TRANSFoRm

Translational Research and Patient Safety in Europe

D5.2: Application of Provenance model

Imperial College London (ICL)

Work Package:	WP5, WT 5.4
Type of document:	Software deliverable
Version:	Final (2.0)
Date:	30th May 2012
Authors:	Roxana Danger, Vasa Curcin (ICL)

TRANSFoRm is partially funded by the European Commission - DG INFSO Under the 7th Framework Programme. **(**FP7 247787)
http://cordis.europa.eu/fp7/ict/
http://ec.europa.eu/information_society/index_en.htm

eHealth
Better Healthcare for Europe

European Commission

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Executive Summary

D5.2: Application of Provenance Model implements the design of the provenance model originally presented in D3.1: Provenance Model, by providing a service based framework with a back-end database and a set of programmatic interfaces that TRANSFoRm software tools invoke to submit their provenance information. Based on the TRANSFoRm's Clinical Research Information Model (CRIM), Clinical Data Integration Model (CDIM), and Clinical Evidence Model (CEM), we define a novel approach of using provenance graph templates to facilitate provenance data capture for various software tools that are part of TRANSFoRm. The approach is demonstrated on all TRANSFoRm tools in sufficiently advanced development stage at present. An analysis and visualisation tool is also presented, with queries defined from the requirements of the three TRANSFoRm use cases.

The key contributions of this deliverable are:

- Concept of provenance graph templates as a novel implementation mechanism
- Provenance templates for Data Quality Tool, Query Formulation Tool, eCRF workbench, and the Decision Support System
- Web service and programmatic APIs for client tools' access to framework
- Distributed database implementation
- Generic provenance query and visualisation tool

Through these, we deliver a consistent methodology for making TRANSFoRm tools provenance-enabled, ensuring both present and future tools can easily integrate with the service interface of the distributed Provenance framework. The gathered data can then be queried and analysed through the Query interface, as demonstrated through the default set of questions provided, e.g. how was a particular process authenticated, or how was the patient recruitment query formulated for a particular study. Thus, we provide full tractability and auditability of the TRANSFoRm work.

The methodology itself is model-based and is generally applicable to heterogeneous software systems with shared process models, which we intend to investigate further in the future.

1. Introduction

This deliverable describes the application of the TRANSFoRm provenance model design, presented in [1], to TRANSFoRm software tools that will be capturing provenance information. Thus, our provenance framework implementation captures the tasks performed in the tools such as Data Quality Tool, Query Formulation Tool, eCRF workbench, Decision Support System and others, and stores them in a semantically annotated database for later auditing and analysis.

Provenance of any piece of data refers to the knowledge about its origin, in terms of entities and actors involved in its creation, e.g. data sources used, operations carried out on them, and users enacting those operations. Making software systems *provenance aware* enables investigation of data sources and services that produced a particular output from the system, together with the individuals who instigated the requests and received those outputs, to establish the exact lineage and assess that correct procedures were followed [2].

The software tools in TRANSFoRm are delivered across the span of four years, and are in various stages of development, which presents the challenge of providing a consistent implementation method that can both address the current needs of tools that are completed, the ones being developed, and the ones that will only commence later in the project. We have addressed this by introducing *provenance templates,* a novel method for making applications provenance aware, that is based on underlying semantic models, such as Clinical Research Information Model (CRIM) [3], but can subsequently be adapted by the tools to include custom ontologies, such as is the case with the Clinical Evidence Model (CEM) to be used in the decision support tools.

The concept of provenance templates simplifies the provenance information capture for the client tools, by defining standard patterns of usage for each tool, and the corresponding provenance graph families that such patterns can produce. We have designed these patterns for all the main tools in TRANSFoRm (Data Quality Tool, Query Formulation Tool, eCRF workbench and Decision Support System), and provided mechanisms for the later tools to define patterns and underlying semantic representations themselves (Data Mining Tools, Linkage Tools, and Mobile Applications). A lightweight approach, with no additional ontologies, and only textual annotations is also described. This is demonstrated on actual code fragments produced in collaboration with relevant partners.

To support auditing and querying of stored provenance data, we provide a lightweight web-based query tool that can be easily integrated into other TRANSFoRm tools (e.g. the workbench) and that supports pre-designed queries, addition of new queries, and visual exploration of the provenance items by browsing the graph structure. Query examples given have been developed from the requirements of use case partners.

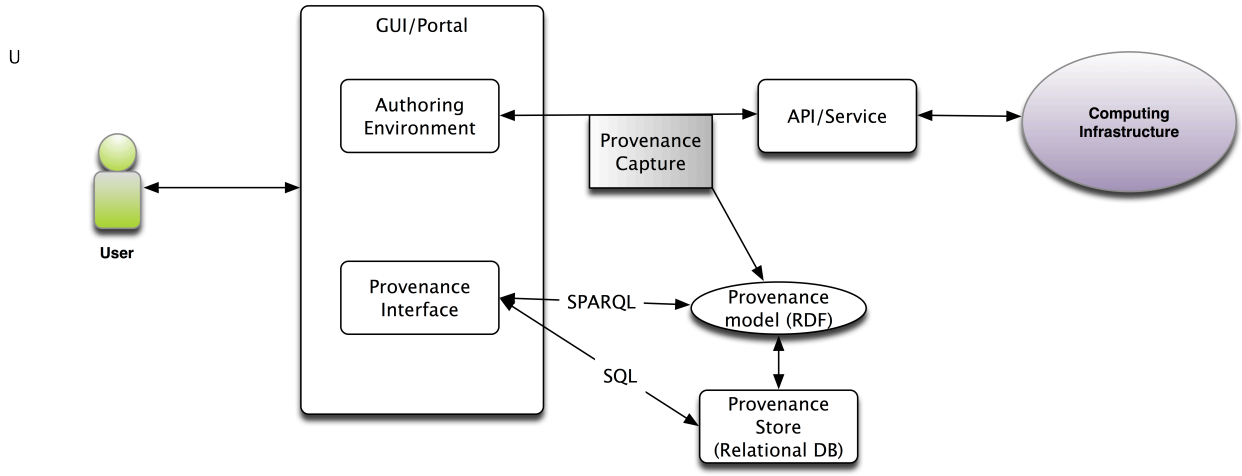

Figure 1: Provenance framework overview

1.1 Overview of the provenance framework

The provenance model for TRANSFoRm was originally defined in [1] as shown in Figure 1. The principal decisions concerning the model were:

1. Create a distributed provenance system based on the Open Provenance Model,
2. Use semantic technologies for data provenance description,
3. Use relational database technology for data provenance storage, with RDF-relational mapping for providing both relational and semantic query view of data provenance

The Open Provenance Model (OPM) [4] is a community standard for provenance description. The provenance information in OPM is represented as directed acyclic graphs (DAG) establishing causal relationships between individual nodes, which represent

artifacts (data entities that are referenced in the system), *processes* (that produce artifacts), and *agents* (physical or software entities that control processes), together with causal dependences between them. Digital representation of provenance elements is also an essential requirement in any provenance model. OPM has adopted Uniform Resource Identifiers (URIs) as the protocol to define the OPM elements, and is advocating the usage of Rich Data Format (RDF) and Ontology Web Language (OWL) technologies for OPM graph format storage and interpretation, to which goal OPMV and OPMO ontologies have been produced [4]. This approach, for example, enables the TRANSFoRm provenance framework to refer to confidential data by using URIs to the actual data, and letting the data sources themselves determine whether the user can access this information.

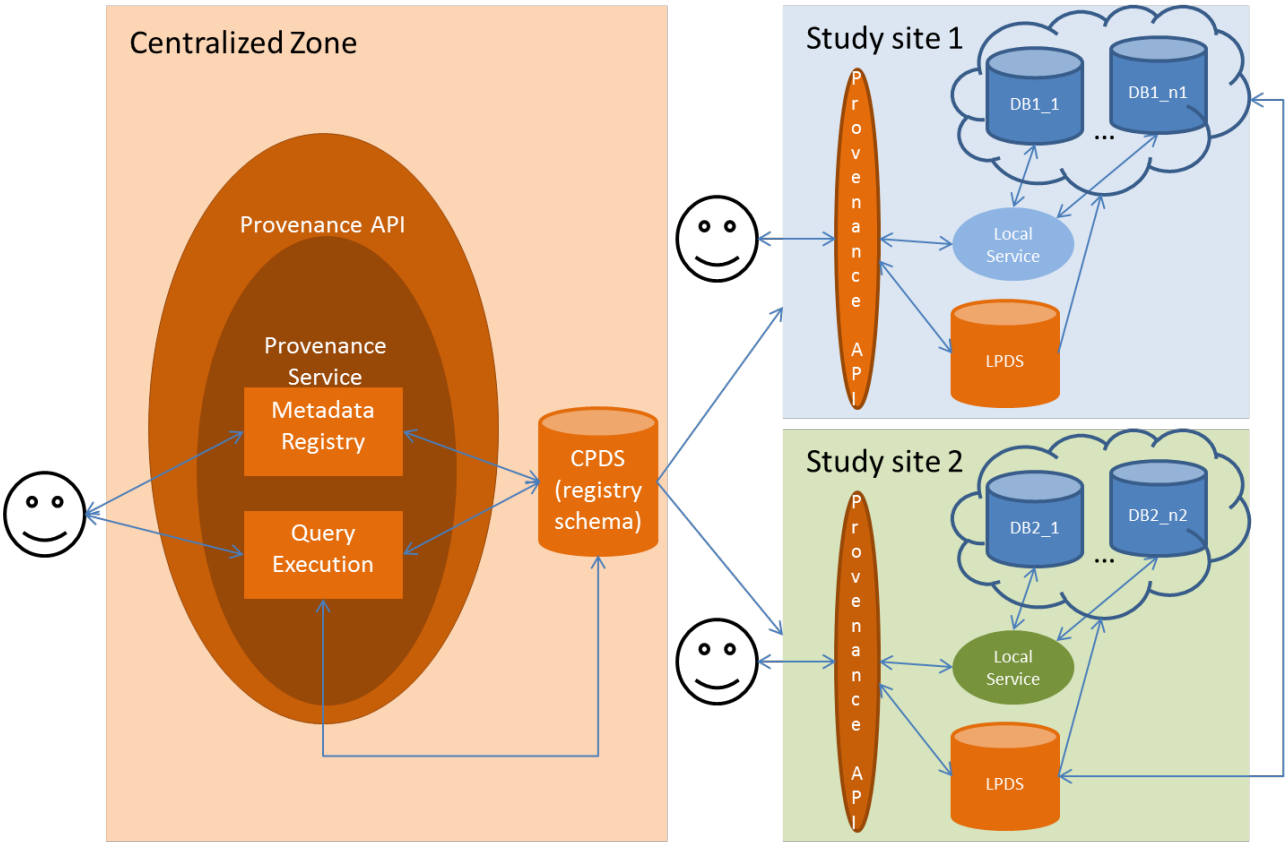

Figure 2: Architecture of the distributed provenance system

Figure 2 shows the distributed architecture of the TRANSFoRm provenance framework, which consists of a centralized zone, and a number of local study sites. Each study site contains a set of databases, local TRANSFoRm services and tools, and a provenance service interface for recording provenance information into the Local Provenance Data Store (LPDS). The Centralized Zone maintains the information about all study sites, and offers services for resolving queries about distributed provenance graphs, that is, graphs which incorporate processes executed and artifacts produced at different study sites.

The access to study sites for each user from the centralized zone is controlled through the *Resource Registry Service,* which uses the Security Solution Layer to check the authorizations, with temporary federated views of the data being stored in the Centralized Provenance Data Store (CPDS).

1.2 Semantic aspects of provenance

Provenance is intricately linked to semantic technologies from its outset. Semantic framework for managing provenance data creates an unambiguous context for understanding and comparing data [5], which lies at the very heart of provenance. Clinical research data in particular, with its rich semantic information reflected in multiple coding schemas, demands such a framework in order to establish a close mapping to domain concepts that are curated, precise, and understandable to researchers.

The provenance implementation approach described in this document has been specifically designed for allowing semantic provenance annotation: each entity of a provenance graph refers to an ontology concept, drawn either from a domain ontology or core OPM profile ontology.

Within TRANSFoRm, at the time of writing, three domain ontologies have been developed: the Clinical Research Information Model (CRIM), the Clinical Data Integration Model (CDIM), and the Clinical Evidence Model (CEM); and one ontology profile, Clinical Research Information Provenance Profile (CRIPP). The concepts associated with these ontologies are used by TRANSFoRm tools to annotate the provenance entities created, and used to formulate queries on the stored provenance data.

1.3 Database technology for data provenance storage

Although OWL technology, commonly associated with RDF format repositories, is used for annotating provenance data, we have chosen relational database approach to storing provenance data instead. The main reason for this is that the volume of provenance data can grow quite rapidly in the working system, compared to some other TRANSFoRm software, e.g. the decision support tool, and thereby requires optimized relational storage.

The database schema implemented in TRANSFoRm relies purely on standard SQL constructs, and as such can be implemented in any number of standard SQL database management systems, such as Oracle, SQL Server, Postgres, and MySQL. For our initial implementation, we have decided on MySQL, due to its free availability, popularity, support for remote access, and documentation quality. However, there is nothing preventing future adopters from implementing an identical database in any of the other listed databases, with only minor configuration modifications.

Despite the relational back-end, the framework still operates on provenance graphs with semantic annotations, and semantic views of the data will be provided using a Data-to-Relational-Query (D2RQ) tool [6]. D2RQ language describes mappings between relational database schemata and OWL/RDFS ontologies, allowing access to RDF views on a relational data through Jena [7] and Sesame [8]. Thus, two modes of querying are supported: relational, which accesses the stored entities directly, or semantic, using SPARQL [9] queries on the RDF representation of the OPM entities. The former allows formulation of queries that can retrieve non-domain-specific information (i.e. when was a study recruitment query created), while the latter is better for broader queries that may operate on complex, semantic queries (i.e. which presented symptoms previously caused the decision support system to come up with a specific diagnosis?).

1.4 Document outline

Section 2 of this document presents provenance templates used to implement provenance capture for TRANSFoRm tools. In Section 3 the underlying implementation of the provenance server is described, together with its API connectivity, internal architecture, and configuration. In Section 4, the underlying database schema and the distributed storage and querying aspects are detailed. Section 5 shows concrete code examples of how tools in TRANSFoRm use the provenance server, while Section 6 demonstrates provenance

data querying and how it can be used for analysis. Finally, in Section 7, the document is summarized and its implications for TRANSFoRm and broader provenance research are listed.

2. TRANSFoRm approach to implementing provenance

In this section, we will present the strategy for provenance data capture by TRANSFoRm tools, using the examples of the four tools that are in mature design and/or development stages: Data Quality Tool (WT5.1), Query Formulation Tool and eCRF workbench (WT5.3), and the Decision Support Tool (WT4.4). The components that will be developed at later stages of the project, WT4.5 Data Mining Tools, WT7.5 Distributed infrastructure authentication mechanisms, and WT7.6 Mobile middleware, will use the same technique to capture information that reflects their domain.

At the heart of the TRANSFoRm provenance implementation lays a novel concept of *provenance graph templates*: abstract graph patterns, based on the CRIM model, that each tool uses to model provenance information relating to its functionality. In this way, the data that each tool is storing, is explicitly specified, simplifying their interactions with the provenance API, which consists of parameterized method calls that instantiate and store concrete provenance graphs in the background.

This section will first introduce the syntax for both provenance graphs and provenance template graphs, before presenting the templates for the major tools in the system.

2.1 Provenance graph syntax

The TRANSFoRm provenance graphs are using the Open Provenance Model (OPM) [4] for its conceptual representation. OPM is a causal graph model, with edges denoting causal relationships (X was caused by Y) and nodes representing the individual occurrences of some entities. This structure allows OPM graphs to describe how the full lineage of a piece of data in terms of multiple events (process instances) that led to it being produced.

Nodes can be of three types: *artifacts*, *processes,* and *agents*.

- Artifacts are pieces of data of fixed value and context, e.g. one version of a data set, or a document. They are denoted by circles in the provenance graph.
- Processes are actions that are performed using artifacts to generate other artifacts, e.g. a selection process uses the full set of eligible patients and generates a subset of these with which to conduct the trial. Processes are represented by rectangles in a provenance graph.
- Agents are the entities controlling process execution, either as human actors (researchers, clinicians, etc.) or as non-mutable pieces of software (e.g. an EHR system would be an agent, since its provenance is outside of the scope of the TRANSFoRm model). They are shown as octagons in the provenance graph.

The various properties of artifacts, processes and agents are represented by arbitrary key-value *annotations* to the nodes. Edges can also have annotations to provide further information on how one occurrence caused another. Some other aspects of OPM, such as accounts, are unnecessary and therefore omitted from our model.[1]

[1] OPM allows for multiple levels of granularity of description of the same set of past events, with each description being a separate *account* of what occurred. Specifically, what is represented as a single process in

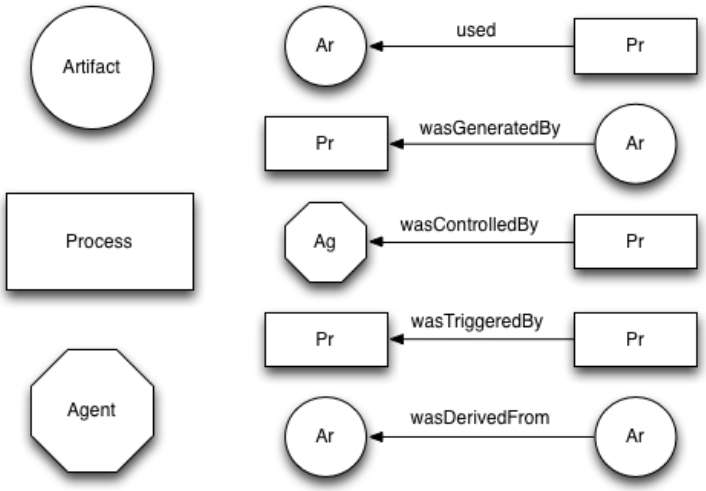

Figure 3: Provenance graph syntax

The OPM constructs are summarized in Figure 3. With regards to graph nodes, circle nodes represent artifacts, rectangles are processes, and the octagons denote agents. The meaning of an edge depends on the nodes it is connecting, so an edge from an artifact to a process is of type *wasGeneratedBy*, from process to artifact is of type *used*, from process to process type is *wasTriggeredBy*, from artifact to artifact it is *wasDerivedFrom*, and from process to agent it is *wasControlledBy*. These types are omitted from the figures for clarity, and are inferred from node types. The complete OPM syntax description can also be found in D3.1 [1] and OPM specification [4].

2.2 Provenance graph templates

Provenance graph templates use graph syntax similar to the provenance graph specifications (Figure 3), but its nodes refer to domain concepts instead of individual occurrences. Thus, template graphs describe predefined system actions whose concrete executions are included in OPM graphs.

2.2.1 Node labels

Node labels have the format:

`<concept>:<instance>-<temporal_descriptor>`

`<concept>` is a domain concept associated with the entity, which belongs to a defined domain ontology, for example CEM in the decision support provenance graphs. `<instance>` is a name that identifies the entity in the graph, its corresponding nodes in the concrete graphs will have a unique identifier provided by the provenance framework. Finally, `<temporal_descriptor>` is a descriptor of the entity graph of the form: `<type><number1>(_<number2>)?`, where `<type>` is `P` if the entity is a process, `A` if it is an artifact and `Ag` if it is an agent; `<number1>` is an index for specifying the order on which the entity is generated/used. If an equivalent entity is generated/used in parallel,

one account can be refined in another account to describe how that process is decomposed into multiple sub-processes with intermediate data being generated and used.

<number2> appears in the temporal descriptor and denotes an index in the parallel processing.

2.2.2 Node colouring

Nodes with different background colours represent that they have been produced, and therefore stored, in different provenance zones. While the majority of processes will create local graphs, there is occasionally a need to specify a provenance template that encompasses several physical locations, and colouration provides a simple way of capturing that information.

2.2.3 Multiple execution instances

In template graphs parallel or serial processing are enclosed in polygons with thick edges, and with the pair <cardinality, processing_type> in the upper left corner, where the cardinality define the number of processing of the same type that can be executed and it is expressed using a mathematical notation (e.g. $\mathbb{N}$ for all natural numbers including 0; $\mathbb{N}_{>0}$ for natural numbers excluding 0; (3, 5], for the set {4, 5}); and the processing_type is P if the enclosed graph will be processed in parallel, and S if the enclosed graph will be processed in serial.

2.3 Query formulation tool

The query formulation tool in TRANSFoRm allows the query formulation, generation, searching and data retrieval to support identifying and recruiting eligible patients for clinical studies.

In Figure 4 a Provenance template for query execution process is shown. This process involves 3 different logical layers of the TRANSFoRm architecture, and at least 3 different tools. The layers are: the upper layer where the query tool is located, the middleware layer and the lower layer where the local services reside. The entities generated in the upper layer are drawn with white background, the entities generated in the middleware are highlighted in red and those from a local area are highlighted in blue.

The overall process begins when a researcher (Ag1) use the Query tool (Ag2) for executing the query (P1) associated to the eligibility criteria (A1) of a clinical study. For the query execution process (P1), the researcher needs also to identify the set of databases (A2) that should be used for recovering the total number of suitable patients.

The Query execution process (P1) triggers a query request process (P2) for executing the query in the different databases, which is the responsibility of the middleware. When the Query Middleware Tool (Ag3) receives a query request (A3), a parallel processing is triggered. Each thread of this parallel concerns to one for databases in A2 and it runs as follows. The eligibility criteria are first translated, in local query creation process (P4), to a query (A4) according to the data structure of the specific database. The resulting query (A4) is sent to the specific Local Query Service (Ag4) that controls the database access and then, a process (P5) waits and captures the

query result (A7). This result is generated in the local area, when a query execution process (P6), controlled by the Local Query Service (Ag4), is executed using as input the query (A4) that had been previously generated in the middleware. The capturing process (P5) generates a query result package (A8) and sends it back to the Query tool.

Once the Query Execution process (P1) has sent the request to the middleware, it triggers a process for acquiring the results from the middleware (P7) and other for aggregating the recovered results (P8). The query result acquisition process (P7) uses the query result package from the middleware (A8) and recover a query result (A9), which is used, in turn, used to aggregate the results of the parallel processing (P8) and finally obtain the aggregated result (A10). As the input of the query execution was a eligibility criteria, the query results (A7, A9, A10) are integers representing the total number of suitable patients in the local area (A7 and A9) and the total number of suitable patients for the clinical study (A10), which sums up the partial results from the local areas.

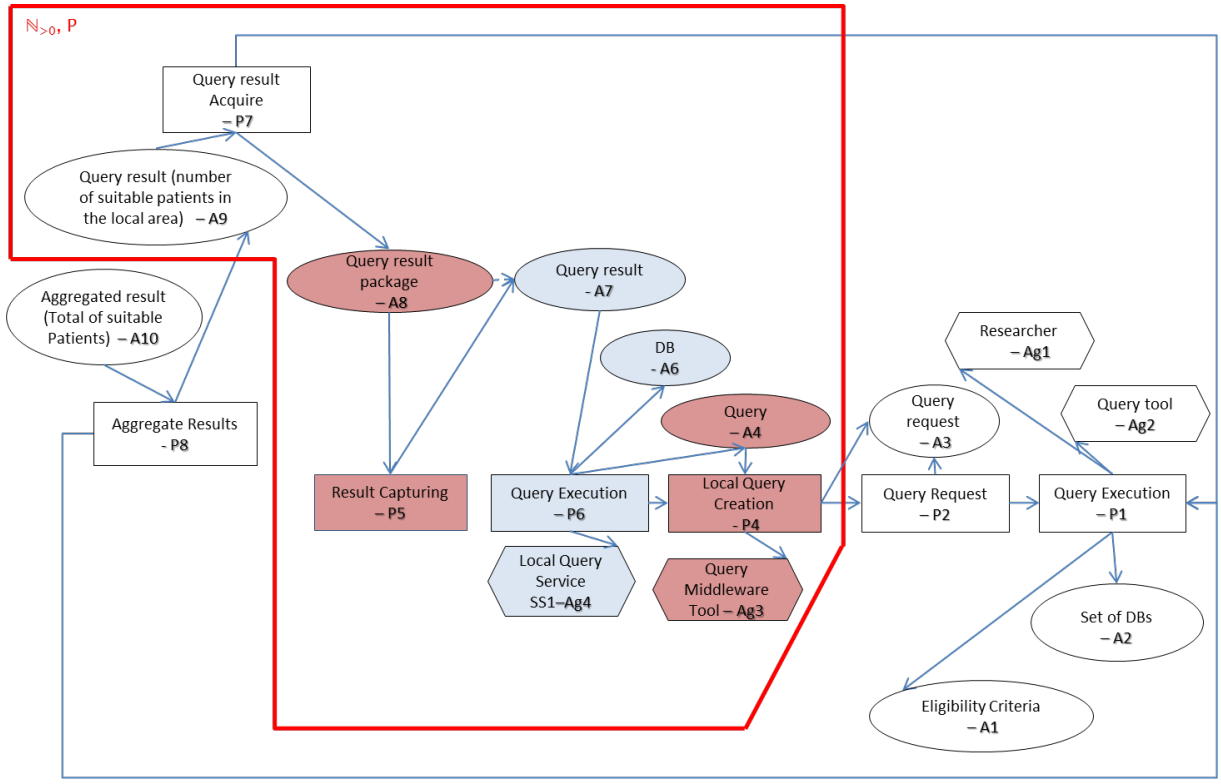

Figure 4: Provenance template for query execution.

A concrete provenance graph of a query execution is shown in Figure 5. Notice that instances associated with each node are identified, and two local databases were queried (in the graph the nodes of each local area are highlighted in blue and in green to differentiate the respective area).

