tools of different nature. The main resource is the Semantic Mediator, responsible for solving the semantic heterogeneities and the database integration tasks. This tool makes use of a collection of satellite resources, dealing with different issues related to the main problem. Table 1 lists the different resources that comprise the ACGT-SM, and describes their natures and aims, together with the different technologies used in their development.

Resource	**Aim**	**Interface**	**Technologies used**
Semantic Mediator	Solving semantic heterogeneities and provide database integration	Web Service	Java[3], SPARQL, OWL[4], OGSA-DAI[5], XML
Mapping API	Providing a set of operations to support the mapping process	API	Java, OWL, RDFS[6], XML
Mapping GUI	Guiding a user in the mapping process	GUI	Java, Swing[7]
OWLBasicModel	Providing a model and set of operations to represent OWL ontologies	API	Java, Jena[8], OWL, RDFS
OntoDataClean	Solving instance level heterogeneities in retrieved data	Web Service	Java, OWL
OntoQueryClean	Solving identifier heterogeneities in queries	Web Service	Java, OWL

Table 1: The ACGT Semantic Mediation Layer Tools

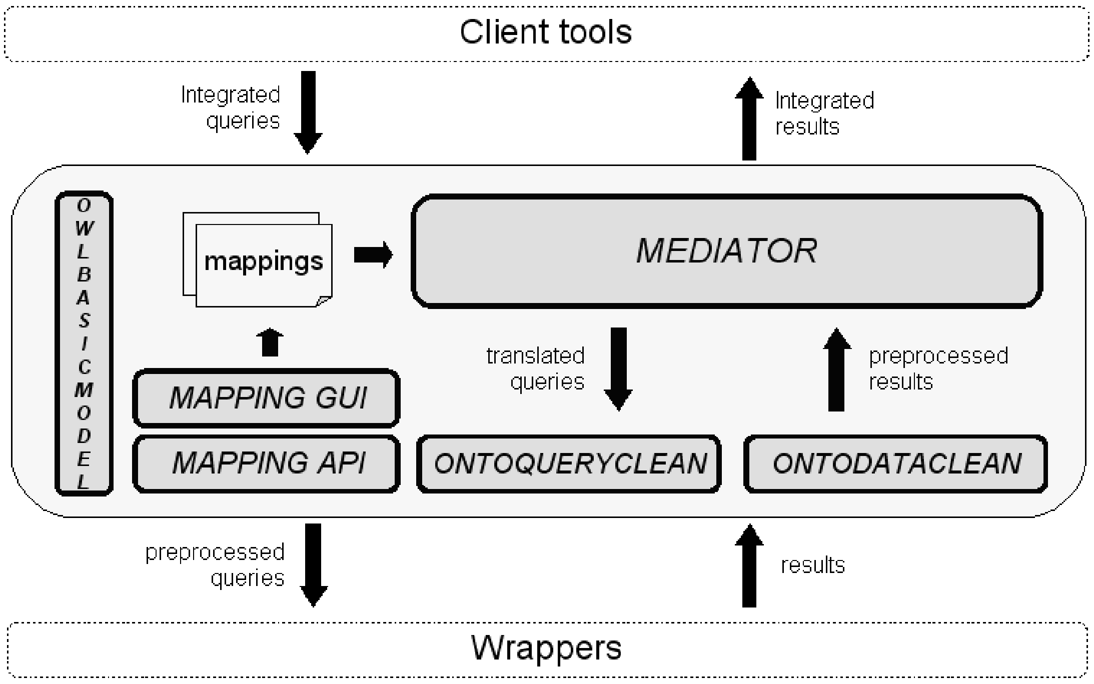


Figure 2: The ACGT Semantic Mediation Layer Architecture

Figure 2 shows the architecture of the Semantic Mediation Layer. The Semantic Mediator acts as the core of the ACGT-SM, coordinating the invocation of services from the others, except in the case of the Mapping GUI and Mapping API, that work independently. The product of these tools (a mapping) is used in the mediation process.

1.2. MO: Short Presentation, Motivation, Challenges and Objectives

The ACGT Consortium chose ontologies as main knowledge representation (KR) tool, in order to represent the relevant parts of medical knowledge gathered along the years by cancer researchers and clinicians involved with the theory and practice of oncology. The advantages of ontologies versus other KR strategies have been extensively covered in the Computer Science literature (see [9] [10] [11] [12]). Medicine is one of the research fields that stand to benefit greatly from the “ontological turn”, as can be seen from the ACGT project.

ACGT aims to collect data coming from various sources; such data, hence, boasts a huge degree of heterogeneity. Clinical trials, in particular, are easy prey to inconsistent gathering procedures and flimsy storage systems. Lack of standardization prevents objective comparison of same types of data acquired as result of such trials, hence renders it quite useless. The consequence is a the lack of interoperability, and the difficulty of carrying out research spanning different trials of the same type (let alone trials of *different* types) [13]. We regard the choice of an ontology-based data management system as a major step towards the advancement of a consistent data collection and processing policy in medical informatics. Furthermore, the taxonomy at the core of each and every ontology can be utilized in building case report forms (CRFs), as a sizeable amount of entries on actual CRFs take the more-or-less direct shape of (a branch of) a taxonomic tree.

Among the challenges of the ACGT MO development, the large scope of the project was certainly the most demanding. Many areas, such as clinical studies, clinical cancer management and care, genomic research etc., had to be reflected; all these could easily make the subject of a plethora of more focused and targeted domain ontologies, wherefrom, ideally, the ACGT MO might be constructed in modular manner. That, unfortunately, could not happen, be it for the simple fact that no such targeted ontologies exist yet, or are not in a consistent shape to meet the quality demands of the ACGT consortium. ACGT partner IFOMIS[1] is active in numerous international efforts aimed at developing cutting-edge ontologies. The Ontology for Biomedical Investigation (OBI) [14], for example, is an ontology that “will support the consistent annotation of biomedical investigations, regardless of the particular field of study”.

In order to provide a consistent and sound representation, the ACGT MO employs the resources of a Top Level Ontology or Upper Level Ontology. We haven chosen the Basic Formal Ontology (BFO)[2] as Top Level for the ACGT MO, since BFO has proven to be highly applicable to the biomedical domain [15]. The ACGT Master Ontology, hence, inherits BFO’s foundational principles: realism (ontologies as representations of reality rather than concrete specifications of conceptual schemes), perspectivalism (many equally valid perspectives on reality), fallibilism (our ontologies are fallible and perpetually evolving), and adequatism (no

[1] http://www.ifomis.uni-saarland.de/

[2] http://www.ifomis.uni-saarland.de/bfo/

emphasis on reducing the various ontological categories to few basic ones). Another design principle implicit in BFO's structure is the conviction that a properly constructed ontology should steer clear of a taxonomical tree that allows multiple parent classes for the same child class [16]. Aside from promoting theoretical accuracy, the directive of multiple inheritance avoidance has, among others, the advantage of yielding a much more intuitive and clutter-free taxonomic tree.

Presented as an .owl file and written in OWL-DL, the ontology was built, and is being maintained/curated, using the Protégé-OWL free open-source ontology editor[3]. In a first instance, the .owl file is intended to be browsed using the same tool, although ACGT member Biovista is currently developing an online browsing tool that will make the ontology available on the WWW (see section 2.5).

1.3. Structure of this document

This document is structured as follows: section 2 gives the detail of the architecture, services and algorithms used for each of the tools comprising the ACGT-SM and the Ontology Viewer. Section 3 regards to the Master Ontology main usages within the project. Section 4 presents the main conclusions from our implementation experience so far and the future lines within WP7.

[3] http://protege.stanford.edu/

2. Semantic Mediation Tools

2.1. The ACGT Semantic Mediator

The ACGT Semantic Mediator is the core software resource among the set of ontology mediation tools developed in WP7. The main aim of this software layer is providing access to integrated sets of data sources, solving the semantic heterogeneities. The Semantic Mediator provides users with two services, namely i) Launch a Query, and ii) Retrieve Schema.

The ACGT Semantic Mediator has two main groups of clients, end users and KDD tools. The profile of the former is a physician involved in a clinical trial. This kind of user normally needs a friendly interface to communicate with the mediator, since it is very difficult for him to formulate a query in SPARQL. This interface is not part of the Mediator itself, and its development will take place during the next phase of ACGT. By contrast, the KDD tools usually communicate with the Mediator via its software interface, which is based in web services technology, using the SPARQL language.

2.1.1. Services and Interface

Table 2 describes the main services offered by the ACGT Semantic Mediator.

Service	**INPUTs**	**OUTPUTs**	**Description**
Launch Query	A SPARQL query	A set of results	Process the query and retrieves the data from the different databases
Retrieve Schema	None	An RDFS representing the schema of the integrated databases.	Retrieves an RDFS that represents the intersection of the available mappings with the global schema. This model represents the universe of possible queries that can be answered by the mediator.

Table 2: Semantic Mediator Services

As can be seen, an RDFS representing the possible queries can be obtained. This schema is a subset of the global schema, and its purpose is twofold: 1) reducing the complexity of the query translation process, and 2) constraining the universe of possible queries that can be formulated, increasing accuracy of the information retrieval.

2.2. Mapping Tool

Establishing mappings between the global and local schema is a key procedure for semantic mediation and database integration. These mappings are used as a means to store the semantic knowledge for the mediation process. They contain the knowledge needed to perform query translation. The ACGT Semantic Mediation Layer follows a Local as View [17] based approach, so local views for the underlying databases need to be created. These local views are contained in the mappings.

2.2.1. Mapping Format

The information contained in a single mapping between the global schema and a schema of a data source can be represented in XML. The complete grammar of the proposed XML format for the mappings can be found as a DTD in Figure 3.

```
<?xml version="1.0" encoding='UTF-8'?>
<!ENTITY % unnamed "">
<!ELEMENT mapping (map)+>
<!ELEMENT map (domain_map,combined_links?,ontoclean?)>
<!ELEMENT ontoclean (conceptualtophysical?,physicaltoconceptual?)>
<!ELEMENT conceptualtophysical (#PCDATA)>
<!ELEMENT physicaltoconceptual (#PCDATA)>
<!ELEMENT domain_map
(src_domain,src_domain_condition?,target_domain,target_domain_condition?)>
<!ELEMENT range_map
(src_range,src_range_condition?,target_range,target_range_condition?)>
<!ELEMENT path_map
(src_path_contition?,src_path,target_path_condition?,target_path)>
<!ELEMENT src_domain (#PCDATA)>
<!ELEMENT src_domain_condition (#PCDATA)>
<!ELEMENT target_domain (#PCDATA)>
<!ELEMENT target_domain_condition (#PCDATA)>
<!ELEMENT src_range (#PCDATA)>
<!ELEMENT src_range_contition (#PCDATA)>
<!ELEMENT target_range (#PCDATA)>
<!ATTLIST target_range id CDATA #IMPLIED>
<!ELEMENT target_range_condition (#PCDATA)>
<!ELEMENT src_path_condition (#PCDATA)>
<!ELEMENT src_path (int_link,int_entity?)+>
<!ELEMENT target_path_condition (#PCDATA)>
<!ELEMENT target_path (int_link,int_entity?)+>
<!ELEMENT int_link (#PCDATA)>
<!ATTLIST int_link value_binding CDATA #IMPLIED>
<!ELEMENT int_entity (#PCDATA)>
<!ATTLIST int_entity id CDATA #IMPLIED>
<!ELEMENT link_map (range_map,path_map)>
<!ELEMENT combined_link (link_map+)>
<!ATTLIST combined_links joined_on CDATA #IMPLIED>
<!ATTLIST combined_links compound_on CDATA #IMPLIED>
```

Figure 3: A DTD representing the XML mapping format

This format can be used to express mappings between paths in an OWL ontology and an RDFS representing the structure of a data source. To support the construction of these mappings, a couple of tools have been created: The Mapping API and the Mapping GUI.

