EU Project No: 601043(Integrated Project (IP))

DIACHRON

Managing the Evolution and Preservation of the Data Web DIACHRON

<table>
<thead>
<tr>
<th>Dissemination level:</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Document:</td>
<td>Report</td>
</tr>
<tr>
<td>Contractual date of delivery:</td>
<td>M8</td>
</tr>
<tr>
<td>Actual Date of Delivery:</td>
<td>29/11/2013</td>
</tr>
<tr>
<td>Deliverable Number:</td>
<td>D1.3</td>
</tr>
<tr>
<td>Deliverable Name:</td>
<td>Design of the DIACHRON Platform and Applications</td>
</tr>
<tr>
<td>Deliverable Leader:</td>
<td>ATHENA</td>
</tr>
<tr>
<td>Work package(s):</td>
<td>WP1</td>
</tr>
<tr>
<td>Status &amp; version:</td>
<td>Final version</td>
</tr>
<tr>
<td>Number of pages</td>
<td>53</td>
</tr>
<tr>
<td>WP contributing to the deliverable</td>
<td>WP1</td>
</tr>
<tr>
<td>WP / Task responsible</td>
<td>WP1/T1.5</td>
</tr>
<tr>
<td>Coordinator (name / contact)</td>
<td>Christos Pateritsas (ATHENA), <a href="mailto:pater@imis.athena-innovation.gr">pater@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>Other Contributors</td>
<td>J. Cheney (UEDIN), G. Flouris (FORTH), P. Hasapis (INTRASOFT), S. Jupp, (EBI-EBML), C. Lange (UBONN), M. Meimaris (ATHENA), G. Papastefanatos (ATHENA), Y. Stavrakas (ATHENA), G. Thor Briem (DataMarket)</td>
</tr>
<tr>
<td>EC Project Officer</td>
<td>Federico Milani</td>
</tr>
<tr>
<td>Keywords:</td>
<td>architecture, design, SOA, dataset model, resources, integration</td>
</tr>
<tr>
<td>Abstract (few lines):</td>
<td>This document presents an overview of the architecture adopted for the DIACHRON. An SOA approach was adopted in order to benefit from the advantages of the service orientation paradigm. The system’s dataset model is also presented here. This model will be used for internal representation of all datasets handled and stored by DIACHRON.</td>
</tr>
<tr>
<td>Ver.</td>
<td>Date</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>-</td>
<td>15/10/13</td>
</tr>
<tr>
<td>-</td>
<td>18/10/13</td>
</tr>
<tr>
<td>-</td>
<td>22/10/13</td>
</tr>
<tr>
<td>0.1</td>
<td>27/11/13</td>
</tr>
<tr>
<td>-</td>
<td>29/11/13</td>
</tr>
<tr>
<td>1.0</td>
<td>29/11/13</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1 INTRODUCTION .................................................................................................................. 6

1.1 Scope and Objectives ........................................................................................................... 6
1.2 Context of the Document ..................................................................................................... 6
1.3 Structure of the Document ................................................................................................ 6
1.4 Definitions, Acronyms and Abbreviations ........................................................................ 6

2 SYSTEM DESIGN .................................................................................................................. 8

2.1 The SOA Approach .............................................................................................................. 8
  2.1.1 The Four Common Types of SOA ................................................................................. 8
  2.1.2 SOA for DIACHRON ................................................................................................. 9

2.2 Platform Layer - Service Inventory ................................................................................... 11
  2.2.1 Data Storage - Query Engine Module ....................................................................... 12
  2.2.2 Change Detection Module ....................................................................................... 12
  2.2.3 Propagation/Repairing Module ................................................................................. 12
  2.2.4 Validation Module ..................................................................................................... 12
  2.2.5 Monitoring Module .................................................................................................... 12
  2.2.6 Citation Module ........................................................................................................ 13
  2.2.7 Quality Evaluation Module ...................................................................................... 13
  2.2.8 Crawling Module ...................................................................................................... 13

2.3 Integration Layer - Service Composition ........................................................................... 13
  2.3.1 Query API .................................................................................................................. 14
  2.3.2 Archiving Service ....................................................................................................... 14
  2.3.3 Propagation Service .................................................................................................... 14
  2.3.4 Monitoring Service ..................................................................................................... 14
  2.3.5 Validation and Repairing Service ........................................................................... 14
  2.3.6 Quality and Ranking Service ..................................................................................... 14
  2.3.7 Crawling Service ...................................................................................................... 14
  2.3.8 Authentication and Security Service ........................................................................ 14

2.4 Pilot Applications .............................................................................................................. 15