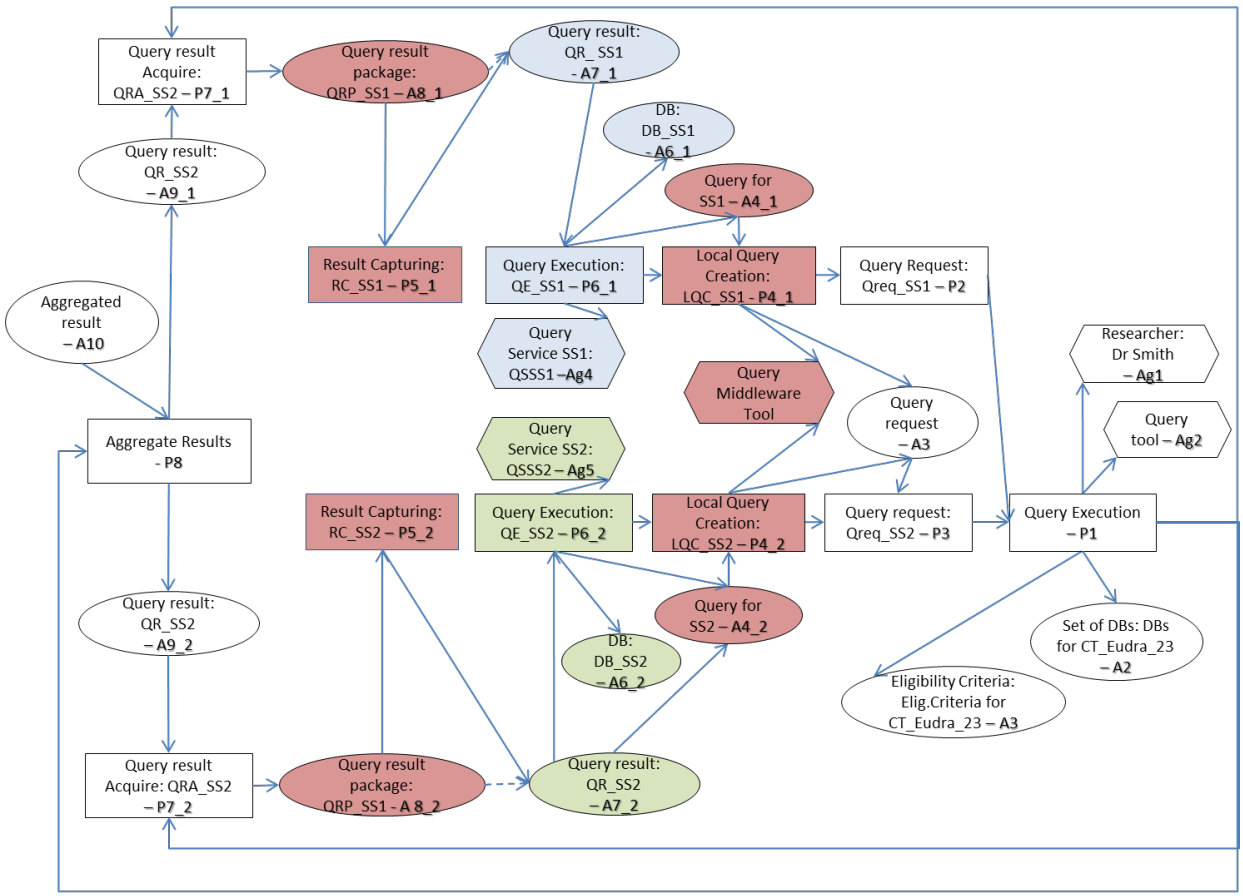

Figure 5: Provenance graph example of a query execution.

The definition of eligibility criteria is a complex process, provenance information of which can be used for change tracking and understanding the design process and the factors behind the success or failure of study recruitments. The full model description is available in D6.2 [3], and only the basics are outlined here.

The full eligibility criteria for a query are *logic or temporal conditions* consisting of inclusion and exclusion *criteria groups*. Each criteria group is, again, a logic or temporal condition joining a set of *criteria*. Each single criteria element is defined by pairs of terms and their admissible values. Admissible values can be expressed using relational (e.g. >, ≥, ==, <, ≤) and temporal (e.g. before, within) operators. An example of a full eligibility criteria in TRANSFoRm Diabetes 2 Use Case is:

$$[C_1 \text{ AND } (C_2 \text{ or } (C_3 \text{ AND } C_4) \text{ or } C_5 \text{ or } C_6 \text{ or } C_7)] \text{ AND NOT } [C_8 \text{ OR } C_9]$$

formed by two criteria groups (expressions between squared brackets) and nine single criteria, C_i, i=[1..9]:

a) Inclusion Criteria: $[C_1 \text{ AND } (C_2 \text{ or } (C_3 \text{ AND } C_4) \text{ or } C_5 \text{ or } C_6 \text{ or } C_7)]$ where:

C_1: Age >= 35;C_2: medication = Insulin (frequency < 3 per day)
C_3: medication = insulin ; C_4: medication = ogla; C_5: medication = ogla;
C_6: medication = metformin; C_7: absence of drug therapy

b) Exclusion Criteria: [C_8 OR C_9], where
C_8: diagnosis = T1D; C_9: diagnosis = endogenous insulin

Based on this, provenance templates are specified for eligibility criteria definition. Figure 6 describes the general framework when an eligibility criteria is defined: using the `Eligibility criteria tool` (`Ag1`), inside the query tool, an user (`Ag2`) can begin `Eligibility criteria definition` process (`P1`), which triggers first a process to create an empty eligibility criteria (`P2`), and then, a series of `Eligibility criteria updating processes` (`P3`) which takes a previous eligibility criteria (`A1`) to obtain a subsequent versions of it (`A2`).

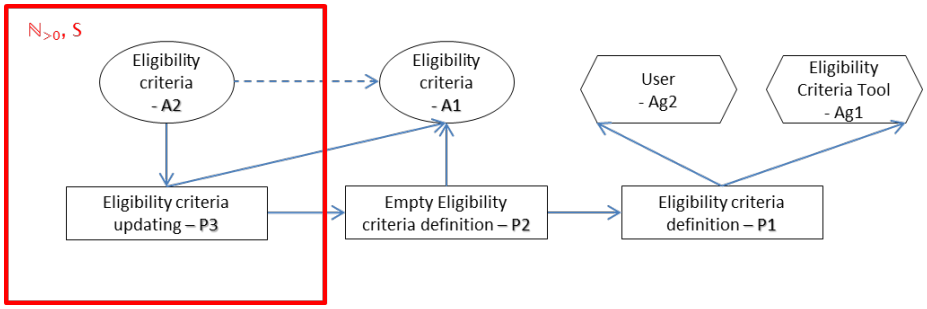

Figure 6: High level provenance template for eligibility criteria definition.

Figure 7 define the OPM template for updating eligibility criteria by adding a new Criteria group, and Figure 8 the OPM template when a Criteria group is deleted. Both templates begin after the user selects the eligibility criteria to update (`A1`). In the first of these figures, the `name of the criteria group` (`A2`) must be defined during the process of `adding new criteria group` (`P2`) and a new empty `criteria group` (`A3`) is generated and then added to the eligibility criteria `A1` to obtain its new version, `A4`.

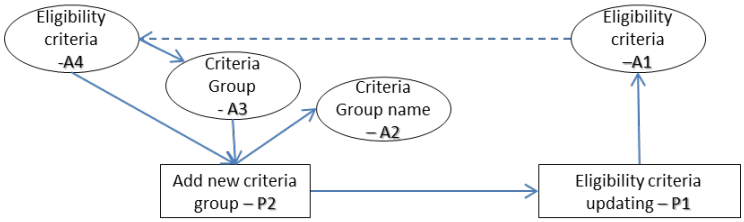

Figure 7: Provenance template for adding criteria group during eligibility criteria definition.

In the case of Figure 8, after selecting the eligibility criteria (`A1`) and the group inside it (A2), the process for `deleting` (`P2`) it will create, as in the previous template, the new version of the eligibility criteria (`A3`).

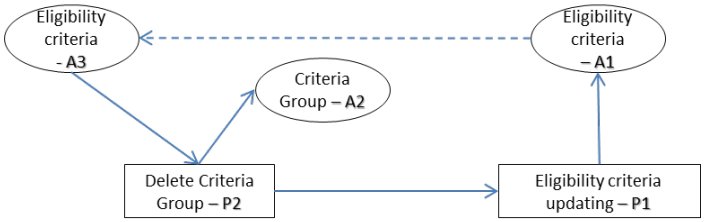

Figure 8: Provenance template for deleting criteria group during eligibility criteria definition.

Figure 9 shows an OPM graph template for the process that allows a user to add a criterion to a criteria group of an eligibility criteria. First, during the process of adding Criterion to Criteria Group (P2) an empty criterion (A3) is created, for a previously selected criteria Group (A2), and the process for criterion definition begins. First, the user has to select a term (P3) to associate to the criterion, using the Controlled vocabulary Service (Ag1). The selected term (A5) is then used to retrieve, from CDIM ontology (A6), the criteria type attributes (A7 generated by P4) and the data sources specific data attributes (A8) that needs to be filled for completing the criterion definition. Using these data (A7 and A8), the eligibility criteria definition interface, allows the user to fill the attribute values (P5) which completes the criterion definition (A9), an update version of the empty criterion (A3). The new criterion is finally used, in the last step (P6), to update the eligibility criteria obtaining its the last version (A10).

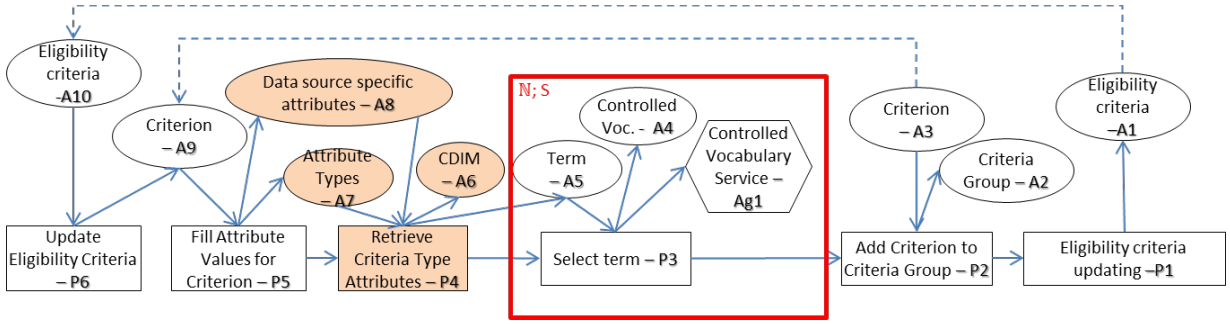

Figure 9: Provenance template for adding a new criterion during eligibility criteria definition.

Figure 10 shows a provenance template similar to the graph shown in Figure 9, with the difference that the triggered process after P1, allows the user to select the criterion (A2) to update the single criteria (P2). Notice that in both graphs the user is allowed to search multiple times for terms in the controlled vocabulary. The last selected term is the one to be used to for updating the criterion.

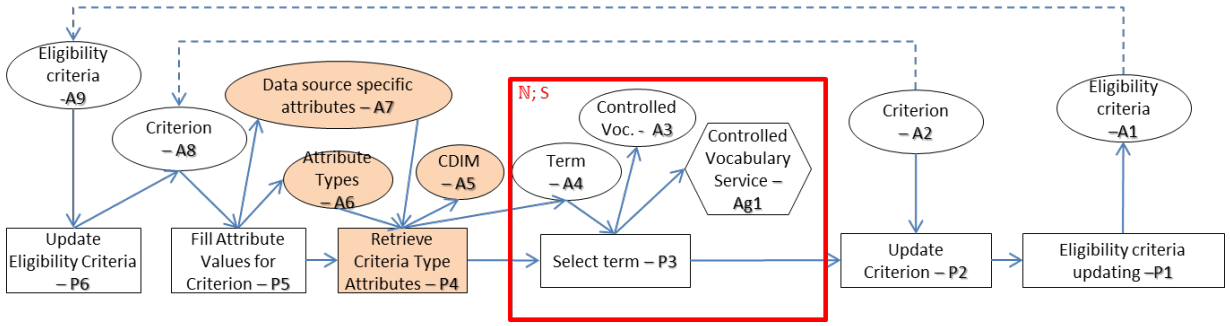

Figure 10: Provenance template for updating a criterion during eligibility criteria definition.

Other two basic operations are needed to update the eligibility criteria: define the operators amongst the criterion in each criteria group (Figure 11), and define the operators amongst the criteria groups (Figure 12).

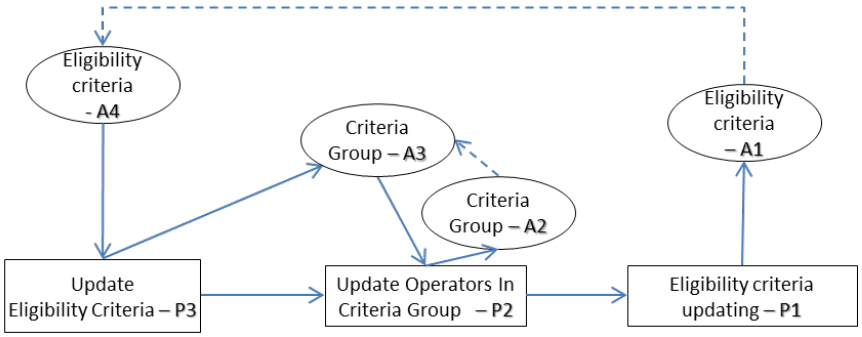

Figure 11: Provenance template for updating operators in a criteria group.

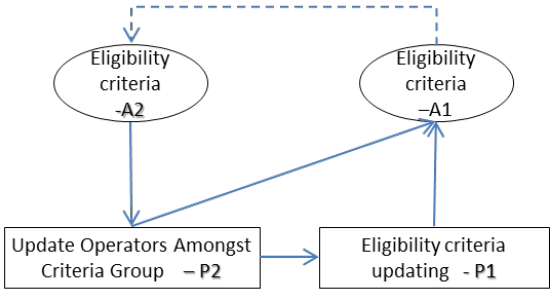

Figure 12: Provenance template for update operators amongst criteria group.

Figure 13 shows an OPM graph example, using the previous templates (Figure 6-Figure 12) for constructing a simplified version of the eligibility criteria example given at the beginning of this section ([Age >=35 AND medication= insulin, frequency < 3, temporal restriction = within 1 day] AND NOT [diagnosis = T1D]). The individual criteria elements are denoted with *x in the diagram, denoting: (*1) Age >= 35; (*2) medication= insulin, frequency < 3, temporal restriction: within 1 day; (*3) Age >=35 AND medication= insulin, frequency < 3, temporal restriction: within 1 day; (*4) diagnosis = T1D; (*5) [Age >=35 AND medication= insulin, frequency < 3, temporal restriction: within 1 day] AND NOT [diagnosis = T1D] (*6) [Age >=35 AND medication= insulin, frequency < 3, temporal restriction: within 1 day] AND NOT [diagnosis = T1D].

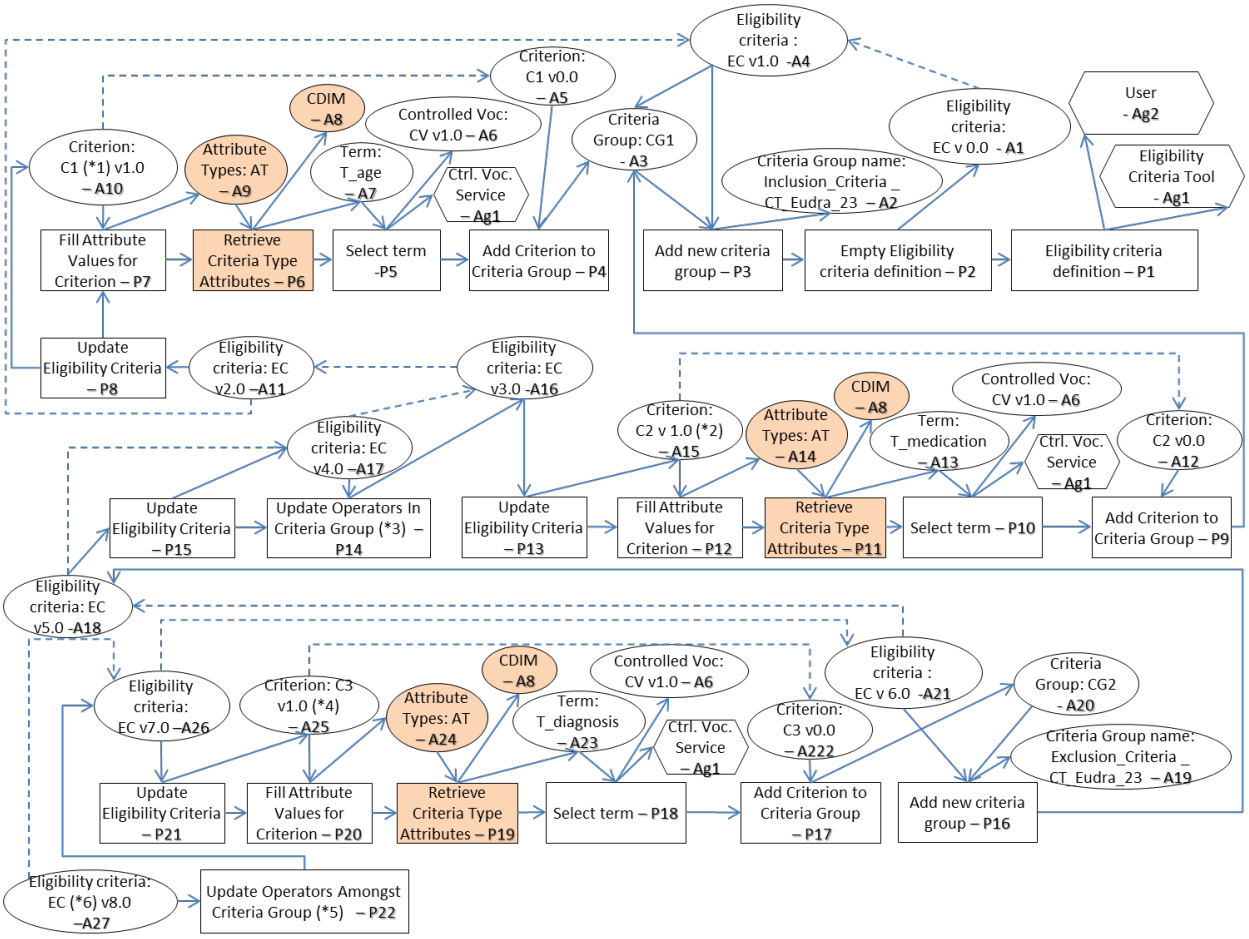

Figure 13: Provenance graph example of a criteria definition.

2.4 Data quality tool

The Data Quality tool (D5.1) aims to assess the quality of primary care data in TRANSFoRm data repositories for research purposes. The tool will maintain a repository of data quality measure definitions that can be used to select a suitable data set for a particular research task.

A provenance template for the task of researcher adding a new quality measure to the data quality measure repository is shown in Figure 14. A new version of the `Data quality measure repository` (`A3`) is generated from the older one (`A2`), after the `Update quality measure repository process` (`P2`) is executed. The process `P2` used a `Quality measure` (`A1`) defined during `New quality measure definition process` (`P1`), which is executed by a researcher (`Ag2`) using the `Data quality tool` (`Ag1`).

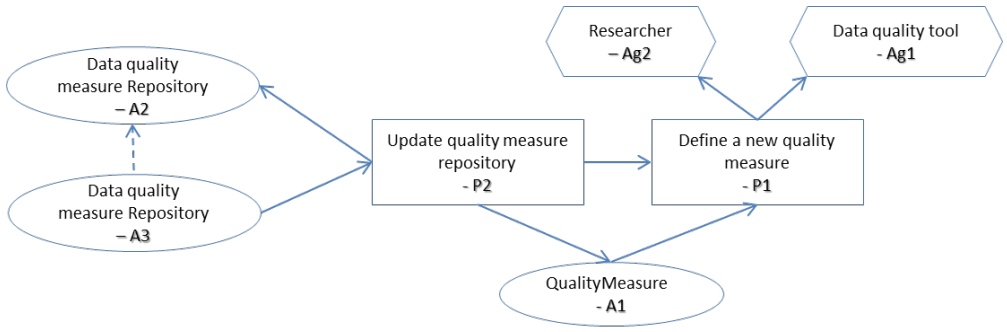

Figure 14: Provenance template for adding a new quality measure to a quality measure repository.

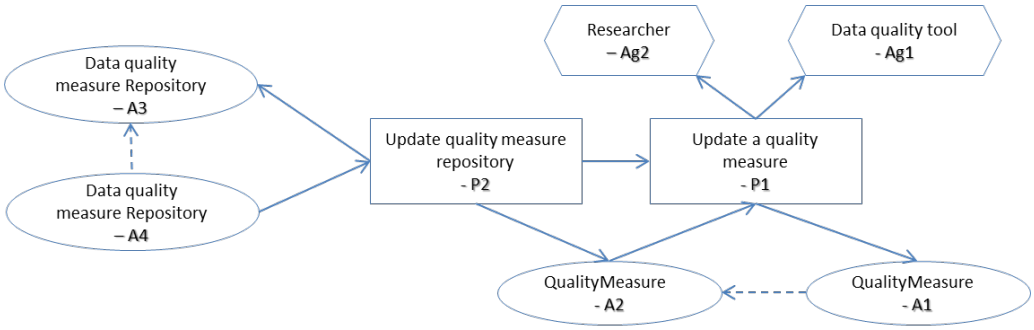

Figure 15: Provenance template for updating a quality measure in the quality measure repository.

In Figure 15, the new version of the repository is created after the processes `Update repository` (`P2`) and `Update quality measure` (`P1`) have been executed by a researcher (`Ag2`) using the `Data Quality tool` (`Ag1`) to update a single quality measure.

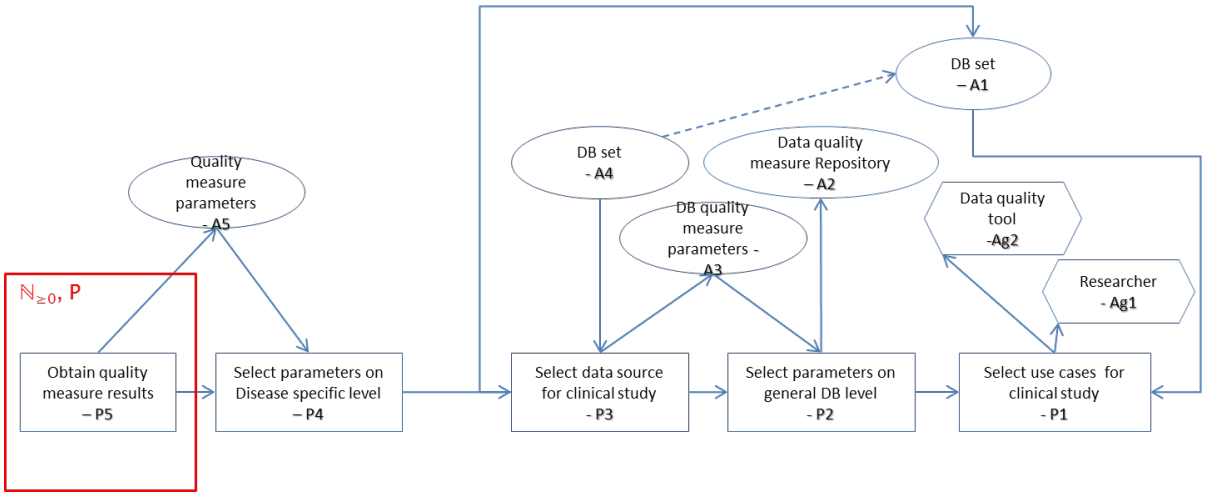

Figure 16: Provenance template for data quality evaluation.

Figure 16 shows a provenance template describing how a researcher can obtain the quality measures of the data in the available research databases. The first step in the process is `Select the use cases (diseases)` (`P1`) that the `Researcher` (`Ag1`) executing the `data quality tool` (`Ag2`) will study. The output of this process is `Set of available databases` (`A1`). Following this, the researcher selects `Set of data quality parameters related with quality of databases` (`P2`), using the

available quality measures in the Data quality measure repository (A2) and generating, in this way, an artifact containing such parameters (A3). The process Select data sources for the clinical study (P3) is then executed. This process uses the Database quality measure parameters (A3) in order to obtain from A1 a subset fulfilling the parameters in A3. At that point, another set of parameters (A5), in this case at level of the use case study (e.g. year for which the study will be performed, set of quality measures for the study under consideration, and their admissible value range), has to be specified (P4), and then used for Obtain the quality measure evaluations (P5) for each database in A4.

The process of obtaining the results is described in the provenance templates in Figure 17 and Figure 18. The template in Figure 17 describes how to obtain these results through an explicit Request (P2) to the Database provider (Ag1). Therefore, the database provider (Ag1) is responsible for evaluating the quality measures (process P3) in his database and for returning back the Data quality measure evaluation (A3) and some other Data quality general information of his database (A4). Finally, the Data quality measure tool will make use of the Provenance server (Ag2) for storing the returned values.

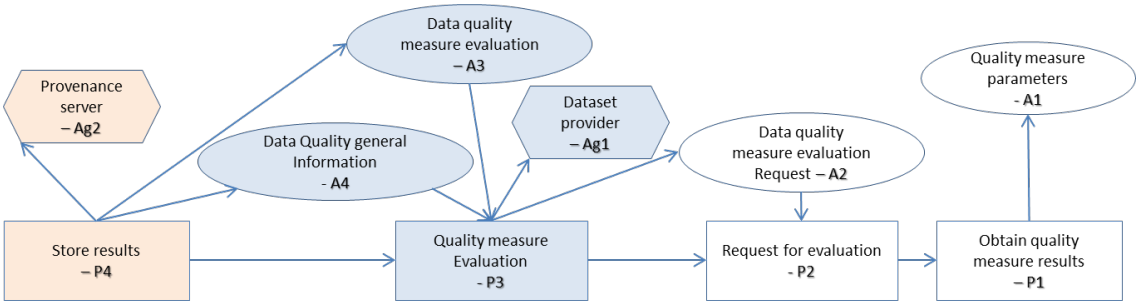

Figure 17: Provenance template for quality measure evaluation.