2.2.2. The ACGT Mapping API

2.2.2.1. Interface and Services

The Mapping API is divided in two main functional sets that encapsulate functions for different usages. The first set, called Mapping Editor, provides functionalities for building and modifying mappings. The second set is called Mapping Browser, and allows users to consult the mapping, providing services to find subsumed paths. This set is designed to be used by the mediation software. The two sets of supported operations are detailed in tables 3 and 4.

Operation	**INPUTs**	**OUTPUTs**	**Description**
save	File path	An XML mapping file	Saves the current mapping into an XML file
addEntryToMapping	A pair of paths	An entry	Adds a mapping entry to the mapping
getEntryList		An entry list	Retrieves the current entry list
getEntry	A position in the entry list	An entry	Retrieves the requested entry
removeEntry	A position in the entry list	-	Deletes the entry in the specified position
getClasses	-	A list of classes	Retrieves the list of properties contained in the model
findPaths	A class and a depth	List of paths	Retrieves all the possible paths with a maximum of the given depth with the specified class as domain

Table 3: Mapping Editor Operations

Operation	INPUTs	OUTPUTs	Description
load	File path	-	Loads a mapping stored in an XML file
getCorrespondence	A path	A path	Retrieves the path that is mapped to the given path
getSubsumedPaths	A path	A list of paths	Retrieves all the subsumed paths of a given path

Table 4: Mapping Browser Operations

2.2.3. The ACGT Mapping GUI

The Mapping GUI is a user interface to build mappings for the semantic mediation process. This interface provides simple and intuitive access to the main Mapping API functionalities. The audience of this tools are the engineers and experts involved in the mapping process. This tool has not been designed to be used by clinicians or any other type of end users of the ACGT platform. The next section gives an overview of the interface of this tool. An screenshot of the Mapping GUI is shown in figure 4.

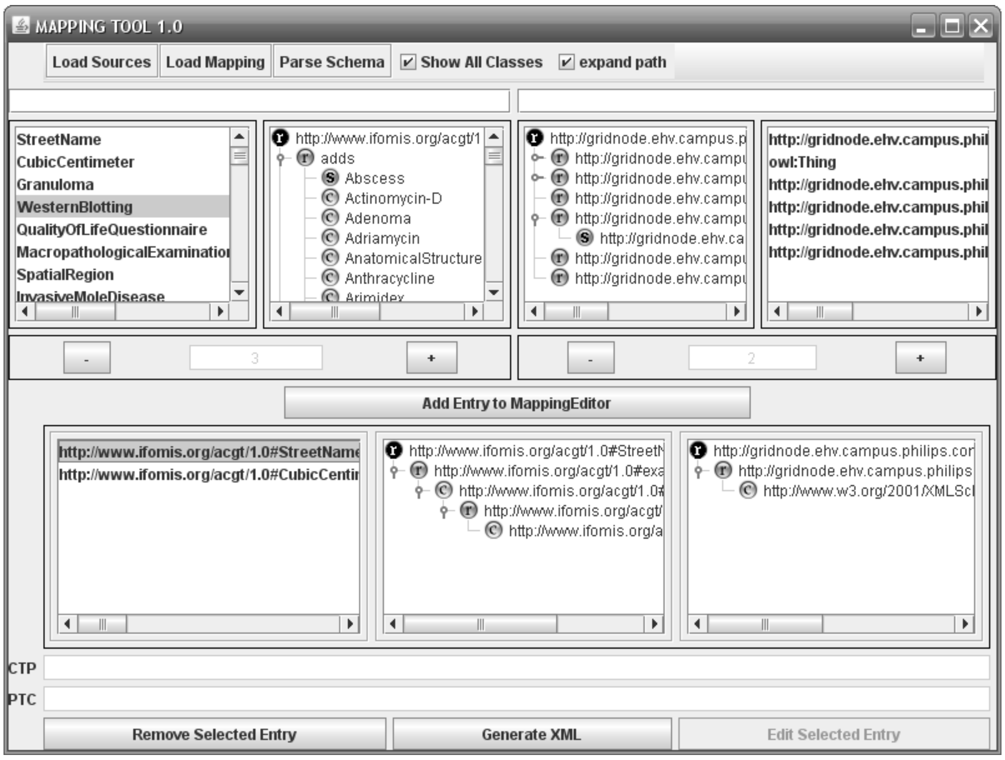

MAPPING TOOL 1.0
Load Sources
Load Mapping
Parse Schema
Show All Classes
expand path
StreetName
CubicCentimeter
Granuloma
WesternBlotting
QualityOfLifeQuestionnaire
MacropathologicalExaminatio
SpatialRegion
InvasiveMoleDisease
http://www.ifomis.org/acgt/1
adds
Abscess
Actinomycin-D
Adenoma
Adriamycin
AnatomicalStructure
Anthracycline
Arimidex
http://gridnode.ehv.campus.p
http://gridnode.ehv.campus.phil
owl:Thing
3
2
Add Entry to MappingEditor
http://www.ifomis.org/acgt/1.0#StreetName
http://www.ifomis.org/acgt/1.0#CubicCentir
http://www.ifomis.org/acgt/1.0#StreetN
http://gridnode.ehv.campus.philips.cor
http://www.w3.org/2001/XMLSc
CTP
PTC
Remove Selected Entry
Generate XML
Edit Selected Entry

Figure 4: The Mapping GUI

As can be seen, the interface is completely symmetric. The source schema is shown on the left side and the target schema on the right. The user selects the paths she wants to be included in the final mapping. The tool allows to export the mapping in XML format.

2.3. Ontology Representation Model: OWLBasicModel

OWLBasicModel is an API designed to offer editing capabilities for ontologies. The purpose is to offer a simple set of features for performing basic editing functions over ontologies, and parsing/writing OWL and RDF files. Many features of the OWL language are ignored in order to offer a very simple model of ontologies.

The API includes functionality for both browsing existing OWL and RDF files and building/editing new or existing files, excluding many of the advanced features of the OWL language. This way, external applications can have a simplified view of the OWL language, performing some basic operations for accessing and modifying OWL files. The selection of OWL features was based on the requirements specification obtained from the Mapping Tool and Semantic Mediation Approach design.

2.3.1. Adopted features from OWL language

Many features of the OWL language are ignored in order to facilitate the editing of OWL files. Basically, OWLBasicModel considers the RDF Schema features plus some OWL restrictions. The next OWL features are ignored by the model:

- No distinction is made between the different types of properties.
- Restriction applied over classes and properties, except for restrictions in the range of a property when applied over a specific class.
- Header information (except for imports)
- Equality statements
- Annotation properties

The architecture of the OWLBasicModel API is shown in figure 5.

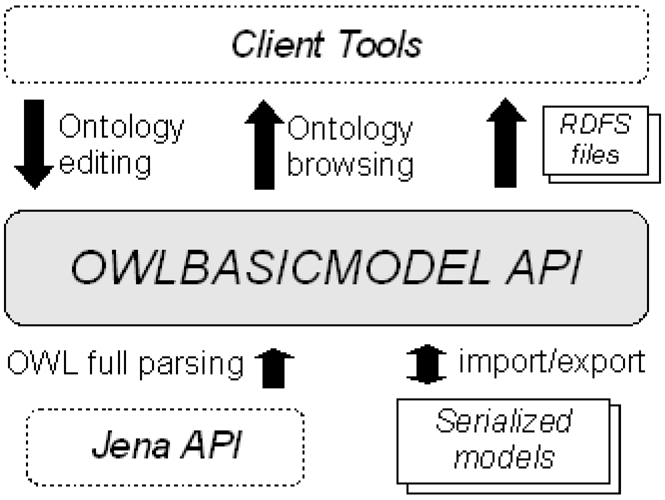


Figure 5: Architecture of the OWLBasicModel API. Arrows indicate data flow among components

The OWLBasicModel API utilizes the Jena API. The latter is an open source semantic Web framework for Java supported by HP. Among other capabilities, it allows parsing ontologies written in OWL language. A complete documentation of the API can be downloaded from the project page, and new versions are released periodically, providing a strong support for users. It is known to be one of the De Facto frameworks for ontology managing, which is why we decided to use it in our API.

The API uses the Jena API to initially parse the OWL files, and then create its own model. A loaded model can be saved into a file using serialization of Java objects. Retrieving the information from these serialized files will result in lower loading times compared to using Jena. One module for producing rdf-schema files from an existing ontology (or created through the API) is included. It allows obtaining the rdf-schema representation of an existing OWL file, or of a newly created ontology.

2.3.2. Services and Interface

Table 5 describes the main services offered by the OWLBasicModel API.

Operation	**INPUTs**	**OUTPUTs**	**Description**
parse	File path	-	Parses an OWL file and creates the corresponding model
load	File path	-	Loads a previously serialized model into memory
save	File path	A serialized model file	Saves a serialized version of the current model into a file

generateRDFSchema	File path	An RDF Schema file	Creates a file containing the RDF Schema representation of the current model
addRDFClass	Class URI, parent class URI	-	Adds a class into the current model
addProperty	Property URI, parent property URI	-	Adds a property into de current model
getRDFClasses	-	List of class URIs	Retrieves the list of classes contained in the model
getProperties	-	List of property URIs	Retrieves the list of properties contained in the model
getPropertyDomainList	Property URI	List of classes URIs	Retrieves the list of classes that form the domain of a given property
getPropertyRangeList	Property URI	List of classes URIs	Retrieves the list of classes that form the range of a given property

Table 5: Description of the main services offered by the OWLBasicModel API

2.4. Instance Level Heterogeneities

2.4.1. OntoDataClean

When performing data integration, there are two kinds of heterogeneities that must be solved in order to allow the correct merging of data coming from different sources. On one side, we have schema level heterogeneities, due to differences in the schemas of the different data sources. This is solved by means of adopting an ontology acting as a common framework. On the other hand, sources may present instance level heterogeneities. These are due to different formats for storing the same kind of data—e.g. scale heterogeneities, utilization of synonyms, etc. OntoDataClean is designed in order to solve the second kind of inconsistencies. It offers a series of transformation methods for preprocessing and homogenizing data. These methods are specifically focused on the biomedical domain, although it can be used on other kinds of data. The tool is offered as a Web Service, so it can be accessed from any machine connected to the Internet. Its interface is designed to accept 2-dimensional data matrices. This format suits the results of querying a generic data repository—one dimension contains the retrieved variables, while the other dimension stores the different results—, thus making it ideal for preprocessing results obtained from any kind of data sources.

In order to increase compatibility and usability by non-technical users, OntoDataClean utilizes an ontology representing the domain of generic transformations that can be applied over a set of data. Instances of that ontology specify how actual data must be preprocessed. Figure 6 shows the basic architecture of the tool.