3 THE DIACHRON DATASET MODEL ..................................................................................... 16

3.1 Objectives .......................................................................................................................... 16
3.2 Entities ................................................................................................................................ 16
  3.2.1 Entity Identification ................................................................................................... 17
  3.2.2 Temporal and Provenance Annotations ..................................................................... 17
  3.2.3 Attributes ................................................................................................................ 18

3.3 Entity Types ....................................................................................................................... 18
  3.3.1 Diachronic Datasets .................................................................................................. 18
  3.3.2 Datasets (dataset instantiations-versions) ................................................................. 18
  3.3.3 Schema Objects ........................................................................................................ 18
  3.3.4 Records ..................................................................................................................... 19
  3.3.5 Diachronic Resources .............................................................................................. 20

3.4 Changes .............................................................................................................................. 23
3.4.1 Resources as Change Contexts ........................................................................................................24
3.5 DIACHRON DATA MODEL INSTANTIATIONS EXAMPLES ........................................................................24
3.5.1 Multidimensional Data: Eurostat Data ..................................................................................................24
3.5.2 Population Projections ............................................................................................................................26
3.5.3 Assumptions ............................................................................................................................................26
3.6 CASE: GENERIC MULTIDIMENSIONAL DATA EXPRESSED WITH THE DATA CUBE VOCABULARY ..............29
3.6.1 Ontological Data: Gene Expression Atlas ..............................................................................................32
3.6.2 Experiment Instantiation .........................................................................................................................34
3.6.3 Experimental Factor Ontology .................................................................................................................36
3.6.4 Zooma ......................................................................................................................................................37
3.7 SUMMARY ..................................................................................................................................................39

4 HOW THE DESIGN MEETS THE REQUIREMENTS ....................................................................................43

5 CONCLUSION ..................................................................................................................................................52

6 REFERENCES ..................................................................................................................................................53
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Architectural levels of the service-orientation paradigm</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Conceptual overview diagram of the system’s architecture</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>DIACHRON Entities</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Datasets</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Records inherit the model of DIACHRON entities but are also described by one or more record attributes</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Resources example</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>The resource set (explicit or inferred) that contains DIACHRON Resources exists within a universal scope and is linked with particular occurrences in records</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>Changes between two dataset instantiations</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Instantiation of the data model for observations 1 and 2 at times $T_m$ and $T_n$</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>Going from Resources (Observation 1) to evaluated contexts (depicted as documents with rows of records), to differences between evaluated contexts</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>The context of “E-GEOD-1085” represented in the DIACHRON data model, at time $T_n$. Not all information is presented in the figure for brevity. However, it can be seen that there are Resources as subjects that are different than “E-GEOD-1085”, such as assays and samples.</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>Going from Resources (<a href="http://rdf.ebi.ac.uk/resource/atlas/E-GEOD-1085">http://rdf.ebi.ac.uk/resource/atlas/E-GEOD-1085</a>) to evaluated contexts (depicted as documents with rows of records), to differences between evaluated contexts</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>Instantiation of the data model for efo:EFO_0000887 at times $T_m$ and $T_n$.</td>
<td>37</td>
</tr>
<tr>
<td>14</td>
<td>Instantiation of the data model for annotation _A1 at time $T_m$</td>
<td>38</td>
</tr>
<tr>
<td>15</td>
<td>Instantiation of the data model for annotation _A2 at time $T_n$. The relation between _A1 and _A2 (zooma:replaces) is also shown in the figure, which shows how an existing versioning model can be incorporated to the data model</td>
<td>39</td>
</tr>
<tr>
<td>16</td>
<td>Overview of the data model, divided into a 2x2 grid. Diachronic and non-diachronic entities are shown in their respective spaces. Dataset D1 is instantiated at times $T_m$ and $T_n$ for exemplary purposes. Entities marked with an asterisk (*) are abstracted in figure 17</td>
<td>40</td>
</tr>
<tr>
<td>17</td>
<td>Entities marked with an asterisk in figure 16 are abstracted as shown here</td>
<td>41</td>
</tr>
<tr>
<td>18</td>
<td>Basic modeling concepts in DIACHRON and their relationships</td>
<td>42</td>
</tr>
<tr>
<td>19</td>
<td>Diagram 1. Inserting a new dataset in the DIACHRON system</td>
<td>44</td>
</tr>
<tr>
<td>20</td>
<td>Diagram 2. Update of an existing dataset (new version)</td>
<td>45</td>
</tr>
<tr>
<td>21</td>
<td>Diagram 3. Querying for a data</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>Diagram 4. Monitoring for changes on through scheduling a task</td>
<td>47</td>
</tr>
<tr>
<td>23</td>
<td>Diagram 5. Propagation of schema changes to previous dataset versions as part of a temporal query execution</td>
<td>48</td>
</tr>
<tr>
<td>24</td>
<td>Diagram 6. Validation and repairing of a dataset</td>
<td>49</td>
</tr>
<tr>
<td>25</td>
<td>Diagram 7. Quality assessment</td>
<td>50</td>
</tr>
<tr>
<td>26</td>
<td>Diagram 8. Creation of dataset through crawling</td>
<td>51</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 SCOPE AND OBJECTIVES