Figure 18 describes how the provenance server can be used to obtain the quality measures of previously evaluated databases. In this case, a Provenance query (A2) must be formulated and executed (P2) by the Provenance server (Ag1), which returns back previously stored values (A3 and A4). If no evaluation values are available in the provenance store (that is, this database has not been previously evaluated for the requested data quality measures), the provenance server returns null values, and the data quality tool must use the processing described in Figure 17 (request the evaluation to the database provider) to obtain the requested quality measures.

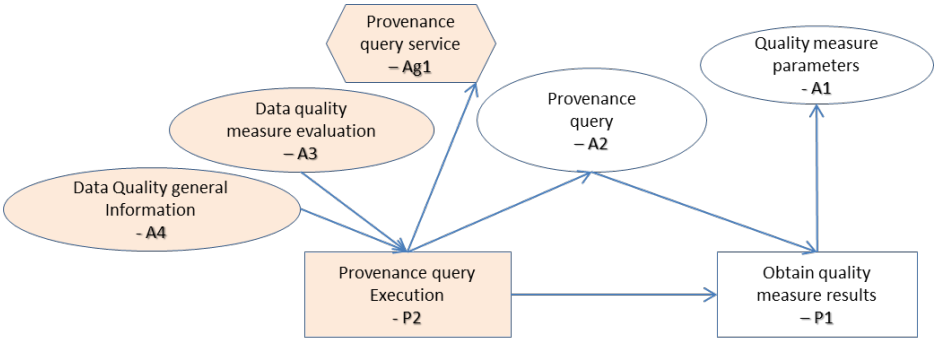

Figure 18: Provenance template for retrieving quality measure results from provenance storage.

2.5 Clinical trial study workbench

The Clinical trial study workbench provides the functionality for controlling clinical trials being conducted within TRANSFoRm. It includes all the phases in the clinical studies, with special emphasis on the eligibility criteria and the eCRF (electronic Case Report Form) lifecycle management, as specified in the CRIM model [3].

Figure 19 shows a fragment of a provenance graph associated with the trial planning phase of a clinical trial. The process `Trial planning` was controlled by agent `Study coordinator Dr. Parker`. Four processes were triggered during this planning phase. The `Trial master file and essential document preparation`, `Labs identification, qualification and preparation` and `Monitoring plan development processes` generated the artifacts `TrialMasterFile TMF_EudtaCT_23`, `LabsFileDescription LF_EudraCT_23` and the `MonitoringPlan MP_EudraCT23`, respectively. Trial master file and essential document preparation used the `Guideline for Good Clinical Practice document`, and the resulting file is gradually getting built during the course of the clinical trial. The `Eligibility criteria definition` process was also executed as a consequence of the beginning of the planning phase. A formal description of the `Eligibility criteria` (as defined by the Clinical Research information Model, D6.2) is generated and taken as input, together with a `Set of databases` identified as useful for the study, for a query execution process. The goal of this query execution is to recover from the different study sites (`Query execution at study site 1` and `2`) the number of patients matching the criteria, in order to aggregate (`Overall results process`) the total number of patients (`Total of suitable patients`) and provide in the Trial Master File the justification of the study applicability. In the graph, the query executions at different study sites as well as the sets of suitable patients are highlighted with different background colours, to denote that these parts of the graph are stored at the respective study sites. After the eligibility criterion was defined, `Researcher Dr. Andrew` began to recruit his five patients. He explained to his patients the study to be conducted answered patient's questions, and some of the patients filled the informed consent. As a result of this process, `Set of informed consents` was obtained, and these signed documents are included in the Trial Master File. The artifact `Trial protocol document TP_EudraCT_23` for the study was then generated as the final

result of the Trial planning process. Note that the derivedFrom relations were added from the artifact to all previously generated documents and to the Trial Master File.

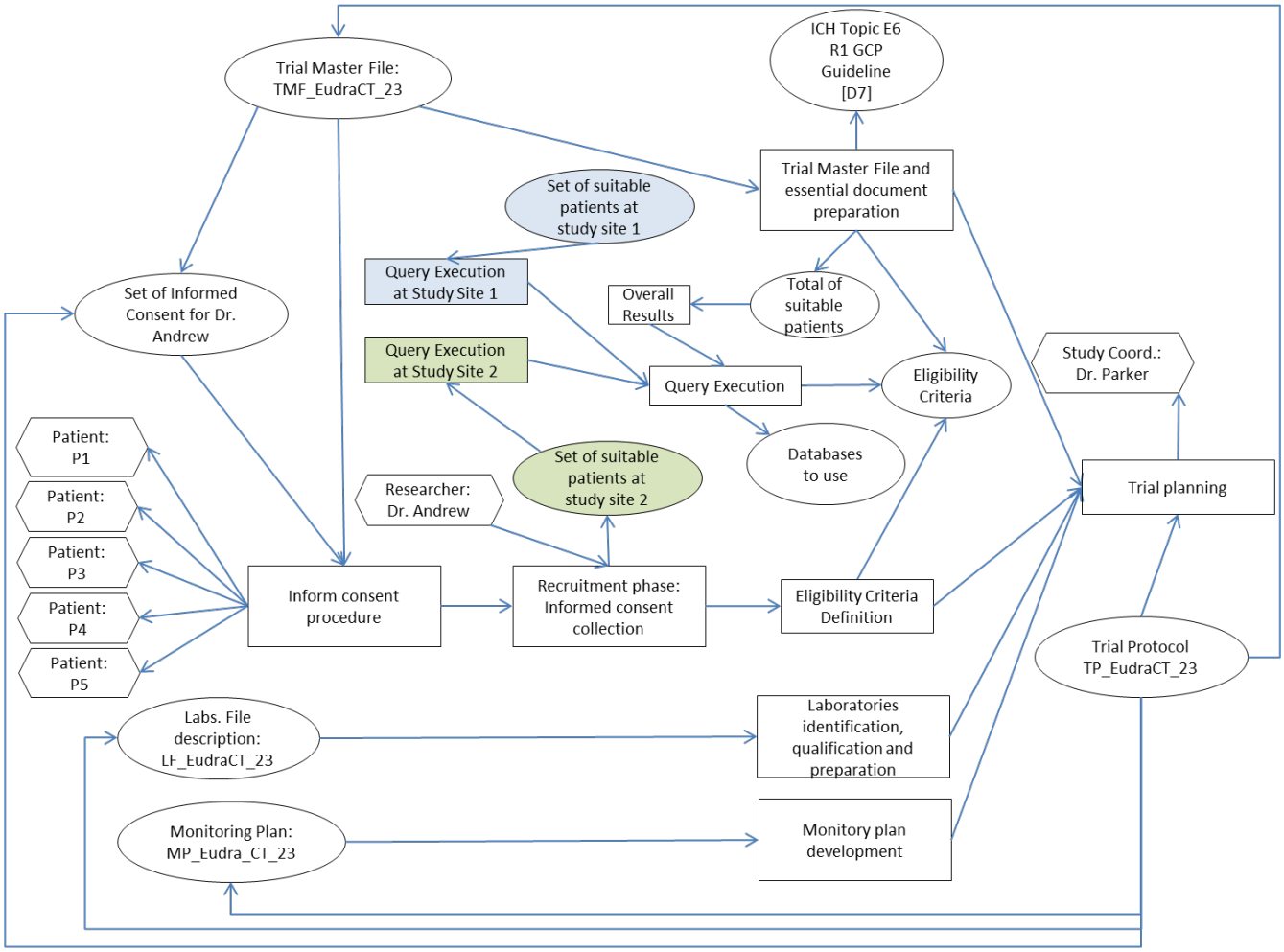

Figure 19: Trial planning phase graph

In Figure 20 a trace of the analysis phase of the RCT is shown, omitting tasks preceding it and other documents. The initialization step was performed by `Study coordinator Dr. Parker`, as represented by the `wasControlledBy` relation between the agent and the `Initialization of analysis data phase`. The next step, connected by `wasTrigerredBy` relation to the previous, was the `Statistical data analysis`, done by `Statistician Dr. Taylor`, and which resulted in a set of `Data analysis workflows DAW_CT_Eudra23` and the `data analysis report DAR_CT_Eudra23`, derived from the former. This report was then used to derive the `Final trial conclusions report TCR_CT_Eudra23` in the task `Finalized Trial`. Once this had completed, the trial is archived for future reference.

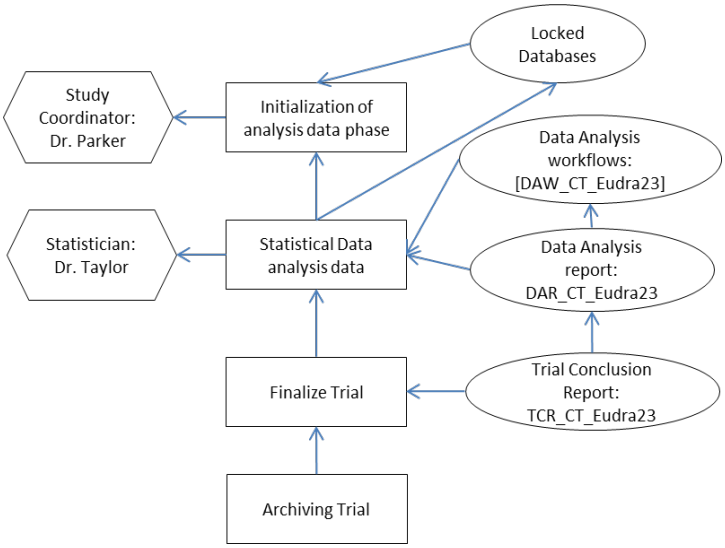

Figure 20: RCT Analysis phase template

Figure 21 displays a provenance graph associated with the conduction phase of a clinical trial. Specifically, it describes the changes in the eCRF of a patient from the version 0.34 until 0.37, during executing different types of interventions. The process `Update eCRF (1)`, that generated `eCRF0.35` from the data in `eCRF0.34`, was executed after an `Assessment of intervention`, that was executed over `Patient: Joan` and controlled by `GP: Dr. Smith`. The assessment phase detected an adverse effect which triggered an alarm, and consequently `Adverse effect procedure`, described in `Trial protocol file TP_EudraCT23` was executed. `Adverse effect report AdEff_2` was generated and used to `Divulgate the recognized adverse effect to the study sites` associated to the study. The study sites replied to this adverse effect report with a warning report (`set of warning reports`). A treatment for controlling the adverse effect was also prescribed.

The process `eCRF update(2)`, which generated `eCRF v0.36` from `eCRF v0.35`, was executed after obtaining the results of the `Image Analysis` derived from a `CAT intervention process`. The `CAT intervention process` used the guidelines provided in the `Trial protocol TP_EudraCT23`.

The process `eCRF update(3)`, which generated the `eCRF v0.37` from `eCRF v0.36`, was executed after a quality life questionnaire, as directed by the protocol, was applied by the `nurse Ms. Wood`, and filled by the `Patient: Joan` (`Questionnaire Quest_3_Joan`).

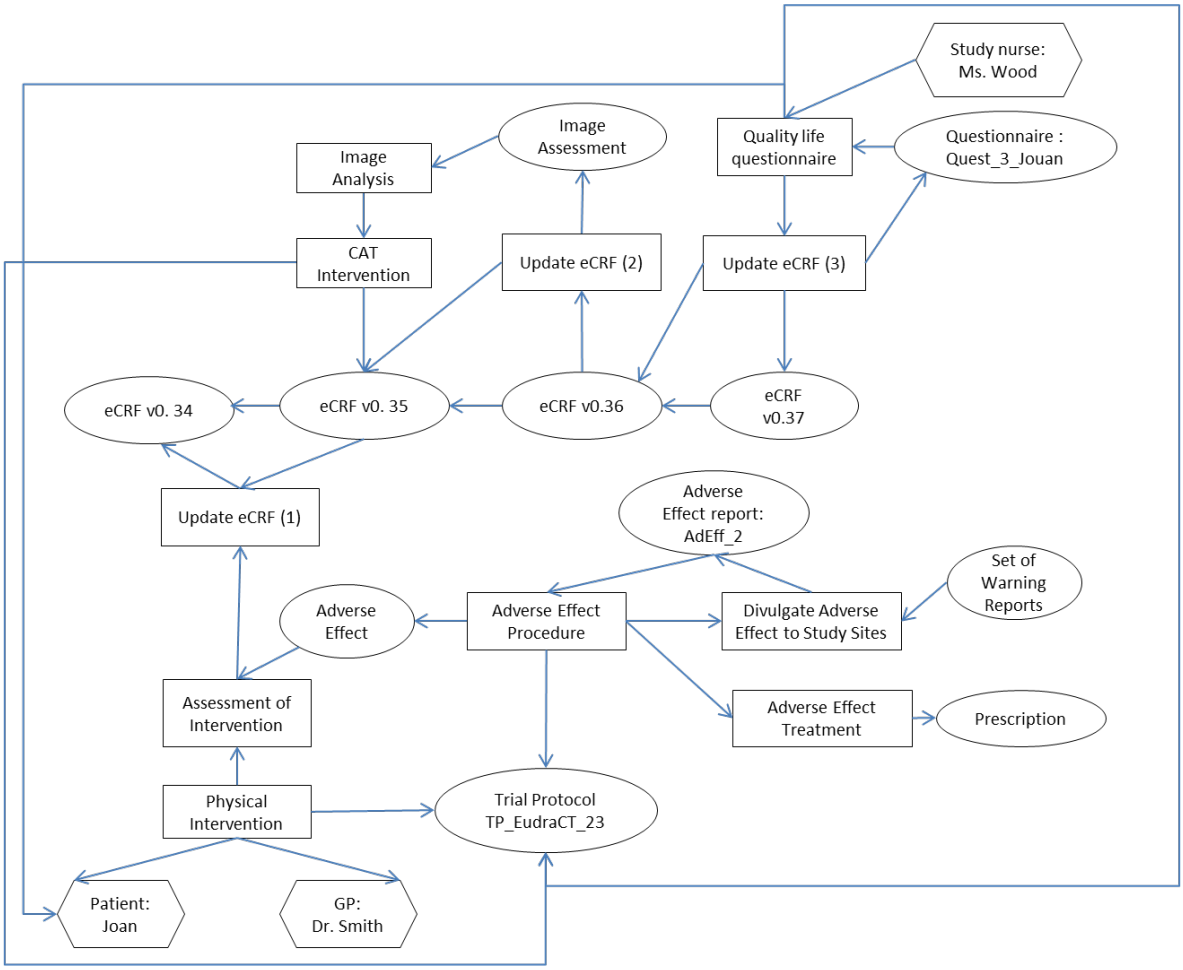

Figure 21: Clinical trial conduct phase graph

2.6 Decision support tool

The TRANSFoRm decision support tool (to be developed in WT4.4) aims to create a flexible system for supporting the formulation and quantification of differential diagnoses as part of the workflow of a primary care consultation, based on presenting patient diagnostic cues, and the patient reason for encounter. The formulation of diagnoses is done through the querying of an underlying clinical evidence service and rule base, and is based on patient diagnostic cues extracted from a primary care EHR system. The goal is to improve patient safety through a reduction in diagnostic error, benefiting patients, clinicians and the health care system.

The underlying clinical evidence is maintained as a set of clinical rules in the clinical evidence repository, which can be consulted by clinicians. Figure 22 shows a provenance template describing the process of obtaining a diagnosis recommendation for a patient during their visit to their primary care clinician. The first executed process `Collect clinical patient cues` (`P1`) is controlled by two software agents: a `Decision support interface (Ag1)` and `Primary care EHR system` (`Ag2`), and two human agents: `Primary care clinician` (`Ag3`) making the consultation and `Patient` (`Ag4`). This process produces `Diagnostic cue set` (`A1`) (signs, symptoms,

risk factors, clinical tests) as a combination of automatic extraction from the primary care EHR system, and manual update of cues as discussed by the patient with the clinician. Diagnostic cue set (A1) is then compared (process P2) with the evidences stored in a Clinical evidence repository (A2) in order to produce Diagnostic decision support recommendation (A3).

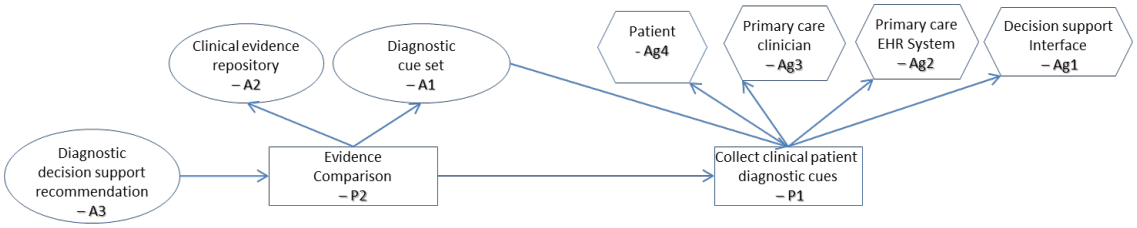

Figure 22: Provenance template for Decision support recommendation

The Clinical evidence repository can be updated by researchers adding (or updating) clinical rules in two different ways: manual updating with clinical evidences described in the biomedical literature (Figure 23) or using a data mining system to obtain such rules (Figure 24).

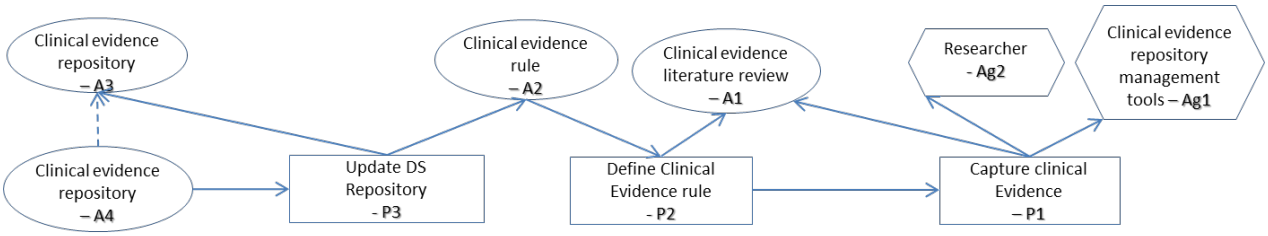

Figure 23: Provenance template for manual updating of Clinical evidence repository.

Figure 23 shows the Clinical evidence repository management tool (Ag1) executed by a researcher (Ag2) and used to execute a Capture clinical evidence option (P1), which allows the researcher to describe a set of cues retrieved from Clinical evidence literature review (A1) and trigger the Defining a Clinical evidence rule process (P2). As a result, a new Clinical evidence rule (A2) is manually defined and is used in Update decision support repository (A3 used by P3) and obtaining a new version of it (A4).

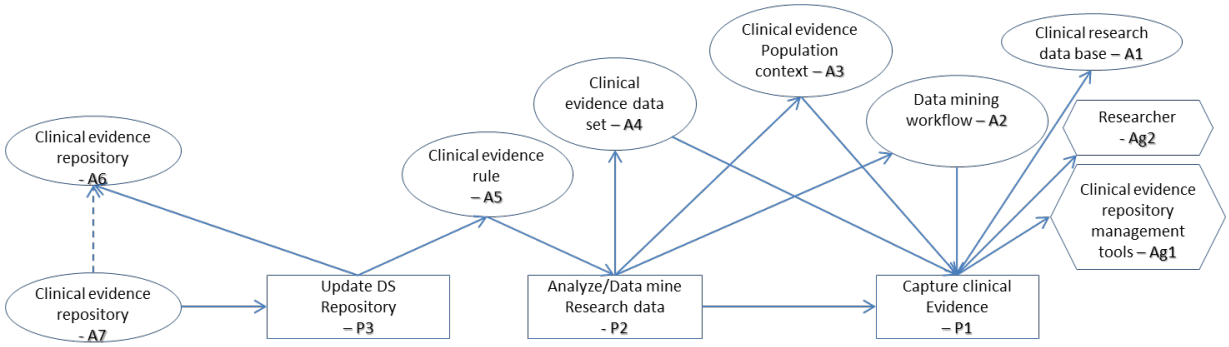

Figure 24: Provenance template for data mining process during Clinical Evidence repository updating.

In the case of the OPM template in Figure 24, the Clinical evidence rule (A5) is obtained from the process Analyse/data mine Research data (P2), which is triggered by the process Capture clinical Evidence (P1). This latter process, as in the previous example, is controlled by the Clinical evidence repository management tool (Ag1) and Researcher (Ag2), but it also uses a Clinical research database (A1) from which the data to mine are extracted, and produces three objects: a Data mining workflow (A2), that define how the data in the research database will be mine; the Clinical evidence population context (A3) that contains demographic description of the data to be mined; and Clinical evidence data set (A4) that contains the data to perform mining on. These three objects are used to mine the data and generate the new clinical evidence rule (A5), after which the process Update decision support repository (P3) proceeds as in Figure 23.

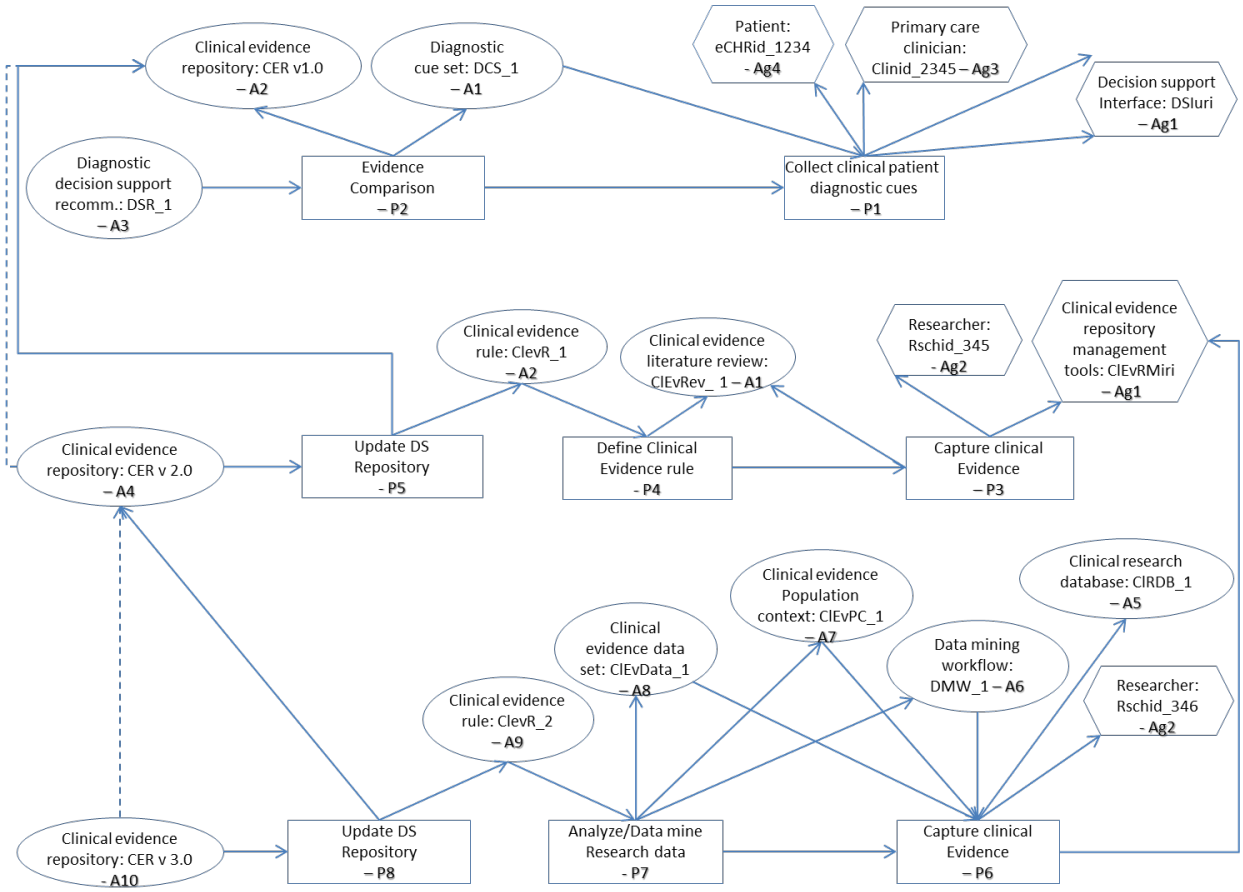

Figure 25: Provenance graph example using the templates for DSS.

The full graph in Figure 25 shows how the Clinical evidence repository CER v1.0 was used to obtain a diagnosis recommendation for Patient: eCHRid_1234 and how it evolved to version 3.0: first, Researcher: Rschid_345 defined a new Clinical evidence rule: ClevR_1 derived from a literature review, and then, Researcher: Rschid_346 executed a data mining process over the Clinical research database: ClRDB_1.

2.7 Summary

In this section, we introduced the concept of provenance templates, as means of defining the usage of provenance framework by TRANSFoRm software tools. The templates are an abstract process model that represents standard patterns of usage for the tools to produce valid provenance graphs with minimal exposure to the technical details underneath. This is done by the tools instantiating the entities in the graph templates and submitting these instantiated graph segments to the provenance framework.

The templates were developed based on the relevant models (CRIM and CEM), and further refined and simplified when needed with partners responsible for individual tools and use case representatives. The templates for the main TRANSFoRm tools were presented: Data Quality Tool, Query Formulation Tool, eCRF Workbench, and the Decision Support Tool. The remaining tools (Data Mining Tools in WT4.5, and Mobile services in WT7.6), which have not started development yet, are expected to define their own ontologies and templates using the ontology profile mechanisms included as part of our implementation.

Most of the data linkage and authentication functionality sits with the middleware (WT7.5), which has separate, low level, auditing mechanisms [1] and thus is not part of the provenance framework. However, it will be linkable to the provenance infrastructure via SAML assertions that are stored in the provenance graphs as the record of authentication.

Having shown how the client applications are making use of the provenance framework, we now proceed to show the details of the technical implementation in terms of the service interface used to access the framework (Section 3) and the underlying distributed storage mechanism (Section 4).

3. Server implementation (API/WSDL)

In this section we present the full information that a programmer would need to connect to and utilize the server functionality. Provenance server sits at the heart of the TRANSFoRm provenance framework. It publishes the programmatic interfaces for the client tools to use when capturing the relevant provenance information, and exposes functionality for querying the stored provenance data through SQL and SPARQL queries. The access to the server and to the stored data is managed using the Security Solution Layer (D3.3) and the access policies defined therein, using SAML assertions.