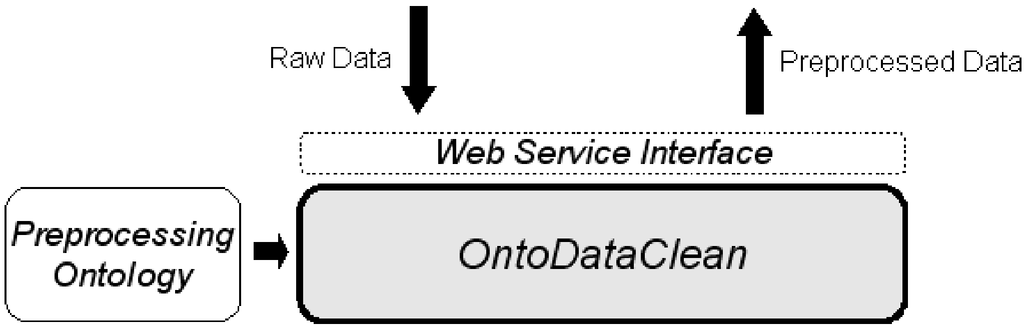


Figure 6: Architecture of the OntoDataClean tool

The input of the tool is the data to be preprocessed, arranged in a 2-dimensional matrix, and the instance of the preprocessing ontology that specifies how that data must be preprocessed. The output will be another 2-dimensional matrix with the resulting preprocessed data. As it was explained before, the preprocessing ontology defines a set of transformations that can be applied over a matrix of data. The ontology is composed by a series of classes, each devoted to a kind of transformation. By instantiating these classes and giving values to their attributes, the user can indicate the tool the exact transformation methods that he wants to use. This preprocessing ontology is shown in figure 7.

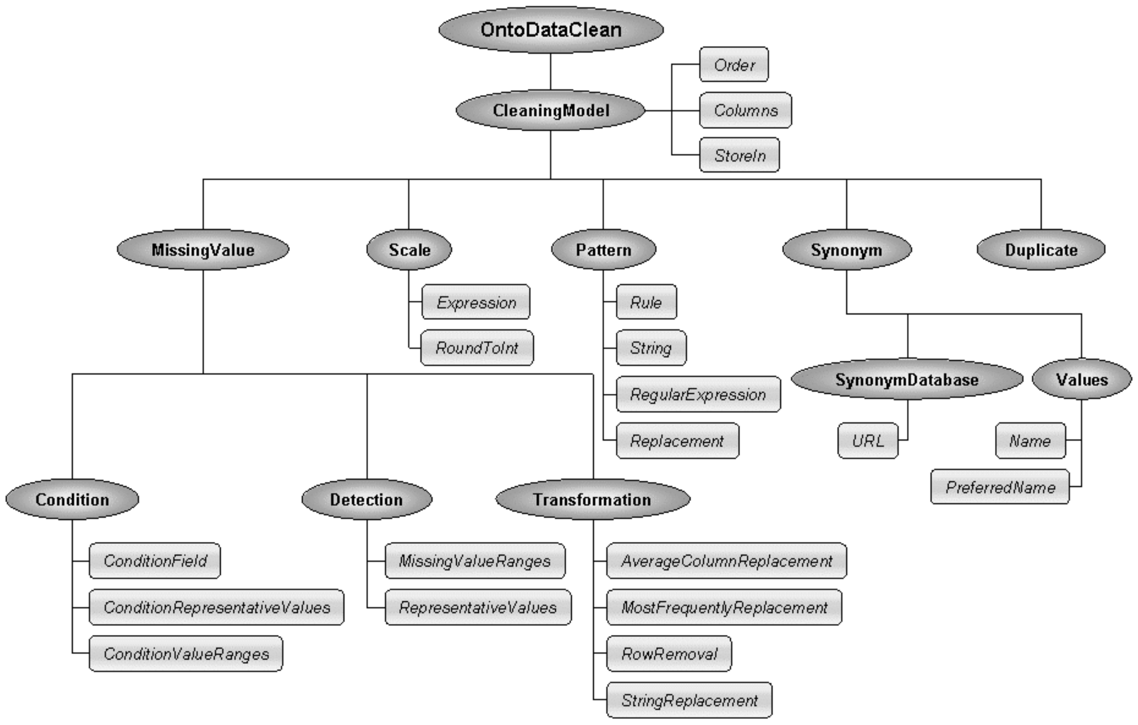


Figure 7: Preprocessing ontology employed by OntoDataClean.

2.4.1.1. Transformation Methods

Table 6 describes the service offered by OntoDataClean.

Service	INPUTs	OUTPUTs	Description
cleanData	2-dimensional data matrix and preprocessing ontology instance	2-dimensional data matrix	Applies the methods specified in the ontology instance to the data contained in the submitted matrix

Table 6: Description of the service offered by OntoDataClean

By means of this service, the user can utilize the five different methods for transforming data in OntoDataClean. The next paragraphs briefly describe these methods, and how to use them.

Duplicate cleaning

This method allows eliminating duplicate values in the data. In order to do this, an instance of the class *Duplicate* must be created. The *Columns* property will be filled in this case with the column names we want to be used in order to evaluate the duplicity of two different rows of data. For each pair of rows in the data matrix, the system will determine its duplicity by comparing all values specified by *Columns*. If every value is equal, then one row is eliminated.

Missing value transformation

The missing value transformation method can perform transformations when specific values (considered by the user as missing values) are encountered in the original data. For this purpose, an instance of the class Detection will be created. Its property *Columns* will indicate the system on which columns of data the method must be applied, as usual. The properties *MissingValueRanges* and *RepresentativeValues* will contain the values subject to be modified by the method—either ranges of values or specific values.

The class that determines how to transform a value subject to be modified is *Transformation*. One instance of this class will be related to the previous instance of *Detection* through the property *DetectionSiblingTransformation*. It is mandatory for each instance of *Detection* to have associated an instance of *Transformation*. Furthermore, one and only one of the four existing properties in *Transformation* must be filled, specifying what kind of transformation to apply. Optionally, one or more instances of the class *Condition* can be attached to the instance of *Detection*. This allows specifying additional conditions in order to determine which values must be transformed.

Scale transformation

This method allows performing algebraic transformations over numeric values, by means of instances of the class *Scale*. The property expression will contain the algebraic expression to apply on the values of the specified columns. Of course, non-numeric values will not be affected by this transformation. The property *roundToInt* allows specifying whether final value must be rounded to the closest integer value, or leave it as a decimal value.

Synonym transformation

The synonym transformation method allows specifying pairs of synonyms which the system will use in order to perform substitutions. The class that must be instantiated is *Synonym*. The property name must be filled with a string value that will be recognized as the value to substitute. The property *preferredName* must contain the value that will substitute the previous value. Optionally, a database can be referenced in order to specify the pairs of values to use in the substitutions.

Pattern transformation

Pattern transformations allow performing complex modifications over string values by specifying a series of properties. It can be used to recognize values that match a given regular expression (using the property *RegularExpression*) in order to substitute these values with another value. The other option is to specify a rule string (using the property *Rule*), which allows reorganizing the original value so it adjusts to a different format.

2.4.2. OntoQueryClean

Instance level inconsistencies must be solved when integrating results from heterogeneous data sources, but also when generating subqueries in a federated environment such as the one adopted by the ACGT Semantic Mediator. When an integrated query is launched against the mediator, it may include literals that restrict the values that are queried. For example, a user might want to restrict the age of patients suffering some kind of illness by specifying that that value must be over 30. The value that he will include in the restriction will depend on the data format offered by the mediator. However, this format may not resemble the one used by the different data sources that the mediator has to access, therefore this restriction must be preprocessed in order to generate the correct subqueries. OntoQueryClean aims to solve this issue by offering a series of methods for restriction preprocessing. It is based on the previously described tool OntoDataClean. Again, it makes use of an ontology defining the domain of transformations for restriction translation. This is an adapted version of the one employed by OntoDataClean. Only three transformation methods are kept—synonym, scale and pattern—and one more for operator preprocessing is included. Figure 8 depicts this ontology.

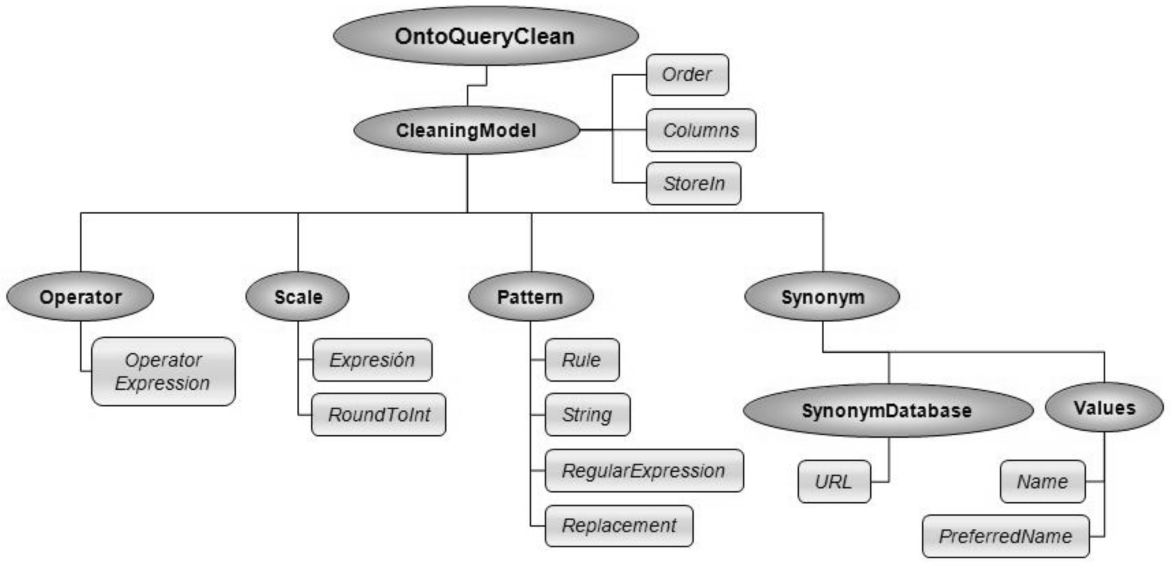


Figure 8: Preprocessing ontology employed by OntoQueryClean

OntoQueryClean makes use of OntoDataClean in order to perform the scale, pattern and synonym transformations. It collects the required information from its own preprocessing ontology and its data matrix, and invokes OntoDataClean. Afterwards, it performs the necessary transformations over the existing operators. Figure 9 represents the architecture of this tool.

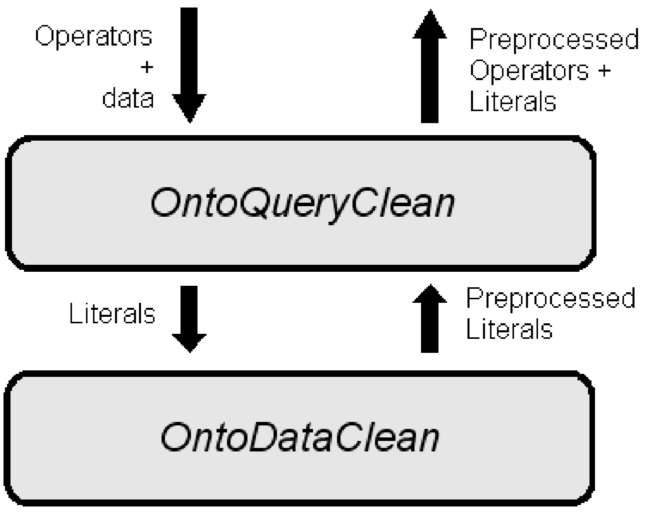


Figure 9: Architecture of OntoQueryClean

The input of the tool is a preprocessing ontology instance and a 2-dimensional data matrix (containing both operators and data). The result will be again a 2-dimensional matrix containing the preprocessed operators and data.