This document’s objective is to present a conceptual architectural model for the DIACHRON system in correspondence with the deliverable D1.1 “Requirements of the DIACHRON Use Cases”. It aims at presenting a high level overview of the system along with some design decisions and guidelines that will affect and guide the detailed design of the system that will be done as part of Task 6.1 of the project. Since DIACHRON is a data centric project, special focus was given to present a conceptual data framework which will enable the system to serve its purpose.

The work presented in the document regarding the DIACHRON Dataset Model is an essential input for all the research work packages of the project (WP2-WP5). This document is the result of the work done by project partners in Task 1.5.

1.2 CONTEXT OF THE DOCUMENT

The document is mainly composed of two parts. The first part presents the service oriented architecture of the system. A SOA approach was selected for DIACHRON since the advantages of service orientation can benefit the project as the document further explains. A three layered architecture was also selected in order to better enable the separation of concerns not only in software terms but also in the task allocation between the partners. The second part of the document describes the DIACHRON Dataset Model. This model will serve as the primary data model on which all functionality of the project will be implemented. Although the pilot partners internally use different data models, a decision was made to map this data models to a common one to allow the functionality to implemented around a unified data model.

1.3 STRUCTURE OF THE DOCUMENT

The document is structured as follows:

Chapter 2 first analyses the SOA approach for the project and then presents the architecture of the system with a conceptual diagram of it accompanied by descriptions of its parts. Chapter 3 describes the DIACHRON Dataset model and provides also examples of its usage with real world datasets provided from the pilots. Chapter 4 associates the architecture of the system with the functional requirements of the project and finally Chapter 5 concludes this document by summarizing the content of the document.

1.4 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

DoW: Description of Work
EBI: European Bioinformatics Institute
EFO: Experimental Factor Ontology
EMBL: European Molecular Biology Laboratory
SOA: Service-Oriented Architecture
SPARQL: SPARQL Protocol and RDF Query Language (recursive acronym)
URI: Uniform Resource Identifier
WP: Work Package
2 System Design

2.1 The SOA Approach

The Service-Oriented Architecture is a software architecture model that is based on the creation of solution logic units, namely discrete and independent software units that provide their functionality as services to other applications.

These units and the services they implement must be used collectively and repeatedly with the purpose of providing the functionality of a larger software system and support the business processes and strategic goals of the organisation. This design paradigm is called “service orientation” and is composed of some basic design principles [1]:

- Standardized Service Contract: Communication of services with other services or other software applications must adhere to a well-defined communication agreement contract, the service contract, which is defined in the service description documents.
- Service Loose Coupling: The dependency of the overall software architecture on a particular service and the coupling of the service to the system are defined only by the service contract.
- Service Abstraction: Service contracts should only contain the necessary information and information about the usage of services. All other information (such as implementation specific logic or technologies used) are hidden from the outside world.
- Service Reusability: Services contain and express agnostic logic and provide functionality with the intention of promoting reuse of the service for the completion of several business processes.
- Service Autonomy: Services possess the complete control of their internal design and execution environment.
- Service Statelessness: Services minimize resource consumption by deferring the management of state information when necessary.
- Service Discoverability: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- Service Composability: Services are effective composition participants, regardless of the size and complexity of the composition.

2.1.1 The Four Common Types of SOA

There are four common architecture levels in SOA for the various granularity levels of SOA[2].

- Service Architecture: The architecture of a single service.
- Service Composition Architecture: The architecture of a set of services assembled into a service composition.
- Service Inventory Architecture: The architecture that maintains a set of related services that are independently standardized, implemented and managed.
- Service-Oriented Enterprise Architecture: The architecture of the enterprise itself, to whatever extent it is service-oriented.

The following figure depicts all the aforementioned architectural levels.
At the 2nd International SOA Symposium, which took place in October 2009, a group of independent SOA practitioners and vendors formed the "SOA Manifesto Working Group". This group during the symposium composed and announced the publication of the SOA Manifesto [3]. The SOA Manifesto is a set of objectives and guiding principles to provide a clear view of service-orientation and SOA.

The manifesto offers a broad definition of SOA, the values it denotes for the signatories and some guiding principles. The manifesto prioritizes:

- Business value over technical strategy
- Strategic goals over project-specific benefits
- Intrinsic interoperability over custom integration
- Shared services over specific-purpose implementations
- Flexibility over optimization
- Evolutionary refinement over pursuit of initial perfection

As of November 2013, the SOA Manifesto had been signed by more than 1000 signatories and had been translated to eleven languages.