The API for the provenance tool exports the functionality for maintaining and querying provenance data, and has been designed for allowing an easy integration with other tools. It consists of the following packages:

- **eu.transformproject.provenance.service**, provides the classes necessary to configure and install the provenance service.
- **eu.transformproject.provenance.storage**, provides the classes necessary to store the opm graphs.
- **eu.transformproject.provenance.query**, provides the classes necessary to query the provenance data.
- **eu.transformproject.provenance.reasoning**, provides the classes necessary for reasoning about the provenance data.
- **eu.transformproject.provenance.representation**, provides the classes necessary for represent, open and save provenance graphs and provenance graph templates using different formats.
- **eu.transformproject.provenance.resources**, provides the classes necessary to describe the real physical resources represented in the OPM graphs.

The complete API documentation can be found online [10]. In the following subsections we present only the most important features of each package. The API is available either as Java API or WSDL web services.

3.1 Configuration (eu.transformproject.provenance.service package)

The provenance tool configuration consists of three basic steps:

1. A MySQL database management system (DBMS) for data storage needs to be created. Creation scripts are provided in Appendix B.
2. The semantic processing of the provenance data needs some file configurations that should be dynamically generated according to the requirements of the new provenance server being installed. Mappings from the relational database to the semantic framework, in D2RL language, are provided in Appendix D.
3. In a distributed environment, the definition of the local provenance stores, and the available resources should be specified.

Three server types are provided within the API: *Simple server, RMI server,* and the *WebService server*. When the provenance tool is installed, one of these server types is used (see Section 5). All three server types are extensions of **ProvenanceServer** class (Table 3)

which represents a default implementation for **ProvenanceServerInterface** (Table 2)**.** The simple server provides an object with the default implementation and the RMI and Web service servers contain also the functionalities of the corresponding Java server technology.

The XML Schema for defining provenance configuration files can be found in Appendix C. Figure 24 shows an example of a configuration file for defining a local provenance store in a non-distributed environment. The provenance store configuration requires a full (`name`) and abbreviated name (`shortName`), a database connection parameters (`DB_URL`, `DB_Name`, `user`, `password`), OWL URI base (`OWL_URI_BASE`) to be used for site entities created, the initial set of ontologies (`OPM_ontology`) that will be used to define the provenance data, the location where the provenance tool will be installed (`provenanceDir`), and the server (`server`) creation parameters.

`OWL_URI_BASE` is an URI prefix which will be used for constructing the URI of the entities in a provenance store. During the installation process the uniqueness of this `OWL_URI_BASE` is verified across the system. However, it is a responsibility of the installer to define useful `OWL_URI_BASE`, that is URIs prefix not previously used that can produce unique identifiers for their OPM entities. When an application executes a provenance code to store an entity, its URI is constructed concatenating this URI prefix with the value at the `OPMEntityKey` field in the `OPMEntity` table (the complete database schema is provided in Section 4). This approach not only allows an easy identification of the provenance store where an entity is stored, but provides for an elegant mapping definition for D2RQ tool. The latter allows translation from SPARQL statements using URIs into the SQL queries to be defined through a simple index recovery instead of using a complex SELECT statement (e.g. using LIKE-string operators).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<provenanceConfig>
        <local name="local zone 1" shortName="lz1">
        <desc>First provenance store at TRANSFoRm project.</desc>
        <DB_URL>jdbc:mysql://localhost:3306/</DB_URL>
        <DB_Name>BD_name</DB_Name>
        <user>user</user>
        <password>password</password>
        <OWL_URI_BASE>uri_base_for_the_new_data_provenance_entities</OWL_URI_BASE>
        <OPM_ontology>some_known_ontology_to_use</OPM_ontology>
        <provenanceDir>provenance_installation_diretory</provenanceDir>
        <server type="simple" name="simpleServerProvenance"/>
        </local>
</provenanceConfig>
```

Figure 26: Example provenance configuration file

Table 2 and Table 3 show, respectively, the interfaces and the classes defined in eu.transformproject.provenance.service package. The most important interface is **ProvenanceServerInterface** (an extension of the **ProvenanceStorageInterface** and **ProvenanceQueryInterface** interfaces), whose methods define the functionalities that can be used through a provenance server, and which are listed below. The first fourteen methods are used for creation of provenance graphs, and the value returned by these methods denotes the URI of the defined provenance entity. The last three methods are used for executing queries on the provenance data.

Table 1: Key methods of ProvenanceServer class

Method	Description
String **addAnnotation**(String uriEntityOrDependence, String owl_prop, String value)	Adds an annotation to an entity or a dependence, owl_prop is the property to annotate which must appear in the ontology profile (or set of domain-user ontologies) associated to the server, the value associated to the property is value.
String **newGraph**()	Adds a new graph in the provenance store.
void **setCurrentGraph**(String idGraph)	Specify that all the new entity definitions belong to the OPM graph with URI idGraph.
String **getOPMGraphForProject**(String projectName)	Create a new graph, and adds to it an annotation associating it with a project.
OPMProfile **getOPMProfileOntologyForApplication**(String uriAppl)	Create or recover an OPM profile ontology and associate it to the software tool URI
String **getEntityURI**(String owl_opm_Class, AnnotationSet annotations)	Generate or retrieve an URI associated to the given parameters. Given an owl_opm_class from the ontology profile (or set of domain-user ontologies) associated to the server, and a set of parameters that characterize the current entity, this method will retrieve a stored opm entity with such features, or create a new one. The URI of the generated/created instance will be returned.
AnnotationSet **composeAnnotation**(String owl_prop, String value)	Create an AnnotationSet object, initialising it with the annotation defined by the pair: owl_prop, value.
void **setNewVersion**(Annotable annotableObject)	Set a new identifier to the annotatable object, making clear that some changes to the object have been produced, so the current is a new version of the preceding data, and adds in the current provenance graph the causal dependence between the previous version of the object and the current one.
void **setProcessStatus**(String processStatus)	Allows to set the status of a process.
void **endProcess**(String processUri)	Allows to store the date-time in which a process has finalised.
void **endProcess**(String processUri, String processStatus)	Allows to store the date-time in which a process has finalised together with its status.
String **addCausalDependence**(String uriEntityCause, String opmo_dependenceClass, String uriEntityDest)	Defines a causal dependence from the entity uriEntityDest to uriEntityCause in the graph idGraph.
String	As the previous method but the causal

addCausalDependence(String uriEntityCause, String uriEntityDest)	dependency created is one of five basic OPM ones (as shown in Figure 3).
void **associateToAccount**(String uriEntity, String uriAccount)	Specify an account for which an specific OPM entity can be viewed.
List **executeSPARQL**(String sparql)	Execute a sparql statement.
List **executeSQL**(String sql)	Execute a sql statement.
List<List> **execute**(String sql)	Using the following format to combine both queries: USING (SQL/SPARQL query) EXECUTE (SQL/SPARQL query) or USING (SQL/SPARQL query) UNION (SQL/SPARQL query) The results of the query in the USING clause are used as variable values for executing different instances of the query in the clause EXECUTE or to join their results then the clause UNION is used.

Table 2: Interfaces in eu.transformproject.provenance.service package

Interface Summary	
ProvenanceConfigInterface	Contains a set of constant field values which are used as tags for the provenance server configuration file.
ProvenanceServerInterface	Defines the set of methods through which the provenance data can be stored and queried.
ProvenanceStorageInterface	Provides methods for provenance data annotation: entity creation, causal dependence definition, and provenance annotation inclusion.
ProvenanceQueryInterface	Provides a set of methods for querying the provenance data.

Table 3: Classes in eu.transformproject.provenance.service package

Class Summary	
CentralProvenanceStore	Provides the functionality for a central provenance store server configuration.
LocalProvenanceStore	Provides the functionality for a local provenance store server configuration.
ProvenanceAdmin	This class provides the code necessary for configuring and initialising the provenance service.
ProvenanceConfigReader	This class provides methods for reading a provenance configuration file and generating a local or central provenance store.
ProvenanceInstaller	This class provides the necessary methods for installing the provenance servers available in this API.
ProvenanceSecurity	This class provides the necessary implementation for the security layer.

ProvenanceServer	This class is a default implementation of the ProvenanceServerInterface, providing all the functionality of a provenance server.
ProvenanceServerRemote	This class extends the ProvenanceServer class to use it as a Java Remote Method Invocation (RMI) object.
ProvenanceServerWebService	This class extends the ProvenanceServer class to use it as a web service.
ProvenanceStore	This class implements the necessary methods for a provenance store server configuration.

3.2 Provenance storage (eu.transformproject.provenance.storage package)

Provenance data storage is based on Java Hibernate technology [11] for mapping between Java objects and their representations in the underlying relational database. All Hibernate configurations are created during the installation process, and the **ProvenancePersistenceManager** object uses native SQL queries to perform database updates. Note that provenance API does not provide any user-accessible delete mechanisms for removing previously stored provenance data. Instead, the potential errors in provenance data annotation are handled by annotating the entities that are erroneous using the annotation property defined in the ontology *rctpo*, *rctpo:annotationValid*, with value *false and rctpo:annotationValidityDesc* with a string describing the annotation error.

This strategy is also applied if during an OPM graph creation, if an inconsistency is found in an entity or dependence type.

Table 4: Classes in eu.transformproject.provenance.storage

Class Summary	
ProvenancePersistenceManager	This class contains a set of methods for provenance data storage. It implements the **ProvenanceStorageInterface**

3.3 Provenance resources (eu.transformproject.provenance.resources)

This package defines an interface (Table 5) and a set of classes (Table 6) for describing the available resources in the provenance system. The `OPMAnnotable` interface defines the following methods:

1. *String* ***getprovenanceURI****(): Returns the uri of the opm entity.*
2. *String* ***view****(): returns a String representing the some information about the opm entity.*
3. *javax.swing.JPanel* ***visualview****(): returns a graphical view of the OPM entity.*

The client applications can use this interface (or the class DefaultOPMAnnotable) for providing domain objects (objects of client applications) mechanisms to a) recover their URIs and b) to construct views of their data, enabling tools such as the Provenance Browser (Section 6.5) to hyperlink to the real data behind the URIs in the provenance stores.

Table 5: Interfaces in eu.transformproject.provenance.resources

Interface Summary	
OPMAnnotable	This interface should be implemented by all classes whose data will be annotated with the provenance API.

Class Summary	
AccessibleResourceMetadata	This class provides a default implementation for describing any king of resources whose provenance has been capture through this API.
AccessibleResourcesMetadataDB	This class provide all the necessary methods for updating the accessible resources described on the opm graphs in tables in the provenance store.
DefaultOPMAnnotable	This class is a default implementation for the interface OPMAnnotable. View() and VisualView() shows the URI of the annotated element.

Table 6: Classes in eu.transformproject.provenance.resources

3.4 Provenance Querying (eu.transformproject.provenance.query package)

The two classes defined in this package are summarized in Table 7. The first class implements the 3 methods for executing provenance queries as described in section 3.1 (`executeSQL, executeSPARQL, execute`). In Section 6, the algorithm for query implementation in the distributed environment is explained in detail.

Table 7: Classes in eu.transformproject.provenance.query

Class Summary	
ProvenanceQuery	ProvenanceQuery provides a set of methods for querying the provenance data. It implements the **ProvenanceQueryInterface**
ProvenanceQueryBasic	This class provides a set of methods for dynamic construction of SQL queries related with the management of the provenance data, used directly for the **ProvenancePersistenceManager**.

3.5 Provenance Reasoning (eu.transformproject.provenance.reasoning package)

A set of classes for reasoning about provenance data is exposed in this package (Table 8). The **ProvenanceReasoning** class contains methods for performing semantic inference on provenance entities, and is used in the analytical queries such as the ones we describe in Section 6.

Table 8: Classes in eu.transformproject.provenance.reasoning

Class Summary	
Ontologies	This class contains useful methods for loading ontologies and recovery definitions (or their provenance) across multiple ontologies.
Ontology	This class contains useful methods for loading a ontology and the easy recovery of its definitions through URIs.
OPMProfile	This class contains a set of useful methods for creating on-line ontology profile. It is useful for those applications which would like some semantic annotation of the provenance data, but does not have a well-defined ontology profile.
ProvenanceReasoning	A class for reasoning about provenance data: consistency, subsumption, equivalency, membership and property validation.

3.6 Graph manipulation (eu.transformproject.provenance.representation package)

During creation, a provenance graph (or one part of it) is maintained in memory in an object of the class `OPMGraph`. All entities of the graph can be changed programmatically through this class, stored to a database through server, and saved to files using three different formats: XML, RDF, and OWL (in `ProvenanceExportFormat`). An `OPMGraph` can also be loaded from these types of files.

`OPMGraphTemplate` represents provenance templates, as used by the TRANSFoRm tools. Known OPM graphs can be modeled with templates and instantiated (in an `OPMGraph`) with the real annotation values when the processes are executed.

Table 9: Classes in eu.transformproject.provenance.representation

Class Summary	
OPMGraph	OPMGraph contains all members and methods for updating the provenance data of an opm graph.
OPMGraphTemplate	A ProvenanceTemplate object contains all the information for deploying opm graphs.

Table 10: Enumerations in eu.transformproject.provenance.representation

Enum Summary	
ProvenanceExportFormat	An enumerated type for describing provenance formats (XML, RDF, OWL) that this API is able to read and save.

3.7 Interaction with the TRANSFoRm security and middleware

The Security Solution Layer described in D3.3 [12], to be delivered with the TRANSFoRm middleware in MS5, and as part of D7.2, provides the authentication and authorization mechanisms for tools in TRANSFoRm. Provenance will not duplicate the logs of the authentication and authorization processes performed within the middleware, but rather focuses on storing the authorization identifiers that the applications are using to access resources. These identifiers are the SAML assertions (described in [12]), that are issued by the Security Solution Layer, after successful authentication, and are used to grant or deny access in each authorization request.

To that goal, the provenance system assumes that TRANSFoRm tools using the middleware authentication and authorization mechanisms will have available these assertion identifiers (`SAid`). Therefore, the tools using the provenance API, will use the method `addAnnotation` in the ProvenanceServer class in order to maintain the `SAid` associated with the authorization process each time that the certificate needs to be associated with the executed process. According to SAML 2.0[2], `urn:oasis:names:tc:SAML:2.0:assertion#ID` is the standard property for identifying SAML assertions, and therefore, `SAid` annotations have to be created with this property.

3.8 Summary

In this section, we provided the full information needed for a programmer to utilize the TRANSFoRm provenance framework. The implementation of the provenance server was presented, together with the application programmer's interfaces (APIs) that the TRANSFoRm tools use to access it, and the key rationales and decisions behind the implementation. The installation and configuration of the server has been described, the basic programming patterns, and the interaction with the security solution layer in TRANSFoRm.

The basic units of provenance information in the system are the provenance graph segments, which are persisted, edited, and queried via the Provenance Server. These graphs are internally stored in a distributed provenance database, which is the subject of the next section.

[2] http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

4. Database implementation & distribution

Having presented the implementation of the provenance server, we shall now look towards how the provenance data is stored and managed. TRANSFoRm provenance framework stores provenance data, expressed as graphs, in a distributed relational database, implemented in MySQL, but portable to Oracle, SQL Server and other standard database management solutions. This section details the database schema used and the distribution solution for managing and querying provenance information across multiple sites.

4.1 Database schema

The provenance database schema is displayed in Figure 10. It consists of 16 tables divided as follows:

1. OPM entity types tables:
    a. OPMEntity: describes the general characteristics for the OPM entities. It contains an identifier and a value, and is associated with an *OPM graph*. It is also annotated with a timestamp, and can be of any entity type (graph, artifact, process, agent, or an account).
    b. OPMGraph: maintains a record for each graph in OPMEntity table, including creation time and an optional description field.
    c. OPMArtifact: maintains a record for each entity in OPMEntity table of "artifact", containing a creation time field.
    d. OPMProcess: maintains a record for each entity in OPMEntity table of "process", containing the creation time, an optional description, the starting and ending date-time of a process execution, and its status (as defined in Table ProcessStatus).
    e. OPMAgent: maintains a record for each entity in OPMEntity table of "agent", containing a creation time field.
    f. OPMAccount: maintains a record for each entity in OPMEntity table of "account", containing a creation time and an optional description field.
    g. AccountAssociations: maintains the accounts to which the different OPM entities are associated.
2. Causal dependency descriptor tables:
    a. OPMDependence: maintains the causal dependence relationships between the entities in OPMEntity table, through the fields OPMDependenceCause and OPMDependenceEffect. It also contains the creation time and dependence type (as defined in Table DependenceType) fields.
    b. OPMExternalDependence: this table allows the distributed storage of the provenance data. Its structure is the same of the previous table, with two more fields: *causeExternal* and *externalDBIdentifier*. The former allows describing whether the identifier in the cause or effect of a dependence is external of the current database and the latter, which external database contains the identifier of the dependence.
3. Annotation descriptor tables:
    a. OPMAnnotation: this table maintains the annotations associated to the entities in OPMEntity table. It constitutes a "triple style" rdf table.

4. Auxiliary tables:
    a. OPMEntityType: this table is initialized with the script of DB creation with the 6 entity types: artifact, agent, process, account, graph, and annotation.
    b. ProcessStatus: this table is initialized with the script of DB creation with the following values: started, ended_OK, ended_error, paused.
    c. OPMDependenceType: this table is initialized with the script of DB creation with the 5 causal dependency types: used, was controlled by, was derived from, was generated by and was triggered by.
    d. Property: this table contains the names of the properties used to annotate the provenance entities.
    e. Ontology: this table describes the ontologies from which the annotation properties have been taken.

As it has been previously mentioned, during the installation process, the database is created and the entity and causal dependence types are initialized in the appropriate tables. However, both tables can be extended with new user types. Semantic relations of these new user types with the types defined in the OPM specifications can be modelled with OPM-profile ontologies.

The *unique identifier* constraint in the OPMEntity table and the trigger operation during updating OPMDependence table avoid the inclusion of inconsistencies in the entity types, and causal dependence definitions.

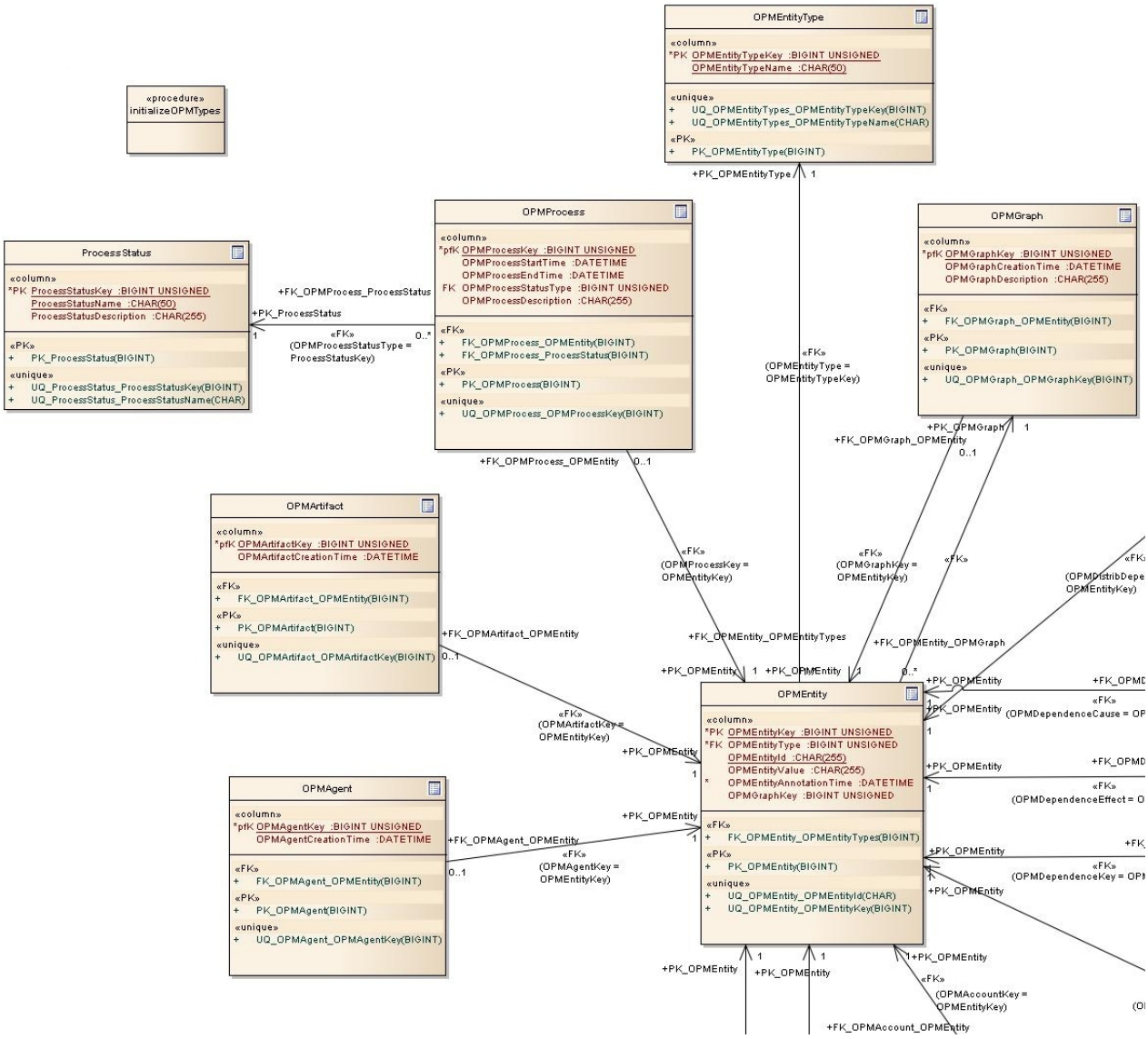

Figure 26: Database schema (Part I)

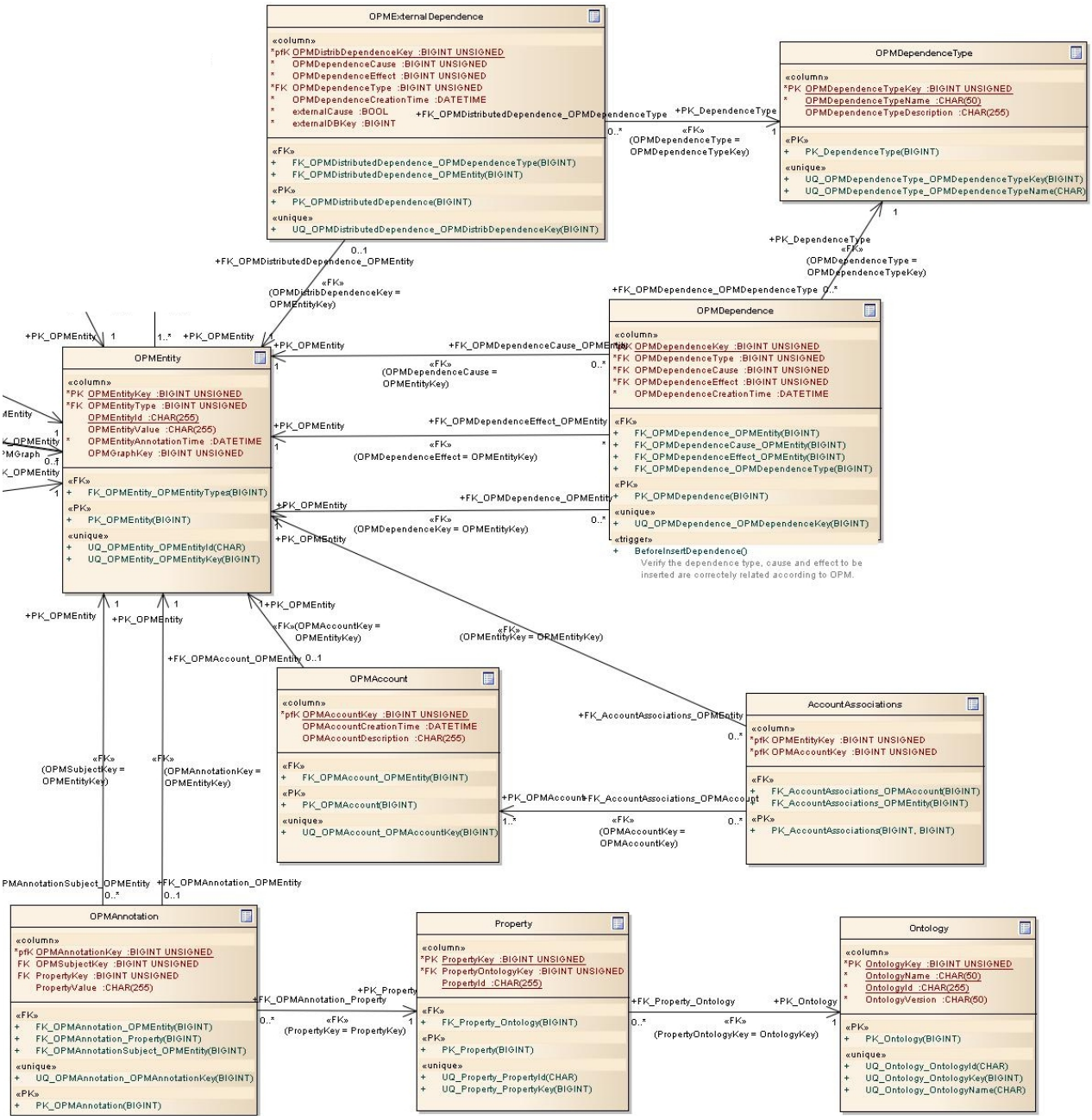

Figure 27: Database schema (Part II)

4.2 Distributed aspects of the provenance system

TRANSFoRm software tools reside at different study sites and by using the provenance server API, maintain the provenance graphs of the performed operations. In such a distributed environment the provenance query tool (described in section 6) may need to retrieve provenance data located at different study sites.

As was previously described, this is addressed by a distributed provenance system that divides the overall operating in Local and Centralized zones. Local zones are associated to study sites where operations subject of provenance annotation are performed. The Centralized zone controls operations amongst various study sites. The resources

(operations, most of them to access proprietary data) that can be accessed from other study sites, as well as the connection details to the different local provenance zones must be described in the central database.

4.2.1 Storing distributed provenance data

When an application at one study site requests the usage of resource located at another study site, the identity of the study site as well as a set of provenance entity identifiers are sent to the requested resource together with the access parameters. When the answer arrives, another set of identifiers is returned together with the required resource. These two sets of provenance entity identifiers contain keys in the `OPMEntity` table that allows tracking of provenance data at both sides of the call. On one hand, the entity effects of the call can be traced in the provenance local zone of the caller application. On the other hand, the entity causes (precedence) of the call can be traced in the provenance local zone of the called application. Note that the causal dependencies associating entities at different local zones are stored in the table `OPMExternalDependences`.

4.2.2 Querying SQL distributed provenance data

When provenance data is queried, the interpreter initially tries to resolve the query in the local zone, referring to the centralized zone if some of the records in the response are linked to data at other local zones.

For describing distributed queries, a simple query syntax is provided, as specified in the following grammar:

```
Query := select_statement ¦ Distributed_select
Distributed_select := FOR local_zones select_statement ¦
select_statement_with_prefix_db
local_zones := all ¦ local_zones_names
local_zone_names := short_local_zone_name {, short_local_zone_name}
```

Where a *select_statement* is a SELECT SQL statement and `short_local_zone_name` is a name given to the local zone in a configuration file. There are two ways to describe distributed queries. First, the user can define the set of local zones provenance stores to be queried or use "**all**" (*all qualifier name*) if the search should be done across the complete system (using the **FOR**). Alternatively, the users can define explicitly the provenance local stores to use in each part of the select sentence (*select_statement_with_prefix_db*), using the syntax: `<short_local_zone_name >.table`, instead of just `table`, in the *from* and *where* clauses of a select statement.

The algorithm for query resolution in the provenance system is described in Algorithm 1, which uses, in turn, the Algorithm 2 and 3. The Algorithm 2 allows to verify whether a query can be answered in the local zone, and the Algorithm 3 creates the federated solution for answered distributed queries in the centralized zone.

Algorithm 1. Distributed query resolution for provenance system

Input: `q`: query with syntax as defined in *Query Grammar*. `lzn`: name of the local zone where the query is formulated.