When receiving a data matrix composed of operators together with literals, OntoQueryClean extracts the literal information and submits it to OntoDataClean so it is properly preprocessed—the information from the preprocessing ontology regarding literal transformation is extracted as well and submitted. After receiving the results, the tool performs the necessary transformations over the operators, and adds the results to the preprocessed literals.

2.4.2.1. Transformation Methods

Table 7 describes the service offered by OntoQueryClean.

Service	**INPUTs**	**OUTPUTs**	**Description**
cleanData	2-dimensional data and operator matrix and preprocessing ontology instance	Preprocessed 2-dimensional data and operator matrix	Applies the methods specified in the ontology instance to the data and operators contained in the submitted matrix

Table 7: Description of the service offered by OntoQueryClean

This service allows using the methods offered by OntoQueryClean. Scale, synonym and pattern transformations are used exactly the same as in OntoDataClean. The operator transformation method allows entering expressions for evaluating result operators given other operators. This is done using the class *Operator*, and specifying this expression in the *OperatorExpression* property. This property can hold constant operators—namely "=", "!=", "<", ">", "<=", ">="—, operators contained in a specific column—given by the name of the column enclosed between '{' and '}' characters, and functions of operators which result in another operator. The supported functions are: i) Opposite —"opp(op)"— computes the opposite of the argument, ii) Complementary —"comp(op)"— calculates the complementary of an operator, iii) Union —"union(op1, op2)"— computes the union of two operators, and iv) Intersection —"int(op1, op2)"— calculates the intersection of two operators. For example, the operator expression "union(<, <=)" would always return the operator "<=".

2.5. The Master Ontology Viewer

The ACGT project has at its core the ACGT Master Ontology (MO) which is the basis for a number of current and planned ACGT services. Since the ontology is a work in progress the ability to explore and visualize it is required for substantial and in depth reviews from the appropriate experts in the group. Since the ACGT MO is already quite large, the ability to search has grown to a necessity, a feature that is not easily found on all ontology viewers.

A lot of current ontology viewers are overloaded with functionality that makes the learning curve of a new tool quite steep. For that reason, and for ubiquitous access, the Biovista Ontology Viewer has been built from the ground up to be a web based viewer, practically usable by anyone who has internet access. This way it is much easier to test, by a wide range of interested partners during the development period, and to later integrate it in the ACGT web site.

Given that most fully featured viewers are proprietary and that both the ACGT consortium and the EU require source availability it was decided to develop the ACGT ontology viewer from scratch thus being able to develop custom features that closely meet the current and future requirements of the project.

In preparation for the development of the Biovista ACGT Ontology Viewer a lengthy review on available OWL ontology viewers was performed. The objective of the review was to find a suitable code base in order to jump start the development.

Some observations from this work are the following:

- All of the reviewed tools are Java based
- Some of the most interesting viewers are also editors which increases by far the complexity of the program
- Some of the most interesting viewers have a non open source, proprietary license
- A lot of open source projects are currently abandoned or they were too simplistic to start with
- The two main backend libraries are the Jena library (HP sponsored) and the OWL API and all projects seem to utilize one of them

Table 8 summarizes our findings on these viewers (Biovista's ontology Viewer is also included for comparison):

Viewer	Web based	Easy to Use	Search	API	Open Source
Protege	no, java based	No, full editor	yes		yes
SVG-OWL (mindswap)	No, java based	Crashes with ACGT ontology			unsure
DUMPONT	Yes but simplistic	yes	no	Jena	yes, dead
SWOOP	No, java based	No, full editor	yes	OWL	Yes
ObjectViewer	Yes but simplistic	yes	no	Jena	yes, dead
Ocelot	Yes but simplistic	yes	no	Jena	Yes, dead
Growl	Yes, applet based		no	OWL	no
Owlsight	yes	yes	no	Jena	no
Biovista OV	Yes	yes	yes	Jena	Yes

Table 8: List of existing ontology viewers

The Ontology Viewer presents a tree view of any given ontology. It introduces the concept of Ontology View's whereby the user is free to design more targeted and specific ontologies starting from the master ontology. Following is a list of the prominent features and capabilities built into the Ontology Viewer:

- The Ontology Viewer UI layout is split into three panels:

  1. The left panel gives a user a listing of the various ontology view's that he may choose from. All user actions related with view management such as creation, updating and deletion are handled from UI elements of this panel.

  2. The center panel provides a tree view of the ontology currently being viewed by the user. The tree constructed from the ontology data is a standard component of the GWT library, and is highly customizable.

  3. The right panel in the layout shows the details of the node that has been selected from the tree view. All actions relating to the node selected are performed through menus available at the bottom of the right panel.

- The master ontology used by the Ontology Viewer is not maintained as a separate local copy instead it is loaded directly from the ACGT SVN every time a user reloads the application. This allows the user to always obtain the most recent view of the master ontology. The URL used for downloading the master ontology is:

  http://acgt.googlecode.com/svn/trunk/document.owl

- The Ontology Viewer provides a quick search feature through which a user may find a node of interest in the current ontology by giving a search string. The search is fast and for cases where multiple matches are found will pop up a dialog box wherein the user can select the node of interest to him.
- The Ontology Viewer also provides a search feature which can be used to find all views that may contain a search string.
- The Ontology viewer has the capacity to communicate with web services. One such service already being used is the Biovista bibliography web service.

3. The ACGT Master Ontology on Cancer

The ACGT Master Ontology has been designed from the outset in accordance with modular principles. A sizeable part of the work undertaken by IFOMIS consisted in parsing actual clinical trial report forms (CRFs) and attempting to capture in electronic format the knowledge contained within. A further step is to develop and enrich the ontology obtained so far with constraints and axioms that govern the behavior of the classes and relations in the ontology: this represents the shifting of focus from the simple quantitative activity of adding nodes in the ontology's subsumption hierarchy, to an essentially qualitative stage in the development of the MO. This step is of particular importance, as the MO-based CRF builder (among others) requires a rich network of relations in order to capture the complexity of clinical trial information, and output CRFs boasting a reasonable—certainly perfectible, if not perfect—degree of exhaustiveness. Adding constraints, however, calls for further enhancing the ontology with information about human anatomy—hence further extending the taxonomic tree.

The initial plan was to proceed, thanks to the modular design of the MO, at importing the Foundational Model of Anatomy (FMA™[4]), which is the topmost, state of the art, electronic representation of human anatomy available nowadays. The FMA is, however, available as a frame-based system, and the challenges of rendering it as an .owl file have been extensively documented. We are aware of two major groups of attempts at obtaining an OWL equivalent of the FMA, one by C. Golbreich [18], and the other by the FMA curators themselves [19]. The latter provides two versions of such a translation: an OWL DL component that contains the FMA constructs that are compatible with OWL DL, and an OWL Full component that imports the OWL DL component and adds the FMA constructs that OWL DL does not allow (e.g., metaclasses and classes as property values). Both versions have major shortcomings, *besides* the fact that the degree of conversion accuracy leaves a lot to desire: the corresponding .owl files are very big, while the latter, though smaller, is not even usable for reasoning and inference purposes, as it is obviously not DL compatible. The sheer size of the OWL files, however, makes both quite challenging from the perspective of effective computability.

The Golbereich translation efforts do, *prima facie*, offer reasons for a more optimistic outlook, as they are more accurate than the rough conversion undertaken by the FMA team—though, nevertheless, it has still been deemed as not being accurate enough. Besides questionable factual accuracy, it still suffers from the same ailments most FMA translations do, namely raising inference problems hard to solve in terms of time and memory, due to the complexity of the information contained.

Confronted with such issues, we have decided on an alternate course of action, targeting the NCI Thesaurus (NCIT). While an OWL format of the thesaurus exists, the NCIT contains much more information, well beyond our needs in terms of strict anatomy. Thankfully, we have managed to locate an effort currently unfolding at Manchester University, under Prof. Alan Rector's tutelage, whose main object is to mine the NCIT for anatomical information. The ontology has been enriched with some parity and partonomy knowledge. It has around 8000 anatomical concepts extracted from the NCI Thesaurus, and annotations and mappings provided by the NCI. We, hence, regard importing this module as

[4] http://sig.biostr.washington.edu/projects/fm/index.html

an answer to our quest for an anatomy section, and the best such conduct currently available in terms of developing the ontology at this stage.

Finally, IFOMIS continues its efforts at (a) providing constraints for the ontology classes (especially in collaboration with ACGT medical partners), and (b) assisting ACGT associates in the mapping process. More concretely, the latter has benefited from the interaction between ACGT partners FORTH and IFOMIS, in that several CIDOC-CRM[5] conceptual schemes and diagrams issued by FORTH have been translated in ACGT MO lingo. It is our hope that our contribution in the mediation process will assist the other partners in developing a comprehensive and robust management system, capable to fully exploit the master ontology, and provide means for ontology enrichment, adjustment and, eventually, improvement.

3.1. The Mapping Process

In order to specify the complexity and the expressiveness of the required mappings we have to carefully examine the source and the target schemata that are going to be integrated. We have to note that in a LAV data integration system we need a richer target schema with equal or higher expressive power to fully capture the semantics of the source schemata

An ontology or a data model can be used as a target schema. That schema uses nodes, links, properties, multiple is-a relations (ISA), and multiple instantiations, in order to describe a domain of interest. An ontology is an explicit declaration of a conceptualization [20] which includes a set of concepts, their definition and their inter-relationships. Moreover, the language used to describe the target schema should permit ISA relationships between both classes and properties. We chose to adopt RDF/S [21] as the target schema language, despite its problems and the lack of expressiveness. This is because it is already a standard proposed by W3C, its industrial usage is starting to emerge, and it is the most practical approach for the time being. Furthermore, it is richer than most source models and adds “no constraints” for integration. A suitable part of an ontology in a suitable encoding can be used or interpreted as target schema. Considering source schemata, ACGT tries to integrate only Relational and XML sources and not object-oriented ones, since the objective is to be practical and effective. The most successful database integration scenarios are those that try to integrate traditional databases.

In this document we use as example target schema the CIDOC Conceptual Reference Model, a part of which is shown in Figure 10. Whereas the CIDOC CRM was created for information from cultural heritage, it is adequate to model other domains as well since it provides a core ontology for information integration. We regard a core ontology as the upper level of an Enterprise or Target model. A well formulated core ontology should be complete expressing the basic concepts that are common across a variety of domains and providing the basis for specialization into the nuanced concepts and vocabularies of individual domains. The primary role of the CRM is to serve as the “semantic glue” needed to transform disparate, localized information sources into a coherent and valuable global resource. Essential to the CRM are subsumption hierarchies of both classes and properties, and the concept of multiple inheritance. It uses properties on properties in order to denote a dynamic sub-typing of roles and the scope notes of some properties declare that these

[5] CIDOC CRM is FORTH’s proprietary ontological framework, destined initially to serve archaeological data environments.

properties can be deduced from other data paths in the model. Furthermore CRM does not enforce cardinality constraints into the target schema as they are considered to be implementation details with only minimal explanatory value.