### 2.1.2 SOA FOR DIACHRON

The SOA model is well suited for the DIACHRON project since a significant degree of service orientation can be achieved for the platform design. Moreover, a SOA model will serve well the specific needs of such a project. Large IP projects involving several partners are perfect candidates for SOA where the diversity of technologies used by each partner and the different time planning for each partner...
regarding its contribution can be addressed smoothly by a SOA model both at integration level but also in project management level.

DIACHRON core functionality can be implemented as a set of independent services (service inventory) since most of them can be aligned with the service orientation principles. Using this service inventory, a service composition layer can be integrated in order to provide more complex services.

1.1. Overview - Layers
In DIACHRON a three layered architecture will be implemented and deployed. Figure 1 presents a conceptual overview of the system. The three layers of the system are:

- **Platform layer - Service Inventory**
  The first (bottom) layer is the core DIACHRON platform and the SOA perspective acts as a service inventory. This layer includes the core functionality of the platform that corresponds to the research challenges of the project.

- **Integration layer - Service Composition**
  The second (middle) layer is a service composition and integration layer and acts as a controller/orchestrator of the functionality of the core layer services in order to provide more complex services to the users.

- **Pilot applications**
  The third (top) layer contains the pilot applications that will be used to demonstrate the system’s functionality and its added value to the pilot-specific business processes.
2.2  PLATFORM LAYER - SERVICE INVENTORY

This layer is a service inventory layer, therefore each module of this layer is considered an independent service. The services of this layer are independent of each other, namely none of them uses another service of this layer to implement its functionality. There isn’t any kind of direct data or message exchange between them. This allows for the system to operate even in the case where some services are disabled. Of course in such a case the system will not be able to provide the overall functionality to its full extent, but whatever adjustments need to be made are limited to the Service layer (integration). The independence of this layer according to the service orientation paradigm is pursued in order to benefit from the advantages of service orientation. More specifically, it provides a larger degree of flexibility in the technologies selected for each service module and its associated deployment options. Additionally, it will allow for separate development planning and development cycles, which can reduce the risks in the cooperation of the project’s partners.
Although each module will be responsible for exposing its own interface for providing functionality to the above layer, all modules will exchange data with this layer under the same data model. This will minimize data transformation functions in the service layer and it will try to limit such functions to the data endpoints.

2.2.1 DATA STORAGE - QUERY ENGINE MODULE
This module provides the persistent storage functionality of the platform for datasets. It accepts datasets and dataset changes modelled after the DIACHRON dataset model and stores them into the archive. It also executes queries expressed in the DIACHRON query language, retrieves the corresponding datasets or dataset parts from the archive and returns them to the service user. The DIACHRON query language will be designed as part of Work Package 2 and it might be an existing language such as SPARQL [4] with the necessary extensions or a completely new query language designed for DIACHRON. The implementation of the query engine and the data storage mechanism is part of the Work Package 4 of the project.

2.2.2 CHANGE DETECTION MODULE
This service accepts as input two versions/instances of a DIACHRON dataset and performs change detection between them. It reports the changes modelled after a change representation model. Change detection can be performed in both the data part and the schema part of the dataset. Additionally change detection can also be performed on a semantically richer level as defined by the Resources context (see paragraph 3.3.5) and by change templates. Change templates are groups of simple change types and/or constraints and rules applied on simple changes properties as defined in [5]. This service will be implemented as part of the work conducted in the Work Package 3 of DIACHRON.

2.2.3 PROPAGATION/REPAIRING MODULE
This service provides mechanisms for propagating changes on the schema part of a dataset to the actual data. Depending on whether the schema and the data belong to the same dataset instance or not, this function can be considered either a repairing action or a propagation action correspondingly. The service applies a set of changes on an input dataset according to a set of propagation rules (guidelines). This service can also be used for repairing as a post-validation action in case of invalid datasets. The implementation of this module belongs also to Work Package 3.

2.2.4 VALIDATION MODULE
This service provides functionality for validating the data of a dataset against the dataset’s schema or other user defined constraints and integrity rules. The validation service doesn’t provide the functionality to execute post-validation actions, since this is offered by the propagation/repairing module. This is also a Work Package 3 task.

2.2.5 MONITORING MODULE
The monitoring service provides functionality for creating dataset monitoring tasks where the user registers the dataset of her interest that she wants to be notified for changes and the desired period that the dataset should be probed for changes. Additionally the service can provide an estimation of the change interval of the dataset so as to optimize the update task.
Datasets can also be registered to be monitored for changes in any part of them but only with respect to the correlation / dependencies to other datasets.

2.2.6 Citation Module

This module creates a Diachronic