```
Output: ResultSet: answer of the query, q.

parsed_q = parsed details of q
if parsed_q.local_zones_to_use <> [lzn] then
        return executeCentralized(q) //Algorithm 3
else
        if can_be_locally_solved(q) //Algorithm 2
                return execute(q)
        else executeCentralized(q) //Algorithm 3
```

Algorithm 1 parses the input query to retrieve the set of local zones (or databases) explicitly mentioned by the user to be used. The local zone name where the formula is formulated is always returned as part of this set. If it contains other local zones different from the current local zone, or by using the Algorithm 2, it is determined that the query cannot be solved in the current local zone, then the query is send to the centralized zone and executed there. Otherwise, the query is solved in the current local zone.

Algorithm 2. Verifying whether a query can be solved in the local provenance zone

```
Input: q: query with syntax as defined in Query Grammar.
       lzn: name of the local zone where the query is formulated.
Output: solved_locally: true if the query can be solved locally.

parsed_q = parsed details of q
if not [<lzn>.OPMDependenceTable] in parsed_q.fromClauseTables then
        return false
else
        fromtables = parsed_q.fromClauseTables \ [<lzn>.OPMDependenceTable]
        fromtables = parsed_q.fromClauseTables ∪
                [<lzn>.OPMExternalDependenceTable, <lzn>.OPMEntity]
        condition = EliminateConditions(parsed_q.condition,
                [<lzn>.OPMDependenceTable])
        condition = condition ∪ createJoinConditions(condition, fromtables)
        sql = "SELECT 1 FROM" + fromtables + "WHERE " + condition + "LIMIT 1";
        resultSet = execute(sql);
        return resultSet.numrow == 0;
```

Algorithm 2, returns false if the OPMDependenceTable is not needed to answer the query as it is the only table containing dependency to tables in local zones different from the current one. Otherwise, the query q is transformed and the overall result depends on the number of records of the transformed query. If it is greater than 0, the answer of query needs to use external dependences, and therefore, it cannot be solved in the local provenance zone. The transformed query is obtained from the original one, by: a) replacing the OPMDependenceTable for OPMExternalDependence from the *from* clause of q and adding the table OPMEntity if it does not appears in the *from* clause, as the goal is to verify is exists any entity in the query that is used in OPMExternalDependence table; b) removing the conditions associated to the OPMDependenceTable, as it is not needed to verify here complex conditions related with OPMDependenceTable table; c) adding to the condition clause the join conditions between OPMEntity,

OPMExternalDependence and the remaining tables in the from clause; d) limiting the search to 1, as it is enough to found one entity with dependences on an external provenance store for deriving the query execution to the centralized zone.

Algorithm 3. Query resolution in the centralized zone.

Input: q: query with syntax as in *Query Grammar*.
local_zone_name: name of the local zone where the query is formulated.
Output: ResultSet: answer of the query, q.

parsed_q = parsed details of q
sql_sentences = GetConnectionSentences(parsed_q.local_zones_to_use)
sql_sentences = sql_sentences ∪
CreateTemporalFederatedTablesFor(parsed_q)
sql_sentences = sql_sentences ∪
CreateUnionViewsOfFederatedTablesFor(parsed_q)
distrib_q = replace in q the *tables* with the corresponding federated union view if the table
executeSentences(sql_sentences)
return execute(distr_q)

For executing a query in the centralized zone, Algorithm 3 uses the federated engine technology of MySQL[3]. The federated engine allows creation of *federated* tables whose data are associated with remote tables in the local provenance stores (connection details of which are known to the Centralized provenance store). Using these federated tables, Algorithm 3, create views to interpret the complete set of distributed data as only one database (using the local zone names as index in the views to interconnect them). Finally, the distributed query, distrib_q, is obtained replacing in q, each table with the corresponding federated union view of the corresponding table. Two optimizations have been incorporated in this process for reducing the number of tables and rows needed for creating the views, respectively.

4.2.3 Querying distributed provenance data with SPARQL

Semantic queries, formulated in SPARQL, are mapped to the relational database model using the Database-to-RDF-Query (D2RQ) tool. A mapping defined in D2RQL language allows this tool to create an RDF view of data stored in relational databases. This mapping includes the URI definition for the semantic instances to be included in the view. As explained in Section 3.1 OWL_URI_BASE in the provenance configuration files specifies a prefix for constructing the URIs of the instances of a specific provenance store. A concrete URI has the form: <OWL_URI_BASE>_<OPMEntityKey> (e.g. a pattern that concatenates the URI prefix of the local zone with the value at the OPMEntityKey field in the OPMEntity table). The provenance API verifies that the OWL_URI_BASEs of the provenance local zones included in the distributed system are all different. In this way, D2RQ and the provenance API can easily locate the database and the provenance entity that are referenced by a specific URI, and vice versa.

As for distributed SQL statements, a user can define the scope of their distributed SPARQL query using the following syntax:

[3] Similar features to which now exist in all major database systems.

```
Distributed_SPARQL statement  := FOR local_zones sparql_statement
```

Variable `local_zones` has the same meaning as in the previous section, and `sparql_statement` is a SPARQL statement.

Distributed SPARQL queries are sent to the centralized zone where the connections to all provenance stores are known and their corresponding D2RQL mappings have been generated by using the template mappings in Appendix D. These template mappings contain two variables: `#OWL_URI_BASE#` and `#shortName#`, which, during the mapping generation, get substituted with the value of the corresponding variable in the configuration file of the specific provenance store. The Appendix D is also divided in two parts: the first contains the mapping for local zones, and the second part defines the mappings that controls distributed dependences.

4.3 Summary

In this section, we described the database schema for the distributed provenance store, and presented the strategies for resolving provenance queries that could potentially span multiple sites. This distribution is largely hidden from the end-users, and resolved at the level of the framework.

Having presented the internal details of the implementation, in the next section, we show how the end-user tools are programmatically using the provenance framework to capture their data.

5. Provenance usage by tools

Having presented the conceptual solution to implementing provenance in software tools, and the inner workings of the TRANSFoRm provenance framework, this section will look into the technical details of provenance usage by the tools in TRANSFoRm. Concrete code examples, based on examples shown in Section 2 were devised with the technical partners, and show how the provenance API is used to capture provenance information. These examples are outlined in this section and fully presented in Appendix A. While all the examples given are using the Java API, Web Service WSDL invocations would follow the same pattern.

5.1 Levels of semantic annotation

The software tools in TRANSFoRm are of varying complexity, and their provenance models differ, from basic concepts that are easily satisfied with standard OPMO classes, to more complex semantic domains which require full ontologies. OPM profiles have been introduced in [4] as OPM graphs extensions for particular domains. As the provenance API described in this document is based on OWL definitions, our OPM profiles are ontologies extending OPMO and OPMV provenance ontologies for linking domain ontologies with provenance description (provenance templates) of their data and processes.

Depending on the annotation requirements, three different levels of semantic annotation can be specified:

- *minimal semantic annotation level* if no domain semantic resources are used,
- *medium semantic annotation level* if an OPM profile is defined using the Provenance API and used to create provenance annotations and
- *maximum semantic annotation level* if a dedicated OPM profile is provided for the annotations.

The provenance server defines a set of constants that allows use directly the provenance concepts: a) PROV_ID, used by the Provenance API to assign the URI to an entity; b) PROV_DESCR, used to add a literal description for an entity as part of his annotation; c) PROV_VALUE, used to add the provenance view of the data that to store in the provenance database; and d) URIs constants to the concepts in OPMO (e.g. PROCESS_URI).

Annotation using the minimal semantic annotation level means that the OPM entities are described with the classes in OPMO, and no additional domain concepts are associated to these entities. Therefore, such tools can use the constants in ProvenanceServer class to annotate their provenance data, `PROV_DESCR`, as for example in:

```
String uriNewMeasureProcess = provserver.getEntityURI(provserver.URI_PROCESS,
            provserver.composeAnnotation(provServer.PROVDESCR,
            "Creation of new quality measure"));
```

The data quality tool code example in Section 5.2 uses the minimal semantic annotation level.

The ideal level of semantic annotation is the maximal one: when an OPM profile ontology exists, and it is used to annotate the OPM entities. For the eCRF workbench application, an OPM profile ontology containing all concepts associated to clinical trials and their data flow, Randomized Clinical Trial Provenance Ontology (RCTPO), was constructed [13], and used to annotate the data. The following is an example of using this level of semantic annotation:

```
String uriClStudy = provserver.getEntityURI("rctpo:ClinicalStudy",
                        provserver.composeAnnotation(provServer.PROV_VALUE,
                                        clStudy.identifier()) );
```

The concept `rctpo:ClinicalStudy` is in the `rctpo` ontology. So, the provenance server can store the annotation directly using this domain OPM class. `clStudy` is a java object in the eCRF tool containing the data of a specific clinical study, and the method `identifier()` which returns the data about this clinical data to store in the provenance store. Notice that this is different from the URI identifier which is computed and returned by the provenance server, which the above example recover in the variable `uriClStudy`. The example in Section 5.3 uses the maximum semantic annotation level.

In some cases, the user needs to capture detailed semantic annotation, but the required profile may not exist. For such instances, Provenance API has defined the `OPMProfile` class, which allows users to create and use new OPM profile ontologies representing their provenance processes, artifacts, and agents. This is particularly useful for tools delivering in later stages of TRANSFoRm, such as the decision support system (WT4.4), data mining tools (WT4.3) and mobile applications (WT7.6), which will need to create or further refine their provenance profiles.

An `OPMProfile` object can be defined per application using the following code:

```
OPMProfile profileOnto = provServer.getProfileOntologyForApplication(
                      applicationName);
```

Now, the `OPMProfile` object (e.g. `profileOnto` in the above code) can be used to define domain's provenance concepts to be stored in the provenance database, and then retrieved for making the entities of the same concept consistent through all annotations. The defined profile can be also exported to an OWL ontology and extended/used in other tools of the domain. The following is an example using such type of annotation level:

```
String uriClinicalCuesCollectProcess =
provserver.getProcessURI(profileOnto.getProcessURI("CollectPatientClinicalCue
s"));
```

Notice that `profileOnto.getProcessURI("CollectPatientClinicalCues")` is used to define (or retrieve if it already exists) a class, named `CollectPatientClinicalCues`, derived from `opmo:Process`.

The example in Section 5.4 uses the medium semantic annotation level.

5.2 Data quality tool

Each time the Data Quality Tool is launched, the following needs to be provided: 1) the provenance server which will allows the data provenance annotation, 2) the URI that identifies the current tool, 3) the user executing the data quality tool. This is performed using the following code:

```
/* code to include when opening the application: recover the provenance and
 * security server, recover the user, recover the application URI
*/
String provZoneShortName = ...; // name of the provenance store
  //to maintain all data
ProvenanceServer provserver = ProvenanceServer.getServer(provZoneShortName);
String uriDataQualityTool = ...;
String uriUser = provserver.getAgentURI( securityServer.getUser() );
String uriDQTExecProcess = provserver.getEntityURI(provserver.PROCESS_URI,
      Provserver.composeAnnotation(provserver.PROV_DESCR, "DTQ Open"));
provserver.addCausalDependence(uriDataQualityTool, uriDQTExecProcess);
provserver.addCausalDependence(uriUser, uriDQTExecProcess);
```

The task of creation, or opening a previously created, Quality measure repository requires the following code to invoke provenance functionality:

```
/* code to include in the method executing the "Create new quality measure
 * repository" option in the quality measure tool interface.
*/
String nameOfRepository = ...; //name of the repository
DataQualityRepo repository = new DataQualityRepo(... nameOfRepository ...);
String uriOPMGraph = provserver.newGraph(); //The provenance server will
                                            //assign an addequate URI.
provserver.setAnnotation(uriOPMGraph, provserver.URIGraph_FOR_PROJECT,
                         nameOfRepository);
provserver.setCurrGraph(uriOPMgraph);
String uriRepo = provserver.getEntityURI(provserver. ARTIFACT_URI,
                        provserver.composeAnnotation(provServer.PROV_VALUE,
                                    repository.identifier()) );


/* code to include for the "Open quality measure repository" option
 * in the quality measure tool interface
*/
String nameOfRepository = ...; //name of the repository to open

String uriOPMGraph = provserver.getOPMGraphforProject(nameOfRepository);
provserver.setCurrGraph(uriOPMgraph);
DataQualityRepo repository = new DataQualityRepo(... nameOfRepository ...);
```

Detailed code examples based on graphs in Figure 14, Figure 15, and Figure 16, are given in Appendix A.

5.3 Query formulation tool/CRF workbench

When the workbench is started, the initialization provenance code is needed:

```
/* code to include when opening the application: recover the provenance,
 * middleware and security servers, the user, the application URI
*/
String provZoneShortName = ...; // name of the provenance store to maintain
all data
ProvenanceServer provserver = ProvenanceServer.getServer(provZoneShortName);
SecurityServer securityServer = ... //code to recover the security server
MiddlewareServer middlewareserver = ... //code to recover the middleware
server
String uriWorkbench = ...;
String uriUser = provserver.getAgentURI( securityServer.getUser() );
String uriApplExecProcess = provserver.getEntityURI("rctop:RCTTool");
provserver.addCausalDependence(uriWorkbench, uriApplExecProcess);
provserver.addCausalDependence(uriUser, uriApplExecProcess);
```

Depending on whether the user sets to create a new clinical study, or if they want to work with an existing study, one of the following two code segments is executed, and both are included in the appropriate menu actions in the workbench code.

```
/* code to include for the "Create new Clinical Study" option in the
 * workbench interface
*/
String nameOfClStudy = ...; //name of the new Clinical study
ClinicalStudy clStudy = new ClinicalStudy(... nameOfClStudy ...);
String uriOPMGraph = provserver.newGraph(); //The provenance server will
                                            //assign an addequate URI.
provserver.setAnnotation(uriOPMGraph, provserver.URIGraph_FOR_PROJECT,
                         nameOfClStudy);
provserver.setCurrGraph(uriOPMgraph);
String uriClStudy = provserver.getEntityURI("rctpo:ClinicalStudy",
                        provserver.composeAnnotation(provServer.ID,
                                    clStudy.identifier()) );
String uriClStudyCreationProcess = provserver.getEntityURI(
                                  "rctpo:RandomizedClinicalStudyCreation" );
Provserver.addCausalDependence(uriWorkbench, uriClStudyCreationProcess);


/* code to include for the "Open Clinical study" option
 * in the workbench interface"
*/
String nameOfClStudy = ...; //name of the Clinical Study to open
String uriOPMGraph = provserver.getOPMGraphforProject(nameOfClStudy);
provserver.setCurrGraph(uriOPMgraph);
ClinicalStudy clStudy = new ClinicalSuty(... nameOfClStudy ...);
String uriClStudyOpenProcess = provserver.getEntityURI(
                                    "rctpo:RandomizedClinicalStudyOpening" );
provserver.addCausalDependence(uriWorkbench, uriClStudyOpenProcess);
```

5.4 Decision support tool

Similarly to the previous tools, the following code segment is needed when initializing the application:

```
/* code to include when opening the application: recover the provenance
 * and security servers, the user, recover the application URI
*/
```

```
String provZoneShortName = ...; // name of the provenance store to maintain
all data
ProvenanceServer provserver = ProvenanceServer.getServer(provZoneShortName);
String uriDSS = ...;
String uriUser = provserver.getAgentURI( securityServer.getUser() );
String applicationName = "DSS";
ProfileOnto profileOnto = provServer.getProfileOntologyForApplication(
                      applicationName);
profileOnto.defineAsExtensible();
String uriApplExecProcess = provserver.getEntityURI(
      profileOnto.getProcessURI("DSSTool Opening")));
provserver.addCausalDependence(uriDSS, uriApplExecProcess);
provserver.addCausalDependence(uriUser, uriApplExecProcess);
```

Also, the creation, or opening a previously created Clinical Evidence repository requires the following code:

```
/* code to include in the method executing the "Create new clinical evidence
 * repository" option in the Clinical evidence management tool interface.
*/
String nameOfRepository = ...; //name of the repository
ClinicalEvidenceRepo repository = new ClinicalEvidenceRepo(...);
String uriOPMGraph = provserver.newGraph(); //the provenance server will
                       //assign an addequate URI.
provserver.setAnnotation(uriOPMGraph, provserver.URIGraph_FOR_PROJECT,
                      nameOfRepository);
provserver.setCurrGraph(uriOPMgraph);
String uriRepo = provserver.getArtifactURI(
                    profileOnto.getArtifactURI("ClinicalEvidenceRepository"),
                    provserver.composeAnnotation(provServer.PROV_VALUE,
                          repository.identifier()) );

/*code to include for the "Open clinical evidence repository" option
* in the decision support interface
*/
String nameOfRepository = ...; //name of the repository to open
ClinicalEvidenceRepo clEvidenceRepo = new ClinicalEvidenceRepo( ...
                                nameOfRepository ... );
String uriOPMGraph = provserver.getOPMGraphforProject(nameOfRepository);
provserver.setCurrGraph(uriOPMgraph);
```

Further detailed code examples of provenance usage by the decision support tool can be found in Appendix A.

5.5 Summary

Each TRANSFoRm tool uses a set of method invocations to interact with the provenance framework. This interaction can happen through any of the APIs provided, and it is left to the tool developers to decide which one fits their design best. In this section, we provided working examples of how such interaction would look like using pure Java API calls. The data submitted can be queried and analysed independently of which tool was used to create it, and the next section will show how that happens.

6. Provenance querying

In this section we describe the Provenance Query Tool for auditing and analysis of provenance data collected in TRANSFoRm. The tool consists of two parts: the *Provenance Browser* and *Query Collection*. The browser provides interactive graphical navigation through stored provenance graphs, with the user clicking on nodes to reveal further detail about the process and data histories captured. The query collection is a set of predefined, parameterized queries that capture the relevant information the users require to answer specific questions about the provenance data. In addition to these, the query collection can be easily extended by adding new parameterized queries by users with knowledge of basic provenance concepts to introduce new needed functionality as the use cases progress and gain deeper understanding of the questions they want to ask about their processes and data outputs.

The queries presented below represent the implementation of questions obtained from the representatives from the three TRANSFoRm use cases, e.g. what are the properties of a query used to recruit patients for a particular study, how was a particular process authenticated etc. The questions are organized by the tool, and for each an SQL or SPARQL implementation is given, as appropriate. SQL statements are processed by the relational database engine, while the SPARQL ones are making use of the Jena reasoner. The following URI prefix declarations are used in all SPARQL queries:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX opmo: <http://openprovenance.org/model/opmo>
```

6.1 Query formulation queries

1. *List all the initiating processes of all the queries that were completed on 01/04/2012.*

```
Input: None
Query:
SELECT OPMProcess.ProcessKey
FROM OPMProcess, OPMDependences
WHERE OPMProcess.OPMProcessEndTime BETWEEN '01/04/2012' AND '02/04/2012' AND
NOT OPMDependence.OPMDependenceEffect = OPMProcess.OPMProcessKey
```

2. *List all processes together with their authentication certificates.*

```
Input: None
Query:
select ?p, ?said
where{
      {?p rdf:type opmo:Process .
      ?p urn:oasis:names:tc:SAML:2.0:assertion#ID ?said .
}
union
select ?p, ?said
where{
      ?p rdf:type opmo:Process .
      ?p (opmo:cause/opmo:effectInverse)* ?entity .
      ?entity urn:oasis:names:tc:SAML:2.0:assertion#ID ?said .
```

```
}
```

3. *What query was used in each database to select cases and controls for a specific clinical study?*

```
Input: clinicalStudy
Query:
FOR all
select ?database, ?query
where{
      clinicalStudy.getProvURI() rctpo:hasEligibilityCriteria ?eligCriteria .
      ?p rdf:type rctpo:EligibilityCriteriaQueryExecution .
      ?p opmo:Used ?eligCriteria .
      ?p1 rdf:type rctpo:EHRDBQueryExecution .
      ?p1 opmo:wasTriggeredBy* ?p .
      ?p1 opmo:Used ?query .
      ?p1 opmo:Used ?database .
      ?query rdf:type rctpo:Query .
      ?database rdf:type rctpo:eHRDatabase .
}
```

4. *Which eligibility criteria were used in the query to recruit patients of a specific clinical trial study?*

```
Input: clinicalStudy
Query:
select ?eligCriteria
where{
      clinicalStudy.getProvURI() rctpo:hasEligibilityCriteria ?eligCriteria .
}
```

5. *Which databases were used to retrieve patient information of the cases of a specific clinical trial study ?*

```
Input: clinicalStudy
Query:
FOR all
select ?database
where{
      clinicalStudy.getProvURI() rctpo:hasEligibilityCriteria ?eligCriteria .
      ?p rdf:type rctpo:EligibilityCriteriaQueryExecution .
      ?p opmo:Used ?eligCriteria .
      ?p1 rdf:type rctpo:EHRDBQueryExecution .
      ?p1 opmo:wasTriggeredBy* ?p .
      ?p1 opmo:Used ?database .
      ?database rdf:type rctpo:eHRDatabase .
}
```

6. *How many sites contributed data for a specific clinical study?*

```
Input: clinicalStudy
Query:
FOR all
select (count(distinct ?studySite) as ?count)
where{
      clinicalStudy.getProvURI() rctpo:hasEligibilityCriteria ?eligCriteria .
      ?p rdf:type rctpo:EligibilityCriteriaQueryExecution .
```

```
      ?p opmo:Used ?eligCriteria .
      ?p1 rdf:type rctpo:EHRDBQueryExecution .
      ?p1 opmo:wasTriggeredBy* ?p .
      ?p1 opmo:Used ?database .
      ?database rdf:type rctpo:eHRDatabase .
      ?database rctpo:belongsTo ?studySite .
      ?studySite rdf:type rctpo:StudySite .
}
```

7. How many patients were recruited per site for a clinical study?

```
Input: clinicalStudy
Query:
select ?studySite, sum(?queryResult) as ?totalpatients
where{
      clinicalStudy.getProvURI() rctpo:hasEligibilityCriteria ?eligCriteria .
      ?p rdf:type rctpo:EligibilityCriteriaQueryExecution .
      ?p opmo:Used ?eligCriteria .
      ?p1 rdf:type rctpo:EHRDBQueryExecution .
      ?p1 opmo:wasTriggeredBy* ?p .
      ?p1 opmo:Used ?database .
      ?database rdf:type rctpo:eHRDatabase .
      ?database rctpo:belongsTo ?studySite .
      ?studySite rdf:type rctpo:StudySite .
      ?queryResult opmo:wasGeneratedBy ?p1 .
      ?queryResult rdf:type ?p1 .
} group by ?studySite, ?database
```

6.2 Data quality queries

1. How many changes have been introduced in the quality data measures repository since a specific date?

```
Input: repository, input_date
Assume: OPMVersions is a view containing the versioning dependences between
artifacts.
Query:
SELECT count(OPMVersions.causeEffect)
FROM OPMVersions, OPMEntity AS entitycause, OPMEntity AS entityeffect
WHERE OPMVersions.effectEntityKey == entityeffect.OPMEntityKey AND
      entityeffect.OPMEntityKey == repository.getprovenanceIRI() AND
      OPMVersions.causeEntityKey == entitycause.OPMEntityKey AND
      entitycause.OPMEntityAnnotationTime >= input_date
```

2. Which researcher did introduce the last change for a specific quality measure?

```
Input: dataQualityMeasure
Assume: OPMVersions is a view containing the versioning dependences between
artifacts.
        OPMDependencesStart is a view containing the transitive dependences.
It contains the field distance with the transitive distance between the cause
and effect.
Query:
SELECT entityagent.OPMEntityId
FROM OPMEntity, OPMEntity AS entityagent, OPMDependencesStart
WHERE OPMEntity.OPMEntityId == dataQualityMeasure.getprovenanceIRI() AND
      NOT (EXISTS (SELECT * FROM OPMVersions
```

```
      WHERE OPMVersions.causeEntityKey = OPMEntity.OPMEntityKey))
      AND OPMDependencesStart.OPMDependenceCause == entityagent.OPMEntityKey
      AND entityagent.OPMEntityType = 3
      AND OPMDpendencesStart.OPMDependenceEffect == OPMEntity.OPMEntityKey
```