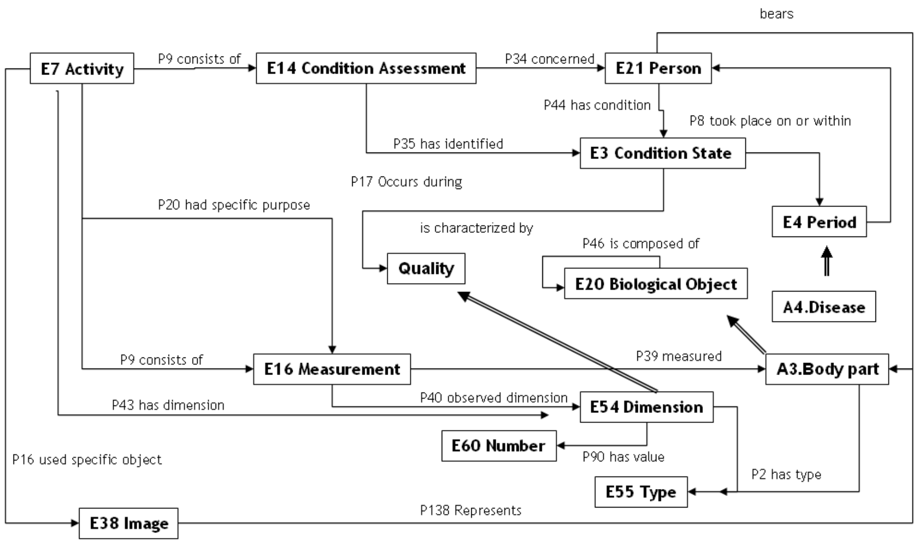


Figure 10: A part of the CIDOC CRM ontology. (The strong lines imply ISA relationships)

Considering source schemata, ACGT tries to integrate only Relational and XML sources and not object-oriented ones, since the objective is to be practical and effective. The most successful database integration scenarios are those that try to integrate traditional databases.

3.1.1. Mapping schema entities

We consider the mapping of two schemata as a sufficient specification for the transformation of each instance of schema 1 into an instance of schema 2 with the same meaning as shown in Figure 12. We refer to the same meaning as it is understood by experts. The definition should be independent of particular instances. The mapping should make it possible to implement an automatic transformation algorithm for all instances of schema 1 into instances of schema 2, only following the specification of the transformation, i.e it should not depend on any further information.

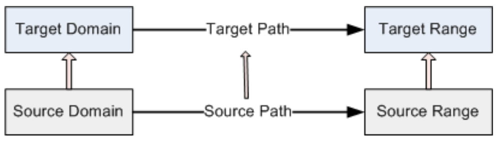


Figure 12: The basic mapping schema

In order to have an efficient mapping, independently of the data model used, we need to define:

- The mapping between the Source Domain classes and the Target Domain classes.
- The mapping between the Source Range classes and the Target Range classes.
- The proper Source Path.
- The proper Target Path.

- The mapping between Source Path and Target Path

In some cases, we may need to combine paths sharing the same instances. Each path f the source schema is mapped individually to the target schema. Each class-role-class can be seen as self-explanatory, context independent proposition. The mapping allows creating sets of propositions equivalent to the meaning of each source document, but in terms of the target schema. Such a mapping is shown in figure 13.

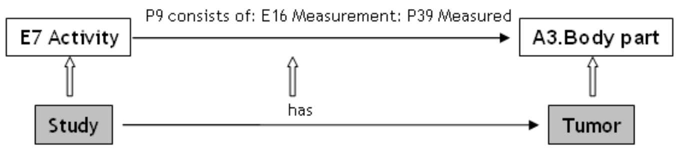


Figure 13: An example mapping

Databases can be seen as collections of statements or propositions. There is no proposition without a relationship. Even though it is easy to make a schema with only one class, it is impossible to make a schema without relationships. The mapping allows creating sets of propositions equivalent to the meaning of each source proposition, but in terms of the target schema. As the propositions are self-explanatory, they can be merged into huge knowledge pools, ignoring the boundaries of the source documents they were derived from.

The inputs of this sub-process are the target and the source schemata (their corresponding propositions) and the output is a file with the mappings. The domain expert has to specify the mappings of the target and the source schemata possibly assisted by an IT expert in the beginning. An appropriate graphical tool should also be used. That tool should be available to make the whole process more intuitive, to ensure consistency between target and source schemas and to help the experts with the appropriate visualizations for the result of those mappings. The “ACGT mapping tool” fulfills that role. Moreover, in a LAV system in order to be able to fully capture the semantics of the source schema the mapping should be complete for each source. We define the mapping of a source to be complete if for every class and role defined in the source schema (or the part of interest) there is a mapping to the target schema.

3.1.2. Identifier construction

This sub-process has to do with the definition of identifier creation and transformation conventions and algorithms for the particular instances in the sources. This is to ensure that local identifiers and keys in sources can be turned into URIs and vice versa. The art is to ensure that (a) no two identifiers denote the same object, and that (b) most probably two independent systems will come up with the same identifier for the same real world item. Requirement (b) is weaker than (a). The identity problems arise in several different contexts and provoke intense philosophical debate. However, it has not been addressed in a systematic and efficient way. Moreover, the interpretation of instances cannot be automated. Consider for example two patient instances with the name “Martin”. There is no method to determine whether the instances refer to the same person using only the schema knowledge. However, instances level heterogeneities solving tools—i.e. Ontodataclean—can automate the process of unique identifier generation.

3.1.3. Terminology matching

The final step is to match terminology that appears typically as data in the sources. There are only three relationships of matching terms: Equivalence, overlap, subsumption. Identifiers for terminology appearing as data must be transformed into URIs and any other data instances. For instance, “Hospital: person_type 12” can be turned into “HospitalPersonType:12”.

3.2. Master Ontology and Global Schema: Query Formulation

One of the main roles of the Master Ontology within ACGT is acting as Global Schema, representing the integration to the universe of integrated databases. The MO is used in this context to build the necessary queries that will begin the process of information retrieval. Although the Semantic Mediator is able to process these queries, translating them into proper ones that are sent to the Data Access Services, this software cannot run without the schema.

There exists two ways to use an ontology in query translation. The first one is using the ontology as domain model, but not as global schema. In this case, the knowledge needed is extracted from the ontology and expressed in such a way that the underlying databases can be integrated. This approach is called Global as View. By contrast, the ontology can be used as global schema in the second way. Here an isolated view for each one of the underlying databases is created using knowledge extracted from the ontology, but there is no need of creating an ad-hoc schema, as the ontology itself can be used for this purpose. This solution is called Local as View. In the ACGT Semantic Mediation Layer a Local as View based approach has been selected. This means that the Master Ontology will act as the representation of all the possible queries for the different integrated repositories.

The main reason for the selection of this approach is the nature of the domain of post-genomic clinical trials. This is a constantly changing domain, since clinical trials are a research discipline. Data in these trials can change throughout the time, so there is a need of a flexible database integration approach. Another reason for the selection of Local as View is the Master Ontology characteristics. The ontology has been created to be friendly for end users. This means that, for a clinician, it is easier to deal with the Master Ontology than with a database schema like model. This proximity will enhance the construction of friendly ontology interface tools for different purposes—e.g. query construction tool.

The following sections explain how the ontology is used as global schema. Section 3.2.1 explains the structure of the ontology and where the schema is contained exactly. Section XX2 is devoted to query formulation using paths from the ontology, and their relation with natural language. Section 3.2.2 explains the main features of the query language used by the mediator, and its relation with the global schema.

3.2.1. Master Ontology structure, RDFS as query Schema

The MO consists of a hierarchy of classes, each of them described by properties and connected to other classes by relations. These classes and relations can be restricted through a variety of assertions. Such restrictions allow external applications to perform inference over the ontology content. For example, two classes may be stated as being disjoint from each other, meaning that no individual can be an instance of both classes simultaneously. OWL-DL [4], the chosen language for representing the MO—specifies the types of relations and restrictions that the MO can include. This language is an extension of RDFS, used for representing schemas of RDF databases.

While proper inference may be very useful in specific areas, it is of no use for the SM. The MO is utilized by the SM as a global schema for representing a virtual RDF repository. For that reason, only the RDF Schema content of the MO is useful when acting as global schema. OWL-DL specific information is ignored for this task—all properties defined in OWL-DL language such as ObjectProperty or FunctionalProperty are considered as plain RDF properties. There is one exception: range restrictions given by the tag owl:allValuesFrom are still considered. This decision was taken based on the following criteria: “Information contained in the MO regarding valid paths should always be considered when using it as a global schema”. An allValuesFrom restriction can turn valid paths into incorrect ones, impacting the range of queries that the global schema should offer. Figure 14 depicts to what extent OWL-DL characteristics of the MO are considered when using it as a global schema by the SM.

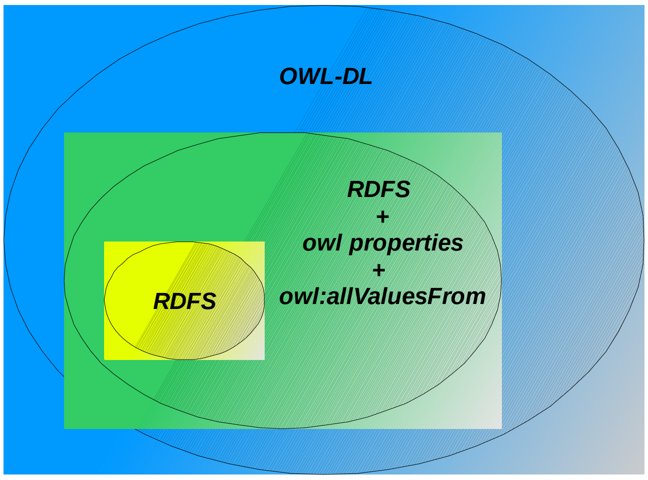


Figure 14: The green area represents the characteristics of the MO considered by the SM when using it as global schema of the virtual RDF repository

Future work may include the use of more features of the MO in the SM. Inference on the ontology content may be applied by an advanced query interface in order to assist users in the construction of proper queries.

3.2.2. Paths vs. Natural Language

Paths in the MO are used to express queries against the virtual RDF repository. A path is comprised by an ordered list of alternating classes—representing entities in the real world—and relations linking those classes—representing how those entities relate to each other. The results of a given query are made up of instantiations of the nodes contained in the path of the query. This query paradigm is somewhat closer to natural language than a standard relational query paradigm. In a relational query language, relations are specified by means of comparing different key fields, which may not have a direct meaning in reality. On the other hand, a path itself specifies which entities are to be queried and how they must relate to each other. Such relations are given by the RDF relations present in the path. These relations do express a real world relation of entities, and are therefore much closer to human understanding of reality. For example, in order to extract which patients are treated in a specific hospital named Hospital_A as part of a clinical trial, a corresponding SPARQL query for the SM would be the one shown in Figure 15.

```
PREFIX h: <http://www.ifomis.org/acgt/1.0/>
SELECT ?patient
WHERE
    ?patient  h:undergoes       ?clTrial
    ?clTrial  h:isFacilitatedBy ?hospital
    ?hospital h:hasIdentifier   ?hID
    ?hID      h:hasStringValue  "Hospital_A"
```

Figure 15: SPARQL query for extracting the patients being treated in a specific hospital

Following the WHERE clause, the path that this query expresses can be easily visualized: Patient → undergoes → ClinicalTrial → isFacilitatedBy → Hospital → hasIdentifier → HospitalIdentifier → hasStringValue → “Hospital_A”. This path can be expressed in natural language by the following sentence: “Patients that undergo clinical trials facilitated by hospitals that have identifiers whose string values match Hospital_A”. Although long, the sentence can be easily understood by a non-technical user.

An equivalent relational query would involve relating many indexes from different tables. These indexes would not necessarily correspond to an entity in reality, thus such query would not be so close to natural language.

Future work might involve developing an advanced GUI that provides, to some extent, a natural query language interface. A dedicated module would be in charge of producing equivalent SPARQL queries that can be sent to the SM.

3.2.3. Query Language

As has been mentioned before, the SM provides access to a virtual RDF repository which represents the integration of all underlying RDF databases. In order to allow querying RDF data, it is necessary to support an RDF query language. The chosen language is SPARQL [fixme]. This language is a W3C recommendation since January 15th, 2008.

Creating a SPARQL query involves specifying one or more paths in an RDF schema (in this case the MO), and relating or restricting some of its elements. A path is composed by a set of triples. A triple consists in two classes (or instances of classes) and a relation between them.

In order to avoid several complexity-related problems of the SM, some restrictions have been imposed over the query language. These do not constrain the types of queries that the user can formulate, but force her/him to follow a style of query formulation that significantly ease the design of the SM. These restrictions are described below:

- Reduction of space of queries: this is not exactly a way to reduce the design complexity of the SM, but rather to increase its efficiency by decreasing the computational complexity of the semantic translation. The idea is to omit elements of the MO which would not produce any results if queried. Following this approach, only mapped classes of the MO and their superclasses are presented in the global schema. In addition to this restriction, it is recommended to avoid as much as possible the use of highly generic classes in queries—the ones in the top of the hierarchy of the MO. Including these in a query may involve elevated response times due to very large searches in the existing mappings.

- Language restrictions: several restrictions on SPARQL have been imposed in order to reduce the complexity of the SM.
    - Variable constraints: The main restriction is about where the constraints must be placed in a query. In order to avoid ambiguities, all constraints must be included in the FILTER section of the query. This is only a syntax restriction does not lead to any loss of expressivity in the query language.
    - Path linking: linking different paths in a query is restricted to the WHERE section. This is due to the same reason as the variable constraint restriction.

These restrictions/recommendations will be formally specified in the document describing the complete approach (D7.4 – Consolidated approach for semantic mediation and integration of heterogeneous data sources for clinical trials).

3.3. Master Ontology in the Clinical Trial Building Procedure

The ultimate goal of ontology-based information management in eHealth is the direct integration of data created in different environments (e.g. clinical data, DICOM data, research data, laboratory data, etc.). The collection and storage of such data is most important for clinical trials. Even today in many multicentre trials paper-based CRFs are still used. From the participating hospitals CRFs are sent to a central data facility, where the data is entered into a trial database. This is very time consuming and error-prone.

Preferable systems used today are Web-based remote data entry systems where the data are captured at the participating site and transferred electronically to the trial central data facility. Most of these management systems allow design of the trial and especially creation of eCRFs (electronic CRFs) by trial leaders without any informatics skills. The user is free in defining the items for the CRFs and the names for the database tables and attributes that are set up automatically from these definitions. The biggest disadvantage of these management systems is the lack of comprehensive and standardized metadata. This limits interoperability which means that similar data in different clinical trials cannot be compared. A sensible analysis of the collected data for extended research across different trials will only be possible if an Ontology is integrated in such tools. It is of utmost importance that ACGT will do so. To explore this approach, an Ontology-based Trial Management System for ACGT (ObTiMA, see D2.2) is under development that enables chairmen of clinical trials to set up a patient data management system with comprehensive metadata in terms of the ACGT-Master Ontology (MO). This will allow the seamless integration of data collected into the ACGT mediator architecture.

Clinicians and other end-user are not used to handling ontologies and they should not be aware of the fact that an Ontology is used in ACGT. They cannot be bothered with theoretical aspects and design principles of databases or ontological metadata. Therefore, in ObTiMA the trial chairman will define both by creating the CRFs for his trials. With the help of the Trial Builder, the trial chairman can define the questions on the CRFs, the order in which the questions will appear and constraints on the possible answers. In order to create a question on the CRF he has to describe it semantically by choosing a description from the ACGT MO. By using the Trial builder all items that are needed by the end-user will be automatically connected to the corresponding item in the Ontology if available. If not, a new item in the ontology will be created. This will maintain the Ontology and will only add such items to the Ontology that are really relevant for the trial. Such a process is described and specified in more detail as a submission system for the maintenance of the MO.

For example, the clinician may want to collect information on the patient's gender. He can see that there is relation between the classes 'Patient' and 'Gender'. To create the referring question the clinician chooses the class 'Gender'. The needed attributes to create the possible answers on the CRF can be determined automatically. On the CRF the label of the question appears as the name of the class 'Gender'. The allowed values are set automatically to "male," "female," and "ambiguous gender" since the class 'Gender' is defined as an enumeration in the ontology containing these values. Thus, a multiple choice question is automatically created on the CRF. This procedure automatically implements the semantics of the ontology in the CRFs. We expect that this description is a path from the ontology starting at the class 'Patient' since this class normally is the focal point of CRFs. Starting from the class 'Patient', for example, the clinician sees the possible classes and relations from the ACGT-MO that are sensible to connect with 'Patient', e.g. that the patient can have diseases, a blood pressure, a birth date, etc. Now, he has the opportunity either to select one of the classes to create a question on the CRF or to explore them further and show their subclasses or relations to assemble more complex questions. This is called the clinical view of the ACGT-MO (as shown in figure 16).

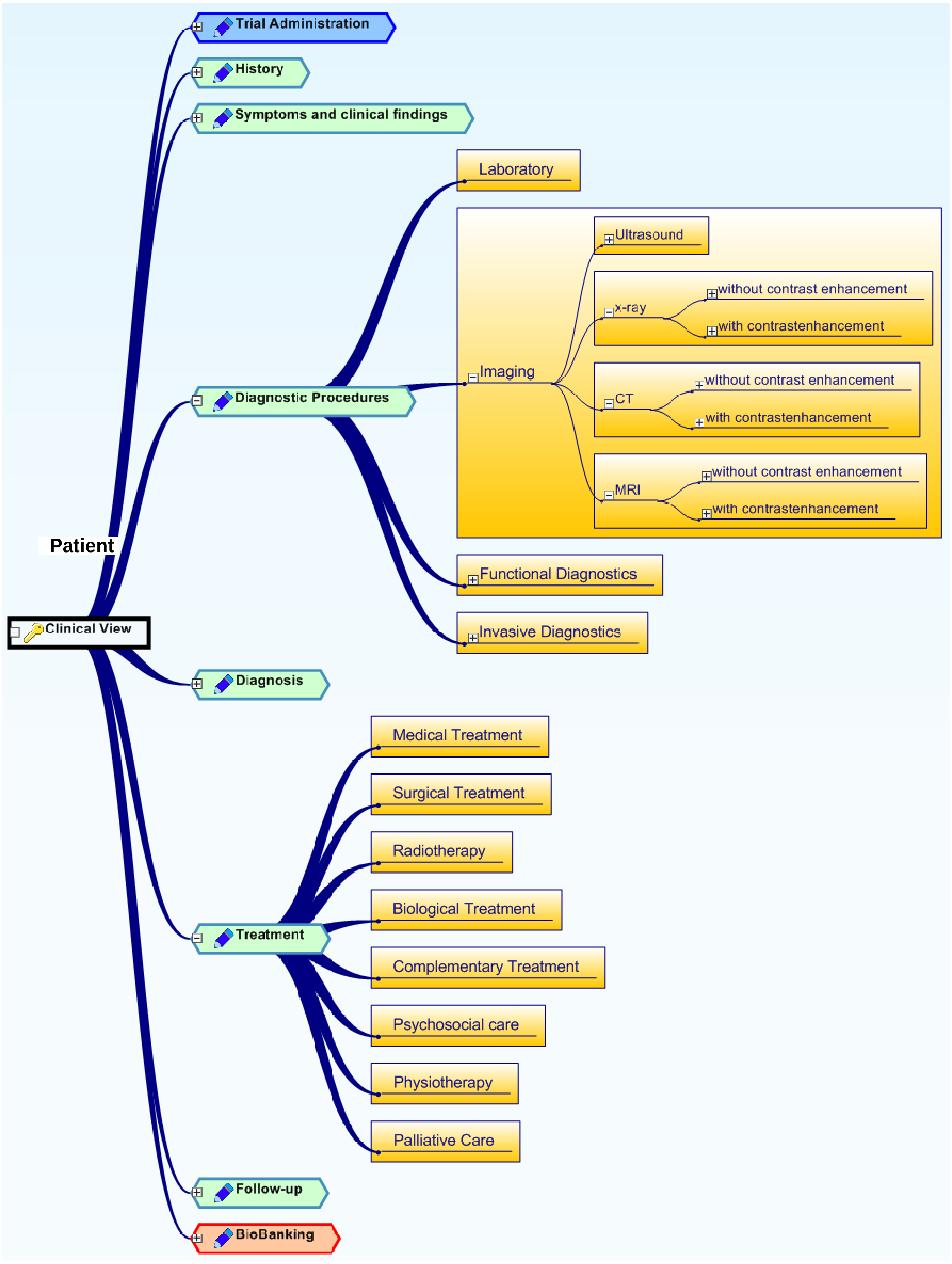


Figure 16: Clinical view of the ACGT-MO.

The clinician will also have the ability to enter every question/item on a CRF via an entry field. This entry field is directly connected to the MO and is in fact a search field of the MO. The MO will be searched by this item and the corresponding Thesaurus. Such a Thesaurus is needed and implemented in the system. Controlled vocabularies are available from the Enterprise Vocabulary Services (EVS). Having found the item in the MO a clinical view of the MO is presented to the clinician depicting the item and the dependencies in a tree diagram. This allows the clinician to validate the correctness of the item and to copy not only the searched item to the CRF but a whole branch or a part of it. In both cases (using the clinical view or using the entry field for search) the clinician is not aware that he is actually dealing with the MO. Two possibilities may occur in case a clinician wants to add a new item to a CRF in the above described way.

1. The item (or a synonym of the item (found in an implemented Thesaurus)) is found in the MO:

   No further steps are necessary. By copying the item (or even a whole branch) to the CRF the link to the MO is automatically created. The end user is only confronted with the clinical view of the MO.

2. The item or a synonym of the item is not found in the MO:

   In this case the clinician is able to add this item to a selected branch of the MO. Entering new classes to the MO is part of the submission system of the ACGT-MO. A new window will open and the user will get the clinical view of the Ontology. He can add the item to any branch he chooses as the correct place of the working version of the Master Ontology. This is done by walking through the clinical view of the MO and labelling the parent branch of the MO. By labelling, the item will be automatically added to the MO. The same can be done for a whole ontology/annotation database. This results in an extended MO. A notification of the change will be automatically sent by email to the curators. The curators will discuss new entries either change them or validate them. If inconsistencies exist, tools have to be developed to clear the inconsistencies with the help of the curator of the MO. In every case the link to the item on the CRF has to be maintained. At regular intervals the draft version of the MO will be updated to the standard MO. The next time a clinician needs the same item in a new CRF, he will only run through point one of this section.

It is of utmost importance to realize that by using this process extending the MO will not become a bottleneck for new clinico-genomic trials within ACGT, nor for the flow of ACGT processes in general. It is no longer necessary to wait for an extended version of the MO including a complete set of all items for a new trial. One can start with creating new CRFs for a trial regardless of the content of the MO. By using the trial builder in the above described way, the MO will automatically be extended, and only those items will be added, that are needed and used in clinico-genomic trials. The more clinico-genomic trials will use the trial builder via the ACGT platform, the faster the MO will grow.