3. What is the average time to evaluate a specified quality measure?

```
Input: dataQualityMeasure
Query:
SELECT average(OPMProcess.OPMProcessEndTime-OPMProcess.OPMProcessStartTime)
FROM OPMProcess, OPMEntity, OPMDependence, OPMAnnotation
WHERE OPMDependence.OPMDependenceEffect == OPMProcess.OPMProcessKey AND
      OPMAnnotation.OPMSubjectKey == OPMProcess.OPMProcessKey AND
      OPMAnnotation.propertyKey == provServer.PROV_DESCR AND
      OPMAnnotation.value LIKE 'Quality measure evaluation process%' AND
      OPMDependence.OPMDependenceCause == OPMEntity.OPMEntityKey AND
      OPMEntity.OPMEntityId == dataQualityMeasure.getprovenanceURI()
```

4. Which researcher did ask for recovering data quality measures associated with a database on a specific date?

```
Input: database, input_date
Query:
SELECT OPMAgent.OPMEntityKey
FROM OPMProcess, OPMAnnotation, OPMDependences, OPMAgent,
     OPMDependences as deps1, OPMEntity
WHERE DATEDIFF(OPMProcess.OPMProcessStartTime - input_date) < 1 AND
      OPMAnnotation.SubjectKey == OPMProcess.OPMProcessKey AND
      OPMAnnotation.propertyKey == provServer.PROD_DESCR AND
      OPMAnnotation.value like 'Quality measure recovery process%' AND
      OPMDependencesStart.OPMDependenceEffect == OPMProcess.OPMProcessKey AND
      OPMDependencesStart.OPMDependenceCause == OPMAgent.OPMAgentKey AND
      OPMDependencesStart.Distance < 4 AND
      deps1.OPMDependenceEffect = OPMProcess.OPMProcessKey AND
      deps1.OPMDependencesCause = OPMEntity.OPMEntityKey AND
      OPMEntiyId = database.getProvURI();
```

5. Retrieve any annotation associated to a specific data set.

```
Input: dataset
Query:
SELECT Property.PropertyId, OPMAnnotation.value
FROM OPMEntity, OPMAnnotation, Property
WHERE OPMEntity.OPMEntityId == dataset.getprovenanceURI() AND
       OPMEntity.OMEntityKey == OPMAnnotation.OPMSubjectKey AND
       OPMAnnotation.PropertyKey == Property.PropertyKey
```

6.3 Clinical trial study workbench queries

The following URI prefix definition are used in the SPARQL queries of this application:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rcto: <http://www.lesc.imperial.ac.uk/rcto/rcto>
PREFIX rctpo: <http://www.lesc.imperial.ac.uk/rcto/rctpo>
PREFIX obi: <http://purl.obolibrary.org/obo/obi>
```

The *Randomized Clinical Trial Ontology* (`rcto`) and the *Randomized Clinical Trial Profile Ontology* (`rctpo`) has been designed specifically for Clinical trial studies, in close collaboration with the WT6.4 partners and are based on the CRIM model (D6.2). The former describes the most important concepts associated with randomized clinical trial studies (Clinical Study, Eligibility Criteria, GP, Patient, Case Report Form, etc.), and imports the *Ontology for Biomedical Investigations* (`obi`) [14]. The latter defines an OPM profile for randomized clinical studies extending `rcto` to describe the provenance graph templates for randomized clinical studies.

1. *Find all the study nurses who have checked a screening marked as erroneous by the monitor at a specified study site.*

```
Input: studySite
Query:
select ?studyNurse
where {
      ?screenProc rdf:type rctpo:ScreenProcess .
      ?screenProc rctpo:hasStatus 'Error' .
      ?screenProc rctpo:wasExecutedAt studySite.getProvURI() .
      ?graph opmo:hasDependence ?dep .
      ?dep rdfs:subClassOf opmo:WasControlledBy .
      ?dep opmo:effect ?screenProc .
      ?dep opmo:cause ?studyNurse .
      ?studyNurse rdfs:subClassOf rcto:ResearchNurse .
}
```

2. *Given the patient identifier, list all versions of a specified eCRF, together with the date and the people involved in each modification.*

```
Input: ch (clinical history identifier of a patient)
Query:
select ?a, ?t, a1
where{
      ?a rdfs:subClassOf rcto:eCRF .
      ?a rcto:hasHCNumber ch .
      ?p opmo:endTime ?t .
      ?p rdf:type opmo:Process .
      ?dep rdfs:subClassOf opmo:WasGeneratedBy .
      ?dep opmo:effect ?a .
      ?dep opmo:cause ?p .
      ?p (opmo:effectInverse/opmo:cause)* ?a1 .
      ?a1 rdfs:subClassOf obi:Person .
   }
```

3. *What was the authorization used for the process-informed consents: conform privacy regulations used in a specific clinical study?*

```
Input: clinicalStudy
Query:
select ?legalDocument
where{
      ?p rdf:type rctpo:InformedConsent .
      ?p opmo:used* clinicalStudy.getProvURI() .
      ?p opmo:used* ?legalDocument .
```

```
      ?legalDocument rdf:type rcto:AuthorizationDocument .

}
```

4. *Which regulations are associated to a linkage process executed during a clinical study?*

```
Input: clinicalStudy
Query:
select ?legalDocument
where{
      ?p rdf:type rctpo:LinkageProcess .
      ?p opmo:used* clinicalStudy.getProvURI() .
      ?p opmo:used* ?legalDocument .
      ?legalDocument rdf:type rcto:LinkageRegulationDocument .
}
```

5. *The randomization tool used in the RCT has been changed and the patients whose randomization has been affected needs to be identified. Which patients were recruited using a particular randomization tool?*

```
Input: randomizationTool
Query:
select ?patient
where{
      ?p rdf:type rctpo:RecruitProcess .
      ?p opmo:wasTriggeredBy*/opmo:wasControlledBy
                                          randomizationTool.getProvURI() .
      ?patient opmo:wasGeneratedBy ?p .
}
```

6.4 Decision support queries

For this application the following profile definitions are used:

```
PREFIX cem: <http://www.rcsi.ie/cem/1.0>
PREFIX cemOPMProfile: <OWL_URI_BASE/cemp/>
```

Prefix `cem` denotes the Clinical Evidence Model ontology and `cemOPMProfile` the OPM profile created with the Provenance API. The latter refers to the `OWL_URI_BASE` variable defined in the configuration file of the local provenance store.

1. *What clinical evidence data set(s) was a decision support recommendation based on?*

```
Input: datasetRecommendation
Query:
select ?clEvidenceDataset
where{
      datasetRecommendation.getprovURI() opmo:wasGeneratedBy ?p .
      ?p rdf:type opmo:Process .
      ?p used* ?clEvidenceDataset .
      ?clEvidenceDataset rdf:type cemOPMProfile:ClinicalEvidenceDataSet .
}
```

2. *From what clinical evidence population context(s) was clinical evidence rule definition created from?*

```
Input: clEvidenceRule
Query:
select ?clEvidencePopulationContext
where{
      ?clEvidenceRule rdf:type cem:ClinicalEvidenceRule .
      ?clEvidenceRule opmo:wasGeneratedBy/opmo:wasTriggeredBy*/ ?p .
      ?p rdf:type cemOPMProfile:MineProcess ¦ cemOPMProfile:LitReviewProcess
.
      ?p opmo:Used ?clEvidencePopulationContext .
      ?clEvidencePopulationContext rdf:type
                                    cem:ClinicalEvidencePopulationContext .
}
```

3. *From what original research database(s) was a clinical evidence rule definition derived from?*

```
Input: clEvidenceRule
Query:
select ?researchDB
where{
      ?clEvidenceRule rdf:type cem:ClinicalEvidenceRule .
      ?clEvidenceRule opmo:wasGeneratedBy/opmo:wasTriggeredBy* ?p .
      ?p rdf:type cemOPMProfile:CaptureClinicalEvidence .
      ?p opmo:Used ?researchDB .
      ?researchDB rdf:type cemOPMProfile:ResearchDatabase .
}
```

4. *What researcher instigated a clinical evidence update that resulted in the generation of a clinical evidence rule?*

```
Input: clEvidence, clEvidenceRule
Query:
select ?researcher
where{
      ?clEvidence rdf:type cem:ClinicalEvidence .
      ?clEvidenceRule opmo:wasDerivedFrom* ?clEvidence .
      ?clEvidenceRule opmo:wasGeneratedBy/opmo:WasTriggeredBy* ?p .
      ?p rdf:type cemOPMProfile:CaptureClinicalEvidence .
      ?p opmo:wasControlledBy ?researcher .
}
```

5. *Has a clinical evidence rule been derived from a clinical evidence data set or from a manual literature review update?*

```
Input: clEvidenceRule
Query:
select ?artifact
where{
      ?clEvidenceRule rdf:type cem:ClinicalEvidenceRule .
      ?clEvidenceRule opmo:wasGeneratedBy/opmo:WasTriggeredBy* ?p .
      ?p rdf:type cemOPMProfile:createClEvidenceRule .
      ?p opmo:used ?artifact .
      ?artifact rdf:type cem:LiteratureReview .
} limit 1
```

The clinical evidence rule is derived from a literature review if the query returns a non-empty result.

6. *Provide a list of updated rules that were processed as part of a particular clinical evidence update process?*

```
Input: updateProcess
Query:
select ?updatedClEvidence
where{
      ?updatedClEvidence prov:laterGenerationThan ?clEvidence .
      ?clEvidenceRule rdf:type cem:ClinicalEvidenceRule .
      ?updatedClEvidence opmo:wasGeneratedBy ?updateProcess .
}
```

7. *What clinical evidence rules were matched to a particular patient diagnostic cue collection as part of a specific evidence comparison process that was run?*

```
Input: PatientCue
Query:
select ?clEvidenceRule
where{
      ?p opmo:Used ?patientCue .
      ?p rdf:type cemOPMProfile:EvidenceComparison .
      ?matchRule opmo:wasGeneratedBy cemOPMProfile:EvidenceComparison .

      ?matchRule rdf:type ?cemOPMProfile:MatchedRule .
      ?matchRule opmo:wasDerivedFrom ?patientCue .
      ?matchRule opmo:wasDerivedFrom ?clEvidenceRule .
      ?clEvidenceRule rdf:type cem:ClinicalEvidenceRule .
}
```

6.5 Provenance query security

As discussed in [1], rights to access the provenance of a data item can differ from the access rights to the data item itself, e.g. it is perfectly possible to investigate the analytical trace of a result without having insight into the individual records of the data sources used to generate it. Therefore, access to a provenance store may be different to application data due to user and system related policies. A data set may not be moved from its storage location, however subject to certain authentication and authorization policies, its provenance may be given accesses to be read and under some conditions to be replicated. In healthcare applications, security of a provenance store is typically more relaxed than that of patient data, since the subjects are not identifiable from the provenance data.

Within the TRANSFoRm provenance query tool, we verify the contents of each returned entity against the operational security policies specified in the Security Solution Layer (D3.3), and remove the segments which the user is not allowed access to.

6.6 Graphical interface

Once the user logs in to the provenance query tool, he is shown the screen in Figure 28. The queries displayed in the menus, and their definitions, are being loaded dynamically from a configuration file stored on the server, matching user's identifier and role. The queries are

shown in a panel on the left-hand side, and clicking on each displays their definition, together with any parameters that need to be provided.

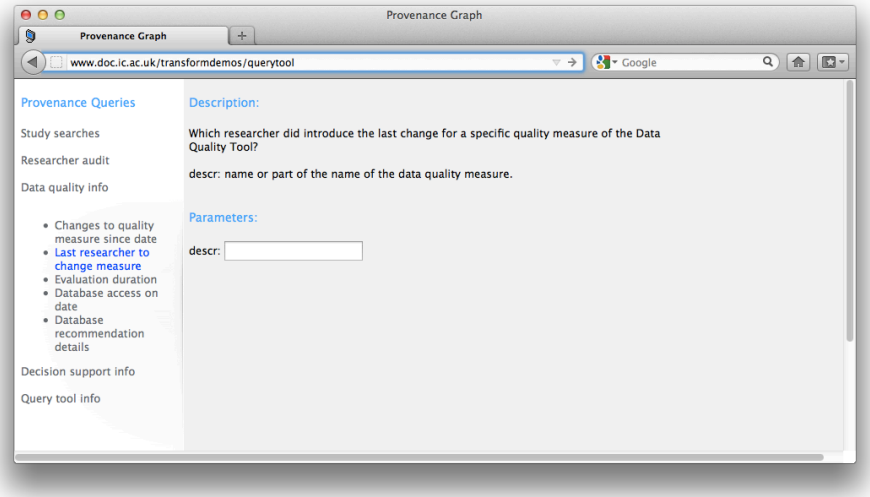

Figure 28: Provenance query tool

After running the query, the user is presented with the tabular results, in which each provenance graph entity is marked with a hyperlink. Clicking on the hyperlink launches the Provenance Browser.

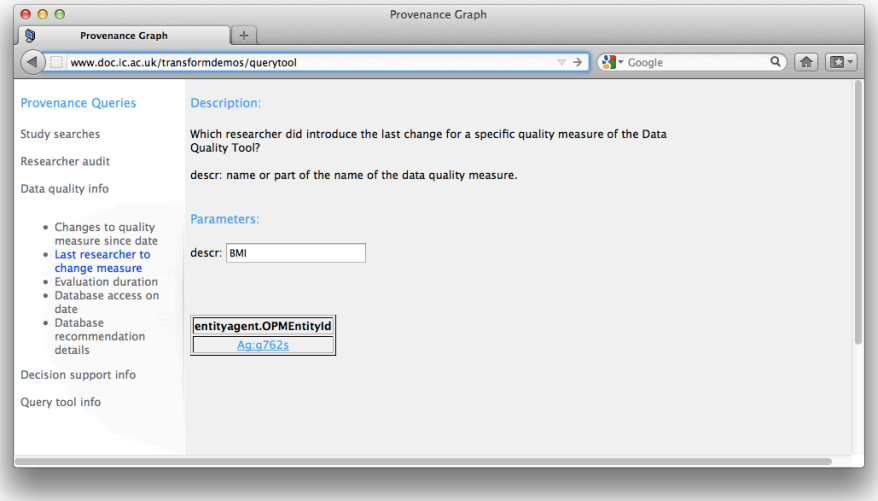

Figure 29: Query tool result

The Provenance Browser, shown in Figure 30, is a visual representation of the provenance space, which the user can traverse to find the origin details of all the processes and artifacts registered with the provenance framework. The active node is always displayed in the center, with the edges branching to other related nodes. Graph segments are dynamically loaded on demand in chunks, once they are requested. The panel on the right displays the connection details, together with any semantic annotations on that node. The annotations

differ based on the nature of the artifact, process, or agent, potentially including security token used in the process, URI of a data set, or parameters used in a query.

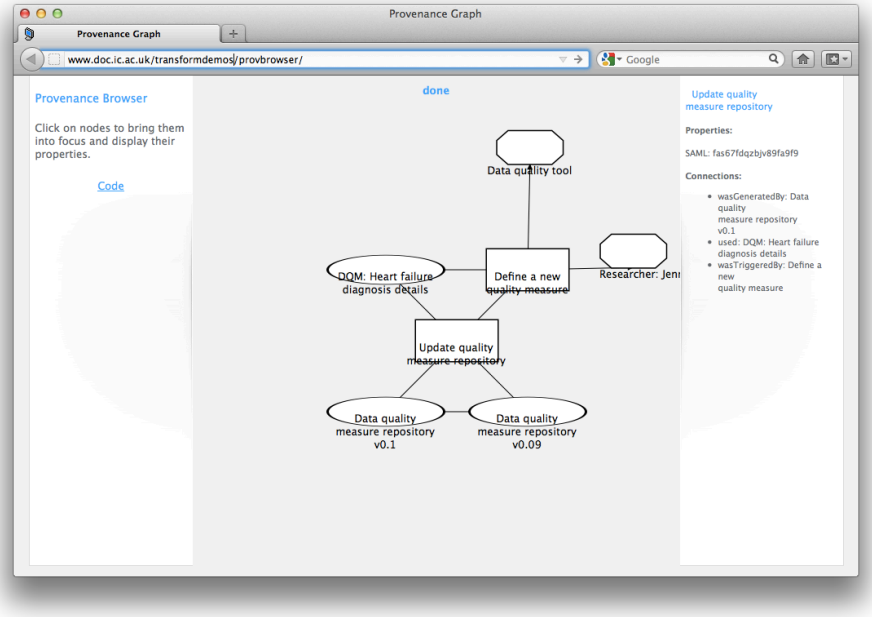

Figure 30: Provenance browser view

The graphical front-end is running as a web application within the same JBoss application server that the provenance server is residing in, and will use the same security policies as the rest of the system via the Security Solution Layer (D3.3). The visualisations were implemented using JavaScript, to achieve wide compatibility with different browser environments.

6.7 Summary

In this section we presented the query capabilities of the provenance framework, using example questions given to us by the partners from the Diabetes, GORD and Decision Support use cases. In addition to the basic set of queries that search for particular artifacts based on their semantic annotations, new parameterized queries are straightforward to add by any user with sufficient system privileges. The query capability can be accessed programmatically, or through a provided web tool, that also allows interactive browsing of result provenance graphs.

7. Conclusions

This document presented the implementation details of the TRANSFoRm provenance framework, together with detailed instructions for TRANSFoRm tools to utilize its functionality. The provenance API allows access to the distributed server and the relational database for storing the provenance data, and its subsequent querying for auditing and analysis. The actual use case examples, developed with project partners, have been used throughout the document to demonstrate the approach.

The novel method of provenance template design, combined with ontology-based OPM profiles, that we introduced in this deliverable, not only provides a consistent approach to implementing provenance support for TRANSFoRm tools, both present and future, but represents a generic approach for introducing provenance support in a wider class of software systems. Indeed, the approach is applicable to any set of disjointed, conceptually interacting tools that share common underlying vocabularies and ontologies.

A key aspect of this method is its model-based nature, whereby each aspect of the stored provenance data is specified in a model (in TRANSFoRm: CRIM, CDIM, or CEM). This not only ensures the conceptual correctness of the data stored, but facilitates the tool interaction with the API, in that there is a limited set of actions that need to interact with the provenance server, allowing us to explicitly define the code required.

Finally, this is also the first distributed provenance tool with dual SQL and SPARQL support, which opens up new possibilities for querying that transcend standard semantic queries that are common in existing provenance tools. This is further assisted by the inclusion of a novel visualisation assistant that provides both textual and graphical access to stored provenance data.

7.1 Further work

Looking beyond TRANSFoRm, the framework will be extended using a native RDF data store in order to perform a comparative evaluation of both storage management approaches. We are also looking at generalizing the model-driven approach developed here to a wider class of process-driven data-intensive settings in healthcare, with the view of developing an automated provenance template generation from BPMN workflows.

8. References

[1] Ashiq Anjum, Vasa Curcin, TRANSFoRm Provenance Framework, D3.1.

[2] Simon Miles, Paul Groth, Steve Munroe, and Luc Moreau. Prime: A methodology for developing provenance-aware applications. *ACM Transactions on Software Engineering and Methodology*, 20(3):1-42, August 2011.

[3] Wolfgang Kuchinke, Christian Ohmann, Clinical Research Information Model, D6.2.

[4] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems,* 27(6):743–756, June 2011.

[5] Satya S. Sahoo, Amit Sheth, Cory Henson, "Semantic Provenance for eScience: Managing the Deluge of Scientific Data," IEEE Internet Computing, pp. 46-54, July/August, 2008.

[6] Christian Bizer. D2RQ - treating non-RDF databases as virtual RDF graphs. In Proceedings of the 3rd International Semantic Web Conference. 2004.

[7] Hewlett-Packard Development Company, Jena Semantic Web Framework, 2009. http://jena.apache.org/.

[8] Jeen Broekstra and Arjohn Kampman and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. International Semantic Web Conference (ISWC). 2002. pp 54--68.

[9] SPARQL Protocol And Query Language, http://www.w3.org/standards/techs/sparql#w3c_all

[10] TRANSFoRm Provenance API, Javadoc documentation, http://www.doc.ic.ac.uk/~rdanger/provenance_doc/.

[11] Hibernate technology, http://www.hibernate.org/docs

[12] Stephen Farrell, TRANSFoRm Technical Security Framework, D3.1.

[13] Vasa Curcin, Roxana Danger, Wolfgang Kuchinke, Simon Miles, Adel Taweel, Christian Ohmann. Provenance model for randomized clinical trials, to appear in Data Provenance and Data Management for eScience, Springer, 2012.

[14] Mélanie Courtot, William Bug, Frank Gibson, Allyson L. Lister, James Malone, Daniel Schober, Ryan Brinkman and Alan Ruttenberg. The OWL of Biomedical Investigations. OWLED 2008.

Appendix A: Code examples of provenance usage in TRANSFoRm tools

A.1 Data Quality tool

In Figure 14, part of a provenance graph was shown, displaying the addition of a new quality measure in the quality measure repository. The code for generating corresponding provenance information is:

```
Example 1. Creating a new quality measure

/* code to include for the "Creating a new quality measure" option
* in the quality measure interface
*/
String uriNewMeasureProcess = provserver.getEntityURI(provserver.
PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR,
"Creation of new quality measure") );
provserver.addCausalDependence(uriUser, uriNewMeasureProcess);
provserver.addCausalDependence(uriDataQualityTool, uriNewMeasureProcess);

QualityMeasure m = ...;
String newMeasureDesc = ...; //the user could have included some
//description considering the Pubmed reference
            // "Rehabilitation (Stuttg). 2007 Jun;46(3):155-63."
String uriNewMeasureArtifact =
provserver.getEntityURI(provserver.ARTIFACT_URI,
                  provserver.composeAnnotation(provServer.PROV_VALUE,
m.identifier().
                  provServer.composeAnnotation("dcterms:description",
                              newMeasureDesc) );


String uriUpdateRepoProcess = provserver.getEntityURI(provserver.
PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR,
                  "updating repository") );
provserver.addCausalDependence(uriNewMeasureProcess, uriUpdateRepoProcess);
provserver.addCausalDependence(m.getprovenanceURI(), uriUpdateRepoProcess);
provserver.addCausalDependence(repository.getprovenanceURI(),
uriUpdateRepoProcess);
// ... here the code for updating the repository
provServer.setNewVersion(repository); // generate the new artifact
                        //representing the new version
                       // of the current repository,
                       //and the link between them
provserver.addCausalDependence(uriUpdateRepoProcess,
                                    repository.getprovenanceURI() );
```

The second example of the Data quality tool (Figure 15) detailed the updating of a previously defined quality measure. The provenance annotation code for such process is:

```
Example 2. Updating a new quality measure
```

```
/* code to include for the "Updating a new quality measure" option
 * in the quality measure tool interface
*/
QualityMeasure m = ...; // recovering the quality measure to update
String uriOriginalMeasureArtifact = m.getprovenanceURI();
String uriUpdateMeasureProcess = provserver.getEntityURI(provserver.
PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR,
              "updating the quality measure " + uriOriginalMeasureArtifact)
);
provserver.addCausalDependence(uriUser, uriUpdateMeasureProcess);
provserver.addCausalDependence(uriDataQualityTool, uriUpdateMeasureProcess);

m.update(...); // code for updating a measure
String uriUpdatedMeasureArtifact = provServer.setNewVersion(m);
provserver.addCausalDependence(uriUpdateMeasureProcess,
                               uriUpdatedMeasureArtifact);

String uriUpdateRepoProcess = provserver.getEntityURI(provserver.
PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR,
                "updating repository") );
provserver.addCausalDependence(uriUpdateMeasureProcess,
uriUpdateRepoProcess);
provserver.addCausalDependence(m.getprovenanceURI(), uriUpdateRepoProcess);
provserver.addCausalDependence(repository.getprovenanceURI(),
uriUpdateRepoProcess);
// ... here the code for updating the repository
provServer.setNewVersion(repository); // generate the new artefact
                               // representing the new version
                               // of the current repository, and the link
                               //between them
provserver.addCausalDependence(uriNewUpdateRepoProcess,
                               repository.getprovenanceURI() );
```

In the third example, the data quality tool is used to obtain the recommendation about what datasets to use for a specific clinical research (Figure 16, Figure 17, and Figure 18). The provenance annotation code is:

```
Example 3. Selecting appropriate datasets for a clinical study

/* code to include for the "use case selection" option
* in the quality measure interface
*/
String clinicalStudyUseCase = ...;
String uriUseCaseForCSProcess =
provserver.getEntityURI(provserver.PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR, "selecting
quality measures " +
                                                                    for
the clinical study + clinicalStudyUseCase) );
provserver.addCausalDependence(uriUser, uriUseCaseForCSProcess);
provserver.addCausalDependence(uriDataQualityTool, uriUseCaseForCSProcess);
//code to recover datasets available for the specific use case
DBset dbset = ...
provserver.addCausalDependence( uriUseCaseForCSProcess,
dbset.getprovenanceURI() );
```

```
// ... here code to open a quality measure repository
repository = ...

//selecting the suitable DB level quality measures
String uriSelectDBQMProcess = provserver.getEntityURI(provserver.PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR, "selecting
the DB level quality measures ");
provserver.addCausalDependence(uriUseCaseForCSProcess, uriSelectDBQMProcess);
provserver.addCausalDependence(repository.getprovenanceURI(),
uriSelectDBQMProcess);
QualityMeasureParameterSet qualityMeasureSet =
selectQualityMeasureParameterSet();
provserver.addCausalDependence( uriSelectDBQMProcess,
qualityMeasureSet.getprovenanceURI() );

//selecting datasources
String uriSelectDSProcess = provserver.getEntityURI(provserver.PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR, "selecting
datasource ");
provserver.addCausalDependence(uriSelectDBQMProcess, uriSelectDSProcess);
dbset.select(QualityMeasureParameterSet); // selects a subset of the oginal
DBset that fulfills the requirements in QualityMeasureParameterSet
provserver.setNewVersion(dbset);
provserver.addCausalDependence(uriSelectDSProcess, dbset.getprovenanceURI());

//selecting the suitable Disease level quality measures
String uriSelectDisBQMProcess =
provserver.getEntityURI(provserver.PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR, "selecting
the Disease level quality measures ");
provserver.addCausalDependence(uriSelectDSProcess, uriSelectDisQMProcess);
provserver.addCausalDependence(repository.getprovenanceURI(),
uriSelectDBQMProcess);
QualityMeasureParameterSet qualityMeasureDisSet =
selectQualityMeasureParameterSet();
provserver.addCausalDependence( uriSelectDisQMProcess,
qualityMeasureDisSet.getprovenanceURI() );

//code for obtaining evaluation (Figure 16)
for (DB db:dbset){
    String uriObtainEvalProcess =
provserver.getEntityURI(provserver.PROCESS_URI,
            provserver.composeAnnotation(provServer.PROV_DESCR, "Obtain
quality measure results");
    provserver.addCausalDependence( uriSelectDisQMProcess,
uriObtainEvalProcess );
    EvaluationData eval_info = recoverEvaluation(db, qualityMeasureDisSet);
    if (eval_info = null)
      eval_info = requestEvaluation(db, qualityMeasureDisSet);
    provserver.addCausalDependence( uriObtainEvalProcess,
eval_info.getProvIdInterch().get(0) );

    // code to use information in eval_info.getEval() and eval_info.getInfo()
    ....