The integration of existing data sources via the mediator is the general policy of the ACGT project. Yet the ultimate goal of ontology-based information management in eHealth must be the direct integration of data created in different environments (e.g. clinical research, laboratory data, etc.). ACGT aims to provide solutions that demonstrate the possibility to already create data in an ontology-governed way.

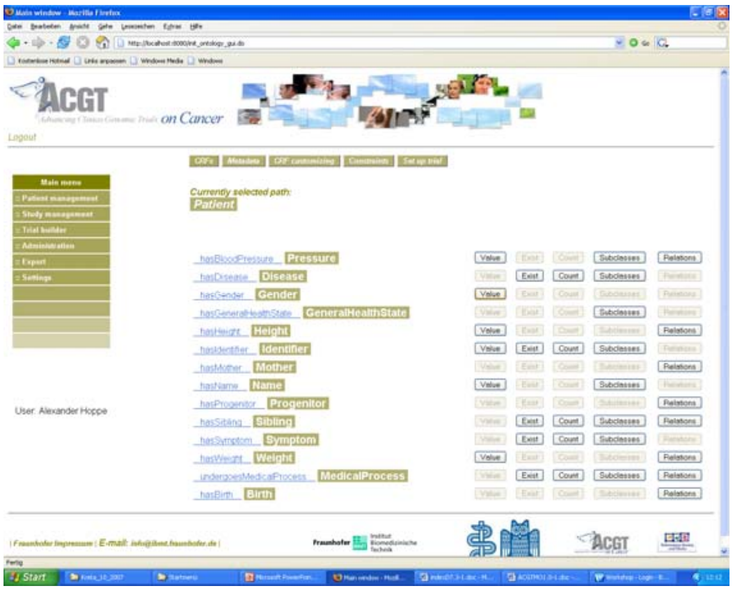
To explore this approach, an Ontology-based Trial Management System for ACGT (ObTiMA) is under development that enables those who undertake clinical trials to set up patient data management systems with comprehensive metadata in terms of the ACGT-MO. This will allow seamless integration of data collected in these systems into the ACGT mediator architecture.

The main components of ObTiMA are the Trial Builder and the patient data management system. The Trial Builder allows a trial chairman to define the master protocol, the Case Report Forms (CRFs) and the treatment plan for the trial, in a way that is both semantically compliant with the ACGT-MO and user-friendly. From these definitions, the patient data management system that allows the collection of relevant data for individual patients can be set up automatically. This collected data is stored in trial databases whose comprehensive metadata has been rendered in terms of the ACGT-MO. The data can thus be seamlessly integrated through OGSA-DAI web services into the mediator architecture. Trial databases with comprehensive ontological metadata and the OGSA-DAI services are both automatically set up from the definitions made by the trial chairman in the Trial Builder.

The process of setting up a clinical trial requires tools to overcome the gap between clinical practice and biomedical reality representation. Even if an ontology provides natural language definitions for its entities and relationships (is, in other words, ‘human understandable’) they are still defined in a way that is not based on practical or clinical perceptions of reality. In order to meet this desideratum, the Trial Builder provides an application specific view on the ontology. This view is meant to assist clinicians in clinical practice, as well as when tackling workflows typical of clinical trial management.

So far a first prototype of the Trial Builder has been developed that shows in principle how the procedure of CRF creation from the ontology can look like. In the following we will describe the functionality as well as the limitations of the prototype. The procedure conducted in the Trial Builder implements the semantics of the ontology in the CRFs in an automatic fashion. The “ontology description” for one item on a CRF is a path from the ontology starting at the class Patient, as this is normally the focal point of CRFs. Starting from the class Patient, for example, the clinician grasps the relevant ACGT-MO classes and relations that connect with Patient, e.g that the patient can have diseases, a blood pressure, a birth date, etc. In figure 17 the ontology view of the Trial Builder that allows the creation of an item from the ontology is shown. The currently selected ontology path is depicted on the top. It initially consists only of the class Patient. Below that the classes and relations that connect with patient are shown. Here the clinician has the opportunity to either select one of the classes in order to create a question on the CRF, or to explore them further and show their subclasses (by pressing the button “Subclasses” behind the selected class) or relations (button “Relations”) and assemble more complex questions. Since we do not want to restrict the questions on the CRFs by the structure of the ontology, different possibilities to create questions from the ontology classes are implemented in the Trial Builder.

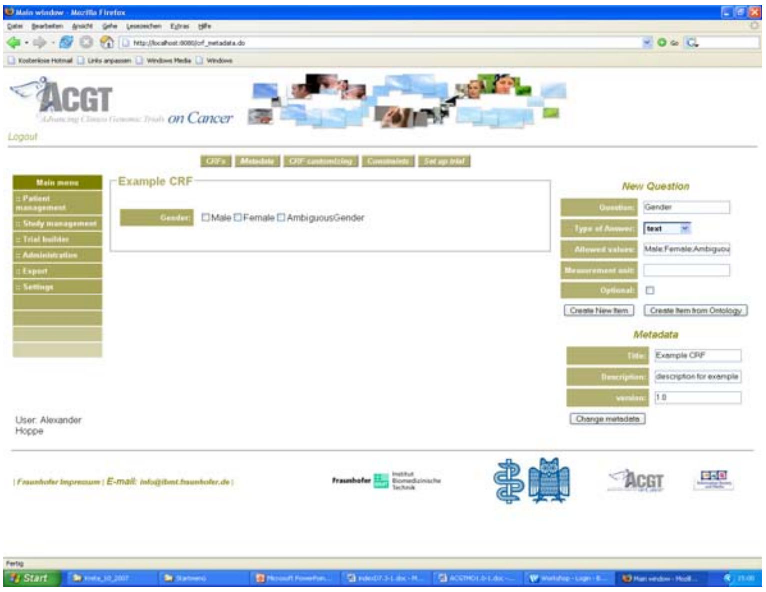
In the current prototype the user is able to create three types of items with respect to the ontology description, that are items of type “Value”, “Exist” and “Count”. For each of them a button appears in the user interface behind the classes.



Currently selected path:
Patient

hasBloodPressure	Pressure	Value	Exist	Count	Subclasses	Relations
hasDisease	Disease	Value	Exist	Count	Subclasses	Relations
hasGender	Gender	Value	Exist	Count	Subclasses	Relations
hasGeneralHealthState	GeneralHealthState	Value	Exist	Count	Subclasses	Relations
hasHeight	Height	Value	Exist	Count	Subclasses	Relations
hasIdentifier	Identifier	Value	Exist	Count	Subclasses	Relations
hasMother	Mother	Value	Exist	Count	Subclasses	Relations
hasName	Name	Value	Exist	Count	Subclasses	Relations
hasProgenitor	Progenitor	Value	Exist	Count	Subclasses	Relations
hasSibling	Sibling	Value	Exist	Count	Subclasses	Relations
hasSymptom	Symptom	Value	Exist	Count	Subclasses	Relations
hasWeight	Weight	Value	Exist	Count	Subclasses	Relations
undergoesMedicalProcess	MedicalProcess	Value	Exist	Count	Subclasses	Relations
hasBirth	Birth	Value	Exist	Count	Subclasses	Relations

Figure 17: Trial Builder View to create an item from the ontology



Example CRF

Gender: ☐ Male ☐ Female ☐ AmbiguousGender

New Question

Question:	Gender
Type of Answer:	text
Allowed values:	Male;Female;Ambiguou
Measurement unit:	
Optional:	☐

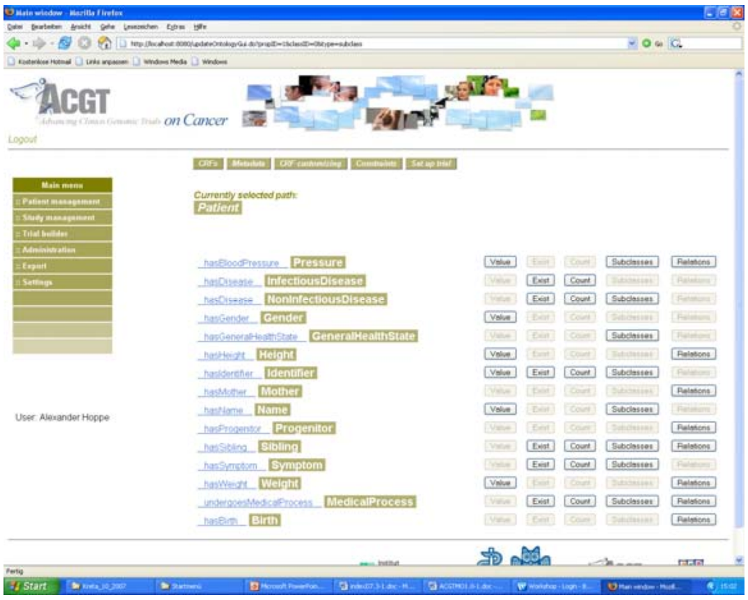
Metadata

Title:	Example CRF
Description:	description for example
version:	1.0

Figure 18: Trial Builder View after creating a value-item from the class gender

“Value-Items” are items that query “values” of attributes from the ontology. Examples are birthdate, gender or weight of the patient’s tumor.

For example the clinician wants to collect information on the patient’s gender on the CRF. He can see that there is a relation between the classes “Patient” and “Gender.” In order to create the corresponding question, the clinician chooses the class Gender by pressing the button “Value” behind this class. The attributes required in order to create the possible answers on the CRF can be determined automatically. On the CRF the label of the question appears as the name of the class (Gender). The allowed values are set automatically to Male, Female, and AmbiguousGender since the class Gender is defined as an enumeration in the ontology containing these values. A multiple choice question has hence been automatically created on the CRF (s. figure 18). The attributes that are automatically determined from the chosen ontology path can be changed in each case manually by the clinical trial chairman. When the clinician e.g. prefers to have the label “ambiguous gender” instead of “AmbiguousGender” he can change that.