}
```

```
//Request evaluation (Figure 17)
EvaluationData requestEvaluation(DB db, QualityMeasureParameterSet qm){
      String uriRequestEvalProcess =
provserverEval.getEntityURI(provserver.PROCESS_URI,
                  provserverEval.composeAnnotation(provServer.PROV_DESCR,
"Request for evaluation Process");
      Request request = createRequest(db, qm);
        provserver.addCausalDependence( uriRequestEvalProcess,
request.getprovenanceURI() );
      ProvenanceIdInterchange idInterchanging = new
ProvenanceIdInterchange();
      EvaluationData eval_info = send_getResults(request); // this method
should use middleware API to send the request and return back the results
        return eval_info;
}

//code inside the study site data provider
EvaluationData executerequest(request){
      String uriEvalProcess =
provserverEval.getEntityURI(provserver.PROCESS_URI,
                  provserverEval.composeAnnotation(provServer.PROV_DESCR,
"Quality measure evaluation Process");
      String uriTool = ... // code to recorer the URI of the evaluation tool
in the provider
      String provZoneShortNameEvaluator = ...; // name of the provenance
store of the evaluator
      ProvenanceServer provserverEvaluator =
ProvenanceServer.getServer(provZoneShortNameEvaluator);
      provserverEvaluator.addCausalDependence( uriTool, uriRequestEvalProcess
);
      provserverEvaluator.addCausalDependence( request.getprovenanceURI(),
uriEvalProcess );
        EvaluationData eval_info = ... // code that evaluates the quality of
the database

      String provZoneShortNameEvalResults = ...; // name of the provenance
store to maintain the evaluation results
      ProvenanceServer provserverEval =
ProvenanceServer.getServer(provZoneShortNameEvalResults);
      String uriStoreResultsProcess =
provserverEval.getEntityURI(provserver.PROCESS_URI,
                  provserverEval.composeAnnotation(provServer.PROV_DESCR,
"Store results Process");
      provserverEval.addCausalDependence( provserverEval.getprovenanceURI(),
uriStoreResultsProcess );
      provserverEval.addCausalDependence( uriEvalProcess,
uriStoreResultsProcess );
      provserverEval.addCausalDependence(
provserverEval.getEntityURI(provserverEval.ARTIFACT_URI,
                        provserver.composeAnnotation(provServer.PROV_VALUE,
eval_info.getEval())), uriStoreResultsProcess );
      provserverEval.addCausalDependence(
provserverEval.getEntityURI(provserverEval.ARTIFACT_URI,
                        provserver.composeAnnotation(provServer.PROV_VALUE,
eval_info.getInfo())), uriStoreResultsProcess );

      // code to return the eval_info through the middleware
```

```
}

//Recover evaluation method (Figure 18)
EvaluationData recoverEvaluation(DB db, QualityMeasureParameterSet qm){
      String provZoneShortNameEvalResults = ...; // name of the provenance
store to maintain the evaluation results
      ProvenanceServer provserverEval =
ProvenanceServer.getServer(provZoneShortNameEvalResults);

      String query = "select ?qm, ?value_eval, ?otherInfo" +
              "where{ ?qm rdf:type qmpo:QualityMeasure ." +
              "       ?eval rdf:type qmpo:QualityMeasureEval ." +
                      ?eval qmpo:db " + db.UIR() + "." +
                      "?eval qmpo:value ?value_eval ." +
                      "?eval qmpo:otherInfo ?otherInfo ." +
              "}";

      String uriProvQueryExecProcess =
provserverEval.getEntityURI(provserver.PROCESS_URI,
                  provserverEval.composeAnnotation(provServer.PROV_DESCR,
"Provenance Query Execution");
      provserverEval.addCausalDependence(provserverEval.getprovenanceURI(),
uriProvQueryExecProcess);
      provserverEval.addCausalDependence(provserverEval.getEntityURI(provserv
erEval.ARTIFACT_URI,
                       provserver.composeAnnotation(provServer.PROV_VALUE,
query) ), uriProvQueryExecProcess);
      SPARQLResults results = provserverEval.executeSPARQL(query);
      EvaluationData eval_info = EvaluationData(results);
      provserverEval.endProcess(uriProvQueryExecProcess);
      provserverEval.addCausalDependence(eval_info.getEval(),
uriProvQueryExecProcess);
      provserverEval.addCausalDependence(eval_info.getInfo(),
uriProvQueryExecProcess);

      ProvenanceIdInterchange idInterchanging = new
ProvenanceIdInterchange();
      idInterchanging.add( uriProvQueryExecProcess );
      eval_info.add(idInterchanging);
      return eval_info;
}
```

A.2 Query formulation tool/eCRF workbench

The first example in section 2.5 describes an OPM graph fragment during the trial planning phase. The following is the provenance code needed:

```
/* code to include for the beginning of the Trial Planning phase
 *
*/
String uriTrialPlanningProcess = provserver.getEntityURI(
                                    "rctpo:TrialPlanningProcess" );
provserver.addCausalDependence(uriUser, uriTrialPlanningProcess);

/* code to include in the trial planning interface, for eligibility
 * criteria definition
*/
```

```
String uriEligCriteriaProcess = provserver.getEntityURI(
                                    "rctpo:EligCriteriaProcess" );
provserver.addCausalDependence(uriTrialPlanningProcess,
                                uriEligCriteriaProcess);
EligibilityCriteria eligCriteria = new EligibilityCriteria( ... ); //code for
                    //create the eligibility criteria object
provserver.addCausalDependence(uriEligCriteriaProcess,
                                eligCriteria.getprovenanceURI());

/*code to include in the trial planning interface, for query execution
Databases databasestoUse = new Databases( ... ); //code for
                    //selecting the set of databases to use
String uriQueryExecProcess = provserver.getEntityURI(
"rctpo:CriteriaQueryProcess" );
provserver.addCausalDependence(eligCriteria.getprovenanceURI(),
                                uriLinkageProcess);
provserver.addCausalDependence(databasestoUse.getprovenanceURI(),
                                uriLinkageProcess);

NumberOfSuitablePatients totalSuitablePatients = 0;
String uriOverallProcess = provserver.getEntityURI(
                                "rctpo:OverallResultsProcess" );
for (Database db : databasestoUse){
      String urilocalQueryExecutionProcess = provserver.getEntityURI(
                                "rctpo:QueryExecutionAtLocalAreaProcess" );
      provserver.addCausalDependence(eligCriteria.getprovenanceURI(),
                                 urilocalQueryExecutionProcess);
      provserver.addCausalDependence(db.getprovenanceURI(),
                                 urilocalQueryExecutionProcess);
      try{
            provserver.addCausalDependence(eligCriteria.getprovenanceURI(),
                                          uriLinkageProcess);
EligCriteriaLocalResult eligCriteriaLocalResult =
middlewareserver.executeQueryForEligCriteria(
                              db, eligCriteria,
                              new String[]{uriLinkageProcess});
            provserver.endprocess(urilocalQueryExecutionProcess);
            NumberOfSuitablePatients numberOfSuitablePatients =
                               eligCriteriaLocalResult.getPatientNumber();
            provserver.addCausalDependence(urilocalQueryExecutionProcess,
                               numberOfSuitablePatients.getprovenanceURI());
            totalSuitablePatients = totalSuitablePatients +
                               numberOfSuitablePatients.getInt();
            provserver.addCausalDependence(
                  numberOfSuitablePatients.getprovenanceURI(),
                  uriOverallProcess);
      }catch(Exception e){
            provserver.setPocessStatus(urilocalQueryExecutionProcess,
                      provserver.PROCESS_STATUS_ERROR);
      }
}
provserver.addCausalDependence(totalSuitablePatients.getprovenanceURI(),
            uriOverallProcess);

/*code to include in the trial planning interface, for recruitment phase,
* at a local study site
*/
String uriRecruimProcess = provserver.getEntityURI(
```

```
                              "rctpo:RecruimentProcess" );
provserver.addDependences(uriUser, uriRecruimProcess);
SetOfuitablePatients suitablePatients = ... //code to recover the set of
                    //suitable patients for the study associated
                    //with the current user
provserver.addDependences(suitablePatients, uriRecruimProcess);
String uriInformConsentProcess = provserver.getEntityURI(
                      "rctpo:InformConsentProcess" );
provserver.addDependences(uriRecruimProcess, uriInformConsentProcess);
InformedConsentSet infConsentSet = ...//code to create a set of Informed
                      //consent documents
for (Patient patient: suitablePatients){
      InformedConsent informConsent = ...//code to recover information of the
                  //informed consent for the patient
      provserver.addDependences(patient.getprovenanceURI(),
                                uriInformConsentProcess);
      infConsentSet.add(informConsent);
}
provserver.addDependences(infConsentSet.gerprovenanceURI(),
                          uriInformConsentProcess);

/* code to include in the trial planning interface, during trial Master file
 * and essential document preparation
*/
String uriTMF_EssencialDocsProcess = provserver.getEntityURI(
          "rctpo:TMF_EssencialDocsPreparationProcess" );
provserver.addDependences(uriTrialPlanningPocess,
                          uriTMF_EssencialDocsProcess);
String guidelinesDocURI = ...;
TrialMasterFile trialMasterFile = new TrailMasterFile(...); //code to
identify
          // the trial Master File, the user must check the use of the
          //eligibility criteria and the result of the linkage process
          //were used for the trial master file creation. Also, the procedure
          //guidelines followed to create
         //the TMF and the set of informed consent must be identified
provserver.addDependences(guidelinesDocURI,
                          trialMasterFile.getprovenanceURI());
provserver.addDependences(eligCriteria.getprovenanceURI(),
                          trialMasterFile.getprovenanceURI());
provserver.addDependences(totalSuitablePatients.getprovenanceURI(),
                          trialMasterFile.getprovenanceURI());
provserver.addDependences(infConsentSet.gerprovenanceURI(),
                          trialMasterFile.getprovenanceURI());
provserver.addDependences(uriTMF_EssencialDocsProcess,
                          trialMasterFile.getprovenanceURI());

/*code to include in the trial planning interface, during other document
description
String uriLabDocsProcess = provserver.getEntityURI(
                  "rctpo:LabDocsPreparationProcess" );
provserver.addDependences(uriTrialPlanningPocess, uriLabDocsProcess);
LabFileDescription labFileDescr = new LabFileDescription(...); //code to
            //identify the laboratories File description
provserver.addDependences(uriLabDocsProcess,
labFileDescr.getprovenanceURI());
String uriMonitoringPlanProcess = provserver.getEntityURI(
                      "rctpo:MonitoringPlanProcess" );
```

```
provserver.addDependences(uriTrialPlanningPocess, uriMonitoringPlanProcess);
MonitoringPlan monitorPlanDescr = new MonitoringPlan(...); //code to identify
                  //the monitoring plan document
provserver.addDependences(uriMonitoringPlanProcess,
                 monitorPlanDescr.getprovenanceURI());

/*code to include in the trial planning interface, during Trial protocol
definition
String uriTrialProtDefProcess = provserver.getEntityURI(
                     "rctpo:TrialProtocolDefinitionProcess" );
provserver.addDependences(uriTrialPlanningPocess, uriTrialProtDefProcess);
TrialProtocol trialProtocol = new TrialProtocol(...); //code to identify the
              //trial protocol document, the user
              //must check that for the protocol all the previous documents
              //have been used
provserver.addDependences(trialMasterFile.getprovenanceURI(),
                          trialProtocol.getprovenanceURI());
provserver.addDependences(labFileDescr.getprovenanceURI(),
                          trialProtocol.getprovenanceURI());
provserver.addDependences(monitorPlanDescr.getprovenanceURI(),
                          trialProtocol.getprovenanceURI());
provserver.addDependences(uriTrialPlanningPocess,
                          trialProtocol.getprovenanceURI());
```

The second example in section 2.3, Figure 20, describes an OPM graph fragment during the analysis phase. The following is the provenance code needed:

```
/* code to include in the analysis phase
*
*/
String uriAnalysisProcess = provserver.getEntityURI(
                                    "rctpo:AnalysisProcessInitialization" );
provserver.addDependences(uriUser, uriAnalysisProcess);
eCRFDatabases eCRFDatabases = ...// code to obtain copy of
         //the locked eCRF databases
provserver.addDependences(uriAnalysisProcess,
eCRFDatabases.getprovenanceURI());

/* code to include in the statistical analysis interface
 *
*/
String uriStatisticalAnalysisProcess = provserver.getEntityURI(
                                    "rctpo:StatisticalAnalysisProcess" );
provserver.addDependences(uriUser, uriStatisticalAnalysisProcess);
provserver.addDependences(uriAnalysisProcess, uriStatisticalAnalysisProcess);
StatisticalWorkflows statisticalworkflows = ...// code to describe the
                   //workflows used for the statistical analysis
provserver.addDependences(uriStatisticalAnalysisProcess,
statisticalworkflows.getprovenanceURI() );
StatisticalReport statisticalreport = ...// code to describe the statistical
                   //report, the user must check the set of statistical
                   //workflows developed for the analysis
provserver.addDependences(statisticalworkflows.getprovenanceURI(),
statisticalreport.getprovenanceURI() );
provserver.addDependences(uriStatisticalAnalysisProcess,
statisticalreport.getprovenanceURI() );

/* code to include in the final report creation
```

```
 *
*/
String uriFinalizeTrialProcess = provserver.getEntityURI(
                                        "rctpo:FinalizeProcess" );
provserver.addDependences(uriUser, uriFinalizeTrialProcess);
provserver.addDependences(uriStatisticalAnalysisProcess,
uriFinalizeTrialProcess);
TrialConclusionFinalReport trialConclFinalReport = ... // code describe the
                    //final report, the user must check
                    //the statistical report was used for the final report
provserver.addDependences(statisticalreport.getprovenanceURI(),
uriFinalizeTrialProcess );
provserver.addDependences(statisticalreport.gerprovenanceURI(),
trialConclFinalReport.getprovenanceURI() );
provserver.addDependences(uriFinalizeTrialProcess,
trialConclFinalReport.getprovenanceURI() );

/* code to include in the archiving process
 *
*/
String uriArchiveTrialProcess = provserver.getEntityURI(
                                        "rctpo:ArchiveTrialProcess" );
provserver.addDependences(uriUser, uriArchiveTrialProcess);
provserver.addDependences(uriFinalizeTrialProcess, uriFinalizeTrialProcess);
```

A.3 Decision Support System

The provenance examples in section 2.6, gave two scenarios in which the provenance annotation is used: diagnostic recommendation and the capture of clinical evidences (either following a literature review or after a data mining process). The code to be included in each of these scenarios is:

```
Example 1. Diagnosis recommendation
/* code to include for the "Diagnostic Recommendation" option
 * in the decision support interface
*/
String uriEHRSystem = ...;
String patient = ...;
EHR patientEHR = ...; // code to obtain EHR record associated to a specific
patient (inside the EHR system)
String uripatient = provserver.getEntityURI(
                     profileOnto.getAgentURIFrom("cem:Patient"),
                     provenance.composeAnnotation(provServer.VALUE,
                           patientEHR.getIdentifier()) );
String uriClinicalCuesCollectProcess = provserver.getProcessURI(

profileOnto.getProcessURI("CollectPatientClinicalCues"));
provserver.addCausalDependence(uriUser, uriClinicalCuesCollectProcess );
provserver.addCausalDependence(uriDSS, uriClinicalCuesCollectProcess );
provserver.addCausalDependence(uriEHRSystem, uriClinicalCuesCollectProcess );
provserver.addCausalDependence(uripatient, uriClinicalCuesCollectProcess );
DiagnosticCueSet diagCueSet = ...; //collect the diagnostic cues
String uriEvidenceComparisonProcess = provserver.getEntityURI(
             profileOnto.getProcessURIFrom("EvidenceComparison") );
provserver.addCausalDependence(uriClinicalCuesCollectProcess,
                               uriEvidenceComparisonProcess );
provserver.addCausalDependence(diagCueSet.getprovenanceURI(),
```

```
                                uriEvidenceComparisonProcess );
provserver.addCausalDependence(clEvidenceRepo.getprovenanceURI(),
                                uriEvidenceComparisonProcess );
DiagnosticRecomendation diagnosis = ...; //code for evidence comparison
provserver.addCausalDependence(uriEvidenceComparisonProcess,
                                diagnosis.getprovenanceURI() );

Example 2. Capture Clinical evidence
/* code to include for "Capture Clinical evidence" option
 * in the Clinical evidence repository management tool
*/
String uriClinicalEvidenceRepositoryManagSystem = ...;
String uriCaptureClinicalEvidenceProcess = provserver.getEntityURI(
                 profileOnto.getProcessURIFrom("CaptureClinicalEvidence") );
provserver.addCausalDependence(uriUser, uriCaptureClinicalEvidenceProcess );
provserver.addCausalDependence(uriClinicalEvidenceRepositoryManagSystem,
                                uriCaptureClinicalEvidenceProcess );
if (CaptureManual)
    defineClinicalEvidence( ... );
else if (MineEvidences)
    ResearchDatabase researchDB = new ResearchDatabase( ... ); //code to
                //recover the DB from which data will be mined
      provserver.addCausalDependence(researchDB.getprovenanceURI(),
                                      uriCaptureClinicalEvidenceProcess );
      DataSet dataset = ...; //code to select the dataset to mine
      ContextPopulation contextPopulation = ...; //code to define the context
                               //population
      provserver.addCausalDependence(uriCaptureClinicalEvidenceProcess,
                                      dataset.getprovenanceURI() );
      provserver.addCausalDependence(uriCaptureClinicalEvidenceProcess,
                                      contextPopulation.getprovenanceURI() );
mineEvidences( ..., dataset, contextPopulation, ...  );

/*code to include for "Define clinical evidence rule" option
* in the Clinical evidence repository management tool
*/
void defineClinicalEvidence( ... ){
   String uriDefineClinicalEvidenceRuleProcess = provserver.getEntityURI(
              profileOnto.getProcessURIFrom("DefineClinicalEvidenceRule") );
   provserver.addCausalDependence(uriCaptureClinicalEvidenceProcess,
                                   uriDefineClinicalEvidenceRuleProcess );
   ClinicalLiteratureReview clLitReview = ...; // code to maintain data of
                     //the literature review
   provserver.addCausalDependence(clLitReview.getprovenanceURI(),
                                   uriDefineClinicalEvidenceRuleProcess );
   ClinicalEvidenceRule clEvidenceRule = ...; // code to construct the
                      // clinical evidence rule
   provserver.addCausalDependence(uriDefineClinicalEvidenceRuleProcess,
                                   clEvidenceRule.getprovenanceURI() );
   String uriUpdateClEvRepoProcess = provserver.getEntityURI(
         profileOnto.getProcessURIFrom("UpdateClinicalEvidenceRepository") );
   provserver.addCausalDependence(clEvidenceRule.getprovenanceURI(),
                                   uriUpdateClEvRepoProcess );
   provserver.addCausalDependence(clEvidenceRepo.getprovenanceURI(),
                                   uriUpdateClEvRepoProcess );
   //...here the code to update the repository
   provserver.setNewVersion(clEvidenceRepo);
   provserver.addCausalDependence(uriUpdateClEvRepoProcess,
```

```
                                    clEvidenceRepo.getprovenanceURI() );
}

/*code to include for "Mine clinical evidence rule" option
* in the Clinical evidence repository management tool
*/
void mineEvidences( ..., dataset, contextPopulation, ...  ){
   String uriMineEvidencesProcess = provserver.getEntityURI(
                profileOnto.getProcessURIFrom("MineClinicalEvidenceRule") );
   provserver.addCausalDependence(dataset.getprovenanceURI(),
                                  uriMineEvidencesProcess );
   provserver.addCausalDependence(contextPopulation.getprovenanceURI(),
                                  uriMineEvidencesProcess );
   MinedRules mineRules = mine(... dataset, contextpopulation, ...); //mine
//the data in the dataset
   for (ClinicalEvidenceRule clEvidenceRule : mineRules){
     ClinicalEvidenceRule clEvRulePrevVersion =
         clEvidenceRepo.getPrevVersion(clEvidenceRule); //try to recover a
                    //matching rule in the repository
                   //that can be considered as an old version of the mined
one
     if (clEvRulePrevVersion == null){
       String uriUpdateClEvRepoProcess = provserver.getEntityURI(
         profileOnto.getProcessURIFrom("UpdateClinicalEvidenceRepository") );
       provserver.addCausalDependence(clEvidenceRule.getprovenanceURI(),
                                      uriUpdateClEvRepoProcess );
 provserver.addCausalDependence(clEvidenceRepo.getprovenanceURI(),
                                     uriUpdateClEvRepoProcess );
       //...here the code to update the repository
       provserver.setNewVersion(clEvidenceRepo);
       provserver.addCausalDependence(uriUpdateClEvRepoProcess,
                                     clEvidenceRepo.getprovenanceURI() );
     }else{
        String uriUpdateClEvRepoProcess = provserver.getEntityURI(
        profileOnto.getProcessURIFrom("UpdateClinicalEvidenceRepository"));

provserver.addCausalDependence(clEvRulePrevVersion.getprovenanceURI(),
                                      uriUpdateClEvRepoProcess );
        provserver.addCausalDependence(clEvidenceRule.getprovenanceURI(),
                                      uriUpdateClEvRepoProcess );
        provserver.addCausalDependence(clEvidenceRepo.getprovenanceURI(),
                                      uriUpdateClEvRepoProcess );
        //...here the code to replace clEvRulePrevVersion by clEvidenceRule
in
        //the repository
        provserver.setNewVersion(clEvRulePrevVersion);
        provserver.setNewVersion(clEvidenceRepo);
        provserver.addCausalDependence(uriUpdateClEvRepoProcess,
                                      clEvidenceRepo.getprovenanceURI() );
    }
}
```

Appendix B: SQL script for database creation

#OPM_DB# is a database to be created, dynamically changed during installation.

```
START TRANSACTION;
DROP DATABASE IF EXISTS #OPM_DB#;
CREATE DATABASE #OPM_DB#;
USE #OPM_DB#;

CREATE TABLE OPMEntityType
(
      OPMEntityTypeKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      OPMEntityTypeName CHAR(50),
      PRIMARY KEY (OPMEntityTypeKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMGraph
(
      OPMGraphKey BIGINT UNSIGNED NOT NULL,
      OPMGraphCreationTime DATETIME,
      OPMGraphDescription CHAR(255),
      PRIMARY KEY (OPMGraphKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMEntity
(
      OPMEntityKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      OPMEntityType BIGINT UNSIGNED NOT NULL,
      OPMEntityId CHAR(255),
      OPMEntityValue CHAR(255),
      OPMEntityAnnotationTime DATETIME NOT NULL,
      OPMGraphKey BIGINT UNSIGNED,
      PRIMARY KEY (OPMEntityKey),
      KEY (OPMEntityType)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMArtifact
(
      OPMArtifactKey BIGINT UNSIGNED NOT NULL,
      OPMArtifactCreationTime DATETIME,
      PRIMARY KEY (OPMArtifactKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE ProcessStatus
(
      ProcessStatusKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      ProcessStatusName CHAR(50),
      ProcessStatusDescription CHAR(255),
      PRIMARY KEY (ProcessStatusKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMProcess
(
      OPMProcessKey BIGINT UNSIGNED NOT NULL,
      OPMProcessStartTime DATETIME,
      OPMProcessEndTime DATETIME,
      OPMProcessStatusType BIGINT UNSIGNED,
```

```
      OPMGraphDescription CHAR(255),
      PRIMARY KEY (OPMProcessKey),
      KEY (OPMProcessStatusType)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMAgent
(
      OPMAgentKey BIGINT UNSIGNED NOT NULL,
      OPMAgentCreationTime DATETIME,
      PRIMARY KEY (OPMAgentKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMAccount
(
      OPMAccountKey BIGINT UNSIGNED NOT NULL,
      OPMAccountCreationTime DATETIME,
      OPMAccountDescription CHAR(255),
      PRIMARY KEY (OPMAccountKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE AccountAssociations
(
      OPMEntityKey BIGINT UNSIGNED NOT NULL,
      OPMAccountKey BIGINT UNSIGNED NOT NULL,
      PRIMARY KEY (OPMEntityKey, OPMAccountKey),
      KEY (OPMAccountKey),
      KEY (OPMEntityKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMDependenceType
(
      OPMDependenceTypeKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      OPMDependenceTypeName CHAR(50) NOT NULL,
      OPMDependenceTypeDescription CHAR(255),
      PRIMARY KEY (OPMDependenceTypeKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMDependence
(
      OPMDependenceKey BIGINT UNSIGNED NOT NULL,
      OPMDependenceType BIGINT UNSIGNED NOT NULL,
      OPMDependenceCause BIGINT UNSIGNED NOT NULL,
      OPMDependenceEffect BIGINT UNSIGNED NOT NULL,
      OPMDependenceCreationTime DATETIME NOT NULL,
      PRIMARY KEY (OPMDependenceKey),
      KEY (OPMDependenceCause),
      KEY (OPMDependenceEffect),
      KEY (OPMDependenceType)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMExternalDependence
(
      OPMDistribDependenceKey BIGINT UNSIGNED NOT NULL,
      OPMDependenceCause BIGINT UNSIGNED NOT NULL,
      OPMDependenceEffect BIGINT UNSIGNED NOT NULL,
      OPMDependenceType BIGINT UNSIGNED NOT NULL,
      OPMDependenceCreationTime DATETIME NOT NULL,
      externalCause BOOL NOT NULL,
```

```
      externalDBKey BIGINT NOT NULL,
      PRIMARY KEY (OPMDistribDependenceKey),
      KEY (OPMDependenceType)
)  DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE Ontology
(
      OntologyKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      OntologyName CHAR(50) NOT NULL,
      OntologyId CHAR(255) NOT NULL,
      OntologyVersion CHAR(50) NOT NULL,
      PRIMARY KEY (OntologyKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE Property
(
      PropertyKey BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
      PropertyOntologyKey BIGINT UNSIGNED NOT NULL,
      PropertyId CHAR(255),
      PRIMARY KEY (PropertyKey),
      KEY (PropertyOntologyKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

CREATE TABLE OPMAnnotation
(
      OPMAnnotationKey BIGINT UNSIGNED NOT NULL,
      OPMSubjectKey BIGINT UNSIGNED,
      PropertyKey BIGINT UNSIGNED,
      PropertyValue CHAR(255),
      PRIMARY KEY (OPMAnnotationKey),
      KEY (PropertyKey),
      KEY (OPMSubjectKey)
) DEFAULT CHARSET latin1 ENGINE=InnoDB;

ALTER TABLE OPMGraph ADD CONSTRAINT FK_OPMGraph_OPMEntity
      FOREIGN KEY (OPMGraphKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMEntity ADD CONSTRAINT FK_OPMEntity_OPMEntityTypes
      FOREIGN KEY (OPMEntityType) REFERENCES OPMEntityType
(OPMEntityTypeKey);

ALTER TABLE OPMArtifact ADD CONSTRAINT FK_OPMArtifact_OPMEntity
      FOREIGN KEY (OPMArtifactKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMProcess ADD CONSTRAINT FK_OPMProcess_OPMEntity
      FOREIGN KEY (OPMProcessKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMProcess ADD CONSTRAINT FK_OPMProcess_ProcessStatus
      FOREIGN KEY (OPMProcessStatusType) REFERENCES ProcessStatus
(ProcessStatusKey);

ALTER TABLE OPMAgent ADD CONSTRAINT FK_OPMAgent_OPMEntity
      FOREIGN KEY (OPMAgentKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMAccount ADD CONSTRAINT FK_OPMAccount_OPMEntity
      FOREIGN KEY (OPMAccountKey) REFERENCES OPMEntity (OPMEntityKey);
```

```
ALTER TABLE AccountAssociations ADD CONSTRAINT
FK_AccountAssociations_OPMAccount
      FOREIGN KEY (OPMAccountKey) REFERENCES OPMAccount (OPMAccountKey);

ALTER TABLE AccountAssociations ADD CONSTRAINT
FK_AccountAssociations_OPMEntity
      FOREIGN KEY (OPMEntityKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMDependence ADD CONSTRAINT FK_OPMDependence_OPMEntity
      FOREIGN KEY (OPMDependenceKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMDependence ADD CONSTRAINT FK_OPMDependenceCause_OPMEntity
      FOREIGN KEY (OPMDependenceCause) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMDependence ADD CONSTRAINT FK_OPMDependenceEffect_OPMEntity
      FOREIGN KEY (OPMDependenceEffect) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMDependence ADD CONSTRAINT FK_OPMDependence_OPMDependenceType
      FOREIGN KEY (OPMDependenceType) REFERENCES OPMDependenceType
(OPMDependenceTypeKey);

ALTER TABLE OPMExternalDependence ADD CONSTRAINT
FK_OPMDistributedDependence_OPMDependenceType
      FOREIGN KEY (OPMDependenceType) REFERENCES OPMDependenceType
(OPMDependenceTypeKey);

ALTER TABLE OPMExternalDependence ADD CONSTRAINT
FK_OPMDistributedDependence_OPMEntity
      FOREIGN KEY (OPMDistribDependenceKey) REFERENCES OPMEntity
(OPMEntityKey);

ALTER TABLE Property ADD CONSTRAINT FK_Property_Ontology
      FOREIGN KEY (PropertyOntologyKey) REFERENCES Ontology (OntologyKey);

ALTER TABLE OPMAnnotation ADD CONSTRAINT FK_OPMAnnotation_OPMEntity
      FOREIGN KEY (OPMAnnotationKey) REFERENCES OPMEntity (OPMEntityKey);

ALTER TABLE OPMAnnotation ADD CONSTRAINT FK_OPMAnnotation_Property
      FOREIGN KEY (PropertyKey) REFERENCES Property (PropertyKey);

ALTER TABLE OPMAnnotation ADD CONSTRAINT FK_OPMAnnotationSubject_OPMEntity
      FOREIGN KEY (OPMSubjectKey) REFERENCES OPMEntity (OPMEntityKey);

DELIMITER //
CREATE PROCEDURE initializeOPMTypes ()
      BEGIN
            INSERT INTO OPMEntityType (OPMEntityTypeName) VALUES
                  ('Artifact'), ('Process'),('Agent'), ('Dependence'),
                  ('Account'), ('Annotation'), ('Graph');
            INSERT INTO OPMDependenceType (OPMDependenceTypeName) VALUES
                  ('Used'), ('WasGeneratedBy'),('WasControlledBy'),
                  ('WasTriggeredBy'), ('WasGeneratedFrom');
      END;

call initializeOPMTypes ();
COMMIT;
```

Appendix C: XML Schema for provenance store configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified"
           elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="provenanceConfig">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="central" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="centralzone" type="storeconfig" minOccurs="0"
                                        maxOccurs="1"/>
                            <xs:element name="localzone" type="storeconfig" minOccurs="1"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="local" type="storeconfig"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="storeconfig">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="shortName" type="xs:string" use="required"/>
        <xs:sequence>
                <xs:element name="desc" type="xs:string"  minOccurs="0" maxOccurs="1"/>
                <xs:element name="DBServer_URL" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="DB_Name" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="password" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="OWL_URI_BASE" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="OPM_ontology" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="provenanceDir" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="server" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:attribute name="type" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="simple"/>
                                    <xs:enumeration value="remote"/>
                                    <xs:enumeration value="webservice"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="url" type="xs:string"/>
                    </xs:complexType>
                </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Appendix D: D2RQL template mappings for SPARQL processing

#OWL_URI_BASE#: URI prefix for constructing the URI of the instances in a local zone.

#shortName#: short name of a local zone.

Part I. Local zones mapping

```
# ----------------------------------------------
# Prefixes

# D2RQ Namespace
@prefix d2rq:        <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .

# Namespace of the opm ontologies
@prefix opmo:  <http://openprovenance.org/model/opmo#> .
@prefix opmv:  <http://purl.org/net/opmv/ns#> .

# Namespace of our ontology
@prefix :      <http://openprovenance.org/dprovenance/example#> .

# Namespace of the mapping file; does not appear in mapped data
@prefix map#shortName#: #OWL_URI_BASE# .

# Other namespaces
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# ------------------------------------------------
# MAPPING TABLE OPMEntityType (OPMEntityTypeKey bigint(20) unsigned, OPMEntityTypeName char(50))
# ------------------------------------------------
map#shortName#:opm_entityType a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class :EntityType;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMEntityType.OPMEntityTypeKey@@";
    .

map#shortName#:opm_entityTypeName a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_entityType;
    d2rq:property :EntityTypeName;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMEntityType.OPMEntityTypeName@@";
    .

# ------------------------------------------------
# MAPPING TABLE OPMEntity (OPMEntityKey bigint(20) unsigned, OPMEntityType bigint(20) unsigned,
OPMEntityId char(255), OPMEntityValue char(255), OPMEntityAnnotationTime datetime, OPMGraphKey
bigint(20) unsigned)
# ------------------------------------------------
map#shortName#:opm_entity a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Entity;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMEntity.OPMEntityKey@@";
    .

# ------------------------------------------------
# MAPPING TABLE OPMGraph (OPMGraphKey bigint(20) unsigned, OPMGraphCreationTime datetime,
OPMGraphDescription char(255))
# ------------------------------------------------
map#shortName#:opm_graph a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:OPMGraph;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMGraph.OPMGraphKey@@";
        .

map#shortName#:opm_graphCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_graph;
    d2rq:property :creationTime;
```

```
    d2rq:column "OPMGraph.OPMGraphCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_graphDescription a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_graph;
    d2rq:property :description;
    d2rq:column "OPMGraph.OPMGraphDescription";
    d2rq:datatype xsd:string
    .

map#shortName#:opm_entity_annotation a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map#shortName#:opm_graph;
        d2rq:dynamicProperty "@@Property.PropertyId@@";
        d2rq:uriColumn "OPMAnnotation.PropertyValue";
        d2rq:join "OPMGraph.OPMGraphKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.OPMSubjectKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.PropertyKey => Property.PropertyKey";
        .

# ------------------------------------------------
# MAPPING TABLE OPMArtifact (OPMArtifactKey bigint(20) unsigned, OPMArtifactCreationTime
datetime)
# ------------------------------------------------
map#shortName#:opm_artifact a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmv:Artifact;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMArtifact.OPMArtifactKey@@";
    .

map#shortName#:opm_artifactId a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :id;
        d2rq:column "OPMEntity.OPMEntityid";
    d2rq:join "OPMArtifact.OPMArtifactKey => OPMEntity.OPMEntityKey"
    .

map#shortName#:opm_artifactValue a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :value;
    d2rq:column "OPMEntity.OPMEntityValue";
    d2rq:join "OPMArtifact.OPMArtifactKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_artifactAnnotTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :annotationTime;
    d2rq:column "OPMEntity.OPMEntityAnnotationTime";
    d2rq:join "OPMArtifact.OPMArtifactKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_artifactCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :creationTime;
    d2rq:column "OPMArtifact.OPMArtifactCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_artifactAnnotations a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :creationTime;
    d2rq:column "OPMArtifact.OPMArtifactCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_artifact_annotation a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map#shortName#:opm_artifact;
        d2rq:dynamicProperty "@@Property.PropertyId@@";
        d2rq:uriColumn "OPMAnnotation.PropertyValue";
        d2rq:join "OPMArtifact.OPMArtifactKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.OPMSubjectKey => OPMEntity.OPMEntityKey";
```

```
        d2rq:join "OPMAnnotation.PropertyKey => Property.PropertyKey";
        .

# ------------------------------------------------
# MAPPING TABLE ProcessStatus (ProcessStatusKey bigint(20) unsigned, ProcessStatusName char(50),
ProcessStatusDescription char(255) )
# ------------------------------------------------
map#shortName#:ProcessStatus a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class :ProcessStatus;
    d2rq:uriPattern "#OWL_URI_BASE#/ProcessStatus_@@ProcessStatus.ProcessStatusKey@@";
    .

map#shortName#:ProcessStatusName a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_entityType;
    d2rq:property :ProcessStatusName;
    d2rq:uriPattern "OWL_URI_BASE@@ProcessStatus.ProcessStatusName@@";
    .

map#shortName#:ProcessStatusDescription a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:ProcessStatus;
    d2rq:property :description;
    d2rq:column "ProcessStatus.ProcessStatusDescription";
    d2rq:datatype xsd:string;
        .

# ------------------------------------------------
# MAPPING TABLE Process (OPMProcessKey bigint(20) unsigned, OPMProcessStartTime datetime,
OPMProcessEndTime datetime, OPMProcessStatusType bigint(20) unsigned, OPMGraphDescription
char(255))
# ------------------------------------------------
map#shortName#:opm_process a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Process;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMProcess.OPMProcessKey@@";
    .

map#shortName#:opm_processId a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_process;
    d2rq:property :id;
    d2rq:column "OPMEntity.OPMEntityid";
    d2rq:join "OPMProcess.OPMProcessKey => OPMEntity.OPMEntityKey";
    .

map#shortName#:opm_processValue a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :value;
    d2rq:column "OPMEntity.OPMEntityValue";
    d2rq:join "OPMProcess.OPMProcessKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_processAnnotTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_artifact;
    d2rq:property :annotationTime;
    d2rq:column "OPMEntity.OPMEntityAnnotationTime";
    d2rq:join "OPMProcess.OPMProcessKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:ProcessStartTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_entityType;
    d2rq:property :processStartTime;
    d2rq:uriPattern "#OWL_URI_BASE#@@ProcessStatus.ProcessStatusName@@";
    .

map#shortName#:ProcessEndTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_entityType;
    d2rq:property :processEndTime;
    d2rq:uriPattern "#OWL_URI_BASE#@@ProcessStatus.ProcessStatusName@@";
    .

map#shortName#:ProcessProcessStatus a d2rq:PropertyBridge;
```

```
    d2rq:belongsToClassMap map#shortName#:ProcessStatus;
    d2rq:property :processStatus;
    d2rq:column "ProcessStatus.ProcessStatusName";
    d2rq:datatype xsd:string;
    d2rq:join "OPMProcess.OPMProcessStatusType => ProcessStatus.ProcessStatusKey";
        .

map#shortName#:opm_process_annotation a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map#shortName#:opm_process;
        d2rq:dynamicProperty "@@Property.PropertyId@@";
        d2rq:uriColumn "OPMAnnotation.PropertyValue";
        d2rq:join "OPMProcess.OPMProcessKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.OPMSubjectKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.PropertyKey => Property.PropertyKey";
        .

# ------------------------------------------------
# MAPPING TABLE OPMAgent (OPMAgentKey bigint(20) unsigned, OPMAgentCreationTime datetime)
# ------------------------------------------------
map#shortName#:opm_agent a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Agent;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMAgent.OPMAgentKey@@";
    .

map#shortName#:opm_agentId a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_agent;
    d2rq:property :id;
    d2rq:column "OPMEntity.OPMEntityid";
    d2rq:join "OPMAgent.OPMAgentKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_agentValue a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_agent;
    d2rq:property :value;
    d2rq:column "OPMEntity.OPMEntityValue";
    d2rq:join "OPMAgent.OPMAgentKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_agentAnnotTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_agent;
    d2rq:property :annotationTime;
    d2rq:column "OPMEntity.OPMEntityAnnotationTime";
    d2rq:join "OPMAgent.OPMAgentKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_agentCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_agent;
    d2rq:property :creationTime;
    d2rq:column "OPMAgent.OPMAgentCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_agent_annotation a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map#shortName#:opm_agent;
        d2rq:dynamicProperty "@@Property.PropertyId@@";
        d2rq:uriColumn "OPMAnnotation.PropertyValue";
        d2rq:join "OPMAgent.OPMAgentKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.OPMSubjectKey => OPMEntity.OPMEntityKey";
        d2rq:join "OPMAnnotation.PropertyKey => Property.PropertyKey";
        .

# ------------------------------------------------
# MAPPING TABLE OPMDependenceType (OPMDependenceTypeKey bigint(20) unsigned,
OPMDependenceTypeName char(50), OPMDependenceTypeDescription char(255) )
# ------------------------------------------------
map#shortName#:opm_dependenceType a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class :dependenceType;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependenceType.OPMDependenceTypeKey@@";
```

```
    .

map#shortName#:opm_dependenceTypeName a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceType;
    d2rq:property :dependenceTypeName;
    d2rq:column "OPMDependenceType.OPMDependenceTypeName";
    d2rq:datatype xsd:string
    .

map#shortName#:opm_dependenceTypeDescription a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceType;
    d2rq:property :description;
    d2rq:column "OPMDependenceType.OPMDependenceTypeDescription";
    d2rq:datatype xsd:string
        .

# ------------------------------------------------
# MAPPING TABLE OPMDependence (OPMDependenceKey bigint(20) unsigned, OPMDependenceType bigint(20)
unsigned, OPMDependenceCause, OPMDependenceEffect bigint(20) unsigned, OPMDependenceCreationTime
datetime)
# ------------------------------------------------
# ---------- used ----------
map#shortName#:opm_dependenceUsed a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Used;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependence.OPMDependenceKey@@";
    d2rq:condition "OPMDependence.OPMDependenceType = 1";
    .

map#shortName#:opm_dependenceUsedCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeUsed;
    d2rq:refersToClassMap map#shortName#:opm_artifact;
    d2rq:join "OPMDependence.OPMDependenceCause => OPMArtifact.OPMArtifactKey";
    .

map#shortName#:opm_dependenceEffectUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectUsed;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    .

# ---------- was generated by ----------
map#shortName#:opm_dependenceWasGeneratedBy a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:WasGeneratedBy;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependence.OPMDependenceKey@@";
    d2rq:condition "OPMDependence.OPMDependenceType = 2";
    .

map#shortName#:opm_dependenceUsedCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseWasGeneratedBy a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeWasGeneratedBy;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceCause => OPMArtifact.OPMArtifactKey";
    .

map#shortName#:opm_dependenceEffectWasGeneratedBy a d2rq:PropertyBridge;
```

```
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectWasGeneratedBy;
    d2rq:refersToClassMap map#shortName#:opm_artifact;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    .

# ---------- was controlled by ----------
map#shortName#:opm_dependenceWasControlledBy a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:WasControlledBy;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependence.OPMDependenceKey@@";
    d2rq:condition "OPMDependence.OPMDependenceType = 3";
    .

map#shortName#:opm_dependenceWasControlledByCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseWasControlledBy a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeWasControlledBy;
    d2rq:refersToClassMap map#shortName#:opm_agent;
    d2rq:join "OPMDependence.OPMDependenceCause => OPMAgent.OPMAgentKey";
    .

map#shortName#:opm_dependenceEffectWasControlledBy a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectWasControlledBy;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    .

# ---------- was triggered by --------------
map#shortName#:opm_dependenceWasTriggeredBy a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:WasTriggeredBy;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependence.OPMDependenceKey@@";
    d2rq:condition "OPMDependence.OPMDependenceType = 4";
    .

map#shortName#:opm_dependenceWasTriggeredByCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseWasTriggeredBy a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeWasTriggeredBy;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceCause => OPMProcess.OPMProcessKey";
    .

map#shortName#:opm_dependenceEffectWasTriggeredBy a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectWasTriggeredBy;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    .

# ----------- was derived from --------------
map#shortName#:opm_dependenceWasDerivedFrom a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:WasDerivedFrom;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMDependence.OPMDependenceKey@@";
    d2rq:condition "OPMDependence.OPMDependenceType = 5";
    .

map#shortName#:opm_dependenceWasDerivedFromCreationTime a d2rq:PropertyBridge;
```

```
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseWasDerivedFrom a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeWasDerivedFrom;
    d2rq:refersToClassMap map#shortName#:opm_artifact;
    d2rq:join "OPMDependence.OPMDependenceCause => OPMArtifact.OPMArtifactKey";
    .

map#shortName#:opm_dependenceEffectWasDerivedFrom a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectWasDerivedFrom;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMArtifact.OPMArtifactKey";
    .
# ------------------------------------------------
# MAPPING TABLE OPMAccount (OPMAccountKey bigint(20) unsigned, OPMEntityKey bigint(20) unsigned,
OPMAccountCreationTime datetime, OPMAccountDescription char(255))
# ------------------------------------------------
map#shortName#:opm_account a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Account;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMAccount.OPMAccountKey@@";
    .

map#shortName#:opm_accountId a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_account;
    d2rq:property :id;
    d2rq:column "OPMEntity.OPMEntityid";
    d2rq:join "OPMAccount.OPMAccountKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_accountValue a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_account;
    d2rq:property :value;
    d2rq:column "OPMEntity.OPMEntityValue";
    d2rq:join "OPMAccount.OPMAccountKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:string;
    .

map#shortName#:opm_accountAnnotTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_account;
    d2rq:property :annotationTime;
    d2rq:column "OPMEntity.OPMEntityAnnotationTime";
    d2rq:join "OPMAccount.OPMAccountKey => OPMEntity.OPMEntityKey";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_accountCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_account;
    d2rq:property :creationTime;
    d2rq:column "OPMAccount.OPMAccountCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_accountDescription a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_account;
    d2rq:property :description;
    d2rq:column "OPMAccount.OPMAccountDescription";
    d2rq:datatype xsd:string;
    .

# ------------------------------------------------
# MAPPING TABLE AccountAssociations ( OPMEntityKey bigint(20) unsigned, OPMAccountKey bigint(20)
unsigned)
# ------------------------------------------------
map#shortName#:opm_entityAccounts a d2rq:PropertyBridge;
       d2rq:belongsToClassMap map#shortName#:opm_entity;
```

```
    d2rq:property opmo:account;
    d2rq:refersToClassMap map#shortName#:opm_account;
    d2rq:join "AccountAssociations.OPMAccountKey => OPMAccount.OPMAccountKey";
    .
```

Part II. Distributed dependencies mappings (example for `opmo:Used` dependence)

```
# ------------------------------------------------
# MAPPING TABLE OPMExternalDependence (OPMDependenceKey bigint(20) unsigned, OPMDependenceType
bigint(20) unsigned, OPMDependenceCause, OPMDependenceEffect bigint(20) unsigned,
OPMDependenceCreationTime datetime, causeExternal bool, DBExternalId char(20))
# ------------------------------------------------
# ---------- used ----------
map#shortName#:opm_dependenceUsed a d2rq:ClassMap;
    d2rq:dataStorage map#shortName#:OPMDatabase;
    d2rq:class opmo:Used;
    d2rq:uriPattern "#OWL_URI_BASE#@@OPMExternalDependence.OPMDependenceKey@@";
    d2rq:condition "OPMExternalDependence.OPMDependenceType = 1";
    .

map#shortName#:opm_dependenceUsedCreationTime a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property :creationTime;
    d2rq:column "OPMExternalDependence.OPMdependenceCreationTime";
    d2rq:datatype xsd:dateTime;
    .

map#shortName#:opm_dependenceCauseUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeUsed;
    d2rq:refersToClassMap map#shortName#:opm_artifact;
    d2rq:join "OPMExternalDependence.OPMDependenceCause => OPMArtifact.OPMArtifactKey";
    d2rq:condition "OPMExternalDependence.causeExternal = true";
    .

map#shortName#:opm_dependenceCauseUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:causeUsed;
    d2rq:refersToClassMap map#shortNameExternal#:opm_artifact;
    d2rq:join "OPMExternalDependence.OPMDependenceCause => OPMArtifact.OPMArtifactKey";
    d2rq:condition "OPMExternalDependence.causeExternal = false";
    d2rq:condition "#shortNameExternal# = @@OPMExternalDependence.DBExternalId@@";
    .

map#shortName#:opm_dependenceEffectUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectUsed;
    d2rq:refersToClassMap map#shortName#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    d2rq:condition "OPMExternalDependence.causeExternal = true";
    .

map#shortName#:opm_dependenceEffectUsed a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map#shortName#:opm_dependenceUsed;
    d2rq:property opmo:effectUsed;
    d2rq:refersToClassMap map#shortNameExternal#:opm_process;
    d2rq:join "OPMDependence.OPMDependenceEffect => OPMProcess.OPMProcessKey";
    d2rq:condition "OPMExternalDependence.causeExternal = false";
    d2rq:condition "#shortNameExternal# = @@OPMExternalDependence.DBExternalId@@";
    .
```

Appendix E: Glossary

Term	Definition
Agent	In OPM terminology, an entity that can control a process – either human (researcher, clinician, etc.) or machine (software system).
Artifact	In OPM terminology, an entity that is produced and used by processes – single version of a data set, or a particular query posed. One of the main issues in provenance modelling is to decide how finely grained artifacts will be.
Clinical Data Integration Model	The model developed in WT6.5 (MS6) that captures the key set of data concepts needed to query TRANSFoRm data sources, based on the requirements of the use cases.
Clinical Evidence Model	The ontology developed in WT4.3 to represent the concepts relevant to decision support.
Clinical Research Information Model	The UMLS model representing the research processes in TRANSFoRm. Based on Primary Care Research Object Model (PCROM).
Clinical Research Information Provenance Profile	Ontology representation of CRIM's UMLS model, used by the Provenance framework.
D2RQ	Language for accessing relational databases as virtual, read-only RDF graphs.
Local Provenance Data Store	Database containing the provenance information produced at the local site
Open Provenance Model	Community based standard for representing and manipulating provenance information. Currently being used as a basis for PROV, a full W3C standard.
Ontology Web Language	Language for representing ontological information in a computable form. Uses RDF as a syntax.
Primary Care Research Object Model	UMLS model that is the basis for CRIM.
Process	In OPM terminology, an event that happened and affected some artifact, or triggered another process. Note that in a concrete provenance graph, it always refers to something that already happened – provenance is never captured in advance.
Profile	In OPM terminology, a subset of entities and relations between them that is visible to a particular set of users. Used to restrict visibility and implement security.
Provenance	Computable representation of the lineage of some entity. Commonly shown in graph format. Can also be seen as a semantically enabled execution log.

Provenance template	Abstract process model that specifies all possible provenance traces of a particular task. Used to facilitate provenance-awareness in client software tools, and ensure correctness of submitted provenance graphs.
SPARQL	Popular query language for querying ontological information.
Uniform Resource Identifier	Method for identifying entities on the Web, heavily used in RDF. Typical examples are Web URL-s, which are URI-s for Web pages.
XML	Mark-up syntax intended to be readable by humans and computable by computer engines.

Appendix F: List of Abbreviations

Acronym	Definition
API	Application Programmer's Interface
BPMN	Business Process Modeling Notation
CDIM	Clinical Data Integration Model
CEM	Clinical Evidence Model
CPDS	Centralized Provenance Data Store
CRIM	Clinical Research Information Model
CRIPP	Clinical Research Information Provenance Profile
DAG	Directed Acyclic Graph
DBMS	Database Management System
D2RQ	Database To Relational Query
GUI	Graphical User Interface
LPDS	Local Provenance Data Store
OPM	Open Provenance Model
OPMO	Open Provenance Model Ontology
OPMV	Open Provenance Model Vocabulary
OWL	Ontology Web Language
PCROM	Primary Care Research Object Model
RDF	Resource Description Format
RDFS	Resource Description Format Schema
RMI	Remote Method Invocation
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
URI	Uniform Resource Identifier
XML	eXtended Markup Language