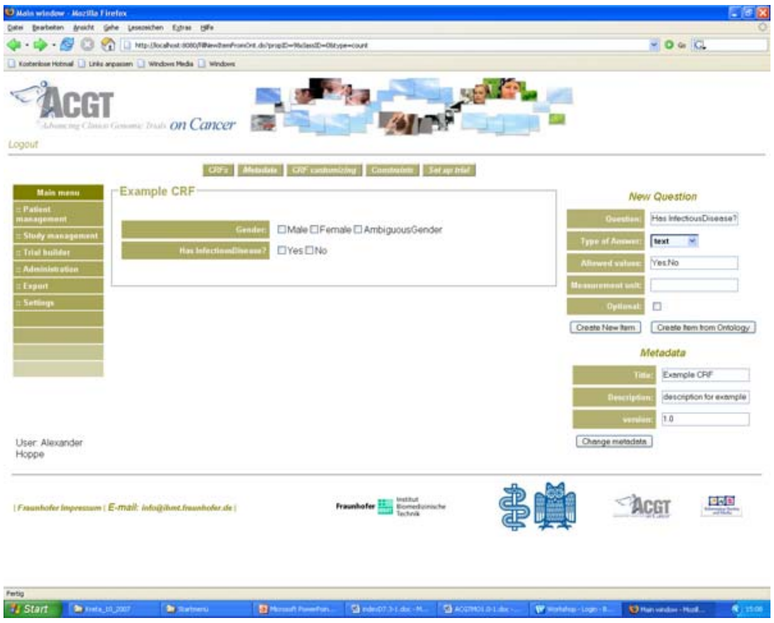


Currently selected path:
Patient

hasBloodPressure	Pressure	Value	Exist	Count	Subclasses	Relations
hasDisease	InfectiousDisease	Value	Exist	Count	Subclasses	Relations
hasDisease	NonInfectiousDisease	Value	Exist	Count	Subclasses	Relations
hasGender	Gender	Value	Exist	Count	Subclasses	Relations
hasGeneralHealthState	GeneralHealthState	Value	Exist	Count	Subclasses	Relations
hasHeight	Height	Value	Exist	Count	Subclasses	Relations
hasIdentifier	Identifier	Value	Exist	Count	Subclasses	Relations
hasMother	Mother	Value	Exist	Count	Subclasses	Relations
hasName	Name	Value	Exist	Count	Subclasses	Relations
hasProgenitor	Progenitor	Value	Exist	Count	Subclasses	Relations
hasSibling	Sibling	Value	Exist	Count	Subclasses	Relations
hasSymptom	Symptom	Value	Exist	Count	Subclasses	Relations
hasWeight	Weight	Value	Exist	Count	Subclasses	Relations
undergoesMedicalProcess	MedicalProcess	Value	Exist	Count	Subclasses	Relations
hasBirth	Birth	Value	Exist	Count	Subclasses	Relations

User: Alexander Hoppe

Figure 19: Trial Builder View after pressing the button “Subclasses” behind the class Disease



Example CRF

Gender:	☐ Male ☐ Female ☐ AmbiguousGender
Has InfectiousDisease?	☐ Yes ☐ No

New Question

Question:	Has InfectiousDisease?
Type of Answer:	text
Allowed values:	Yes;No
Measurement unit:	
Optional:	☐

Metadata

Title:	Example CRF
Description:	description for example
version:	1.0

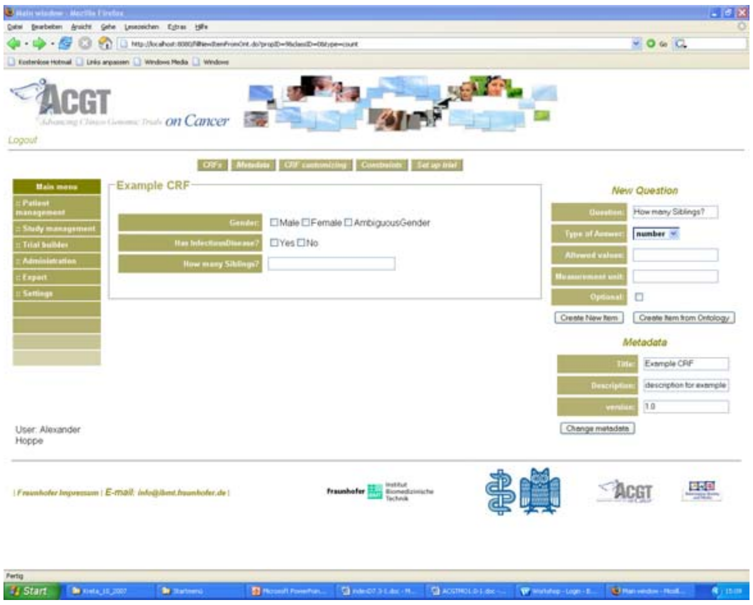
User: Alexander Hoppe

Figure 20: Trial Builder View after creating an exist item from the class InfectiousDisease

“Exist-Items” are items that query if an instance for the selected class exists for the patient. E.g. if the patient has an infectious disease or if the patient has a tumor. To create an item that queries if the patient has an infectious disease the clinician has to press the button “Subclasses” behind the Class “Disease”. The effect is that the subclasses “InfectiousDisease” and “NonInfectiousDisease” are shown in the following view (s. figure 19). The clinician can create the question by pressing the button “Exist” behind the class

"InfectiousDisease". The item is automatically created. The label of the item is assembled from "Has" and the name of the class. This is only a suggestion that is not always correct and therefore can be changed by the clinician. The allowed values are automatically set to "yes" and "no" (s. figure 20).

"Count-Items" are items that query how many instances for a selected class exist for the patient. For example the question how many siblings the patient has. For creating the Item "Count of siblings" the clinician has to press the button "Count" behind the class Sibling. On the CRF the item is automatically created. The suggestion for the label is assembled of "How many" and the class name plus an "s". The data type is set to "number" (s. figure 21).


Example CRF

Gender: ☐Male ☐Female ☐ AmbiguousGender

Has InfectiousDisease? ☐Yes ☐No

How many Siblings?

New Question

Question: How many Siblings?

Type of Answer: number

Allowed values:

Measurement unit:

Optional: ☐

Metadata

Title: Example CRF

Description: description for example

version: 1.0

User: Alexander Hoppe

Figure 21: Example CRF created with the Trial Builder from the ontology

The attributes for setting up the appropriate database for storing the data are determined automatically from the selected ontology paths and the changes the trial chairman applies to them, e.g. the question itself, data type of the answer and optionally possible data values, range constraints and measurement units. The complete ontology annotation is stored in one XML –file for each CRF in the databases for the patient management system. From these files the mapping data for the mediator can be created automatically depending on what information the mediator can "digest" in order to keep the queries feasible.

The described process leads to the direct integration of the data collected in the clinical trial at hand into the semantics of the ontology. Using the ontology will not be time consuming for the clinician, and it will be a useful tool aiding his work in clinical trial management. Through the integration of the ACGT-MO into ObTiMA, data sharing between clinical trials becomes possible in an easy way. This is necessary to leverage the collected data for further research like cross-trial analysis.

The described prototype still has some drawbacks and limitations. Currently it is not possible to create all types of items that are needed on a CRF. Items from more complex descriptions of ontology classes can not be created. It cannot be indicated that two items refer to the same instance. E.g. when creating an item that queries the length of a tumor and one that queries the weight of a tumor it can not be indicated if the tumor in both items is the same or if the items refer to different tumors.

Between two instances in different items no relations can be applied. E.g. when one creates an item that queries the radiation dose for a patient and one that queries the weight of the tumor it is not possible to indicate that the radiation of the one item is applied to the tumor from the other item.

At the moment in each step only one item can be created. It would be very useful, if more than one item could be create in one step. E.g. when selecting the class tumor it could be possible to create items for each of the attributes of this tumor. The usability of the graphical user interfaces has to be improved in order that they are more understandable by clinicians. We are currently redesigning the user interfaces addressing these aspects.

4. Conclusions and Future Work

The present document is a report of the software tools implementation made for semantic mediation in WP7. These tools are an evolution of the first version of the semantic mediation layer, and implement the changes and improvements that were decided upon after the first implementation experience. These changes and improvements are:

- Change query language from RDQL to SPARQL
- Use of an enhanced mapping format
- Development of a simple and fast ontology representation model
- Provide access to a simplified schema of the integrated repository
- Development of a mapping API and tool
- Development of tools to deal with instance level heterogeneities in queries and databases
- Ontology Browsing

It has been decided to separate these new features in different tools. The first version of the Semantic Mediator followed a monolithic approach. From our point of view, it is easier to maintain the software if it is composed by a set of collaborating components. Furthermore, these components can be reused in other environments—e.g. OntoDataClean tool is used as well to make data preprocessing in Data Mining.

In the next phase of the project, we plan to build a friendly, intuitive end-user interface aiding users to construct queries for the mediator. This querying interface will be based in web technologies, and will be made available through the ACGT Portal.

Another important task is improving the performance of the mediation approach. There still exist issues related to the Local as View approach. The first step towards this goal has already been taken, with the constraining of the integrated repository schema. What we plan for future enhancement is to optimize the produced views.

Finally, we have identified that producing the mappings is a bottleneck within database integration process. We plan to speed up the mapping process by designing and implementing a tool that aid users in building the mappings.

Glossary

Term	**Meaning**
API	Application Programming Interface
BFO	Basic Formal Ontology
CRF	Clinical Report Form
DTD	Document Type Definition
EVS	Enterprise Vocabulary Service
FMA	Foundational Model of Anatomy
GUI	Graphical User Interface
KDD	Knowledge Discovery in Databases
KR	Knowledge Representation
LAV	Local as View
NCIT	National Cancer Institute Thesaurus
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SVN	Subversion
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
GWT	Google web Toolkit

References

[1] Prud'hommeaux E, Seaborne A. SPARQL Query Language for RDF. W3C Candidate Recommendation 2007. Available at: http://www.w3.org/TR/rdf-sparql-query/

[2] SPARQL Query Results XML Format. Available at: http://www.w3.org/TR/rdf-sparql-XMLres/

[3] The Java programming language. Available at: http://java.sun.com/

[4] Web Ontology Language (OWL). Available at: http://www.w3.org/2004/OWL/

[5] Antonioletti M, et. Al. The design and implementation of grid database services in OGSA-DAI. In: Concurrency and Computation: Practice and Experience 2005; 17(2-4):357-76.

[6] RDF Vocabulary Description Language 1.0: RDF Schema. Available at: http://www.w3.org/TR/rdf-schema/

[7] Trail: Creating a GUI with JFC/Swing. Available at: http://java.sun.com/docs/books/tutorial/uiswing/

[8] Jena – A Semantic Web Framework for Java. Available at: http://jena.sourceforge.net/

[9] Guarino, N. and Giaretta, P., Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing 1995. IOS Press, Amsterdam: 25-32

[10] Guarino, N. and Poli, R., The Role of Formal Ontology in the Information Technology (editorial). International Journal of Human and Machines Studies (special issue), 43(5/6): 623-624

[11] Guarino, N., Formal Ontology, Conceptual Analysis and Knowledge Representation. International Journal of Human and Computer Studies, 43(5/6): 625-640

[12] Uschold, M. and Gruninger, M., Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review, 11(2): 93-136

[13] ACGT, "User Requirements for an Ontology Based Clinical Data Management System and for the Trial Builder," Project Number: FP6-2005-IST-026996, Deliverable ID: D 2.2 (draft), 2007

[14] Ontology for Biomedical Investigations. Available at: http://obi.sourceforge.net/index.php

[15] Grenon, P., Smith, B., Goldberg, L., "Biodynamic Ontology: Applying BFO in the Biomedical Domain," in: Ontologies in Medicine, D. M. Pisanelli, Ed., Amsterdam: IOS Press, 2004, pp. 20-38

[16] Ceusters, W., Smith, B., Kumar, A., Dhaen, C., "Mistakes in Medical Ontologies: Where Do They Come From and How Can They Be Detected?," in: Pisanelli DM (ed.): Ontologies in Medicine: Proceedings of the Workshop on Medical Ontologies, Rome, October 2003. IOS Press, Amsterdam, 2003

[17] Levy AY, Rajaraman A, Ordille JJ. Querying heterogeneous information sources using source descriptions. In Proceedings of the Twenty-second Internacional Conference on Very Large Databases (VLDB'96), Mumbai (Bombai), India, September 1996; 251-62.

[18] Golbereich, C. et. al., Foundational Model of Anatomy in OWL: experience and perspectives, Journal of Web Semantics, Web Semantics: Science, Services and Agents on the World Wide Web, Volume 4, Issue 3, Pages 181-195

[19] Noy, N. et. al, Pushing the envelope: challenges in a frame-based representation of human anatomy. Data & Knowledge Engineering, Volume 48, Issue 3, pp. 335-359

[20] M. Uschold, M. Gruninger, "Ontologies: Principles, methods and applications," Knowledge Engineering Review , vol 11(2), pp. 93-155, 1996.

[21] G. Klyne, J.J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation, (2004), Available at http://www.w3.org/TR/rdf- concepts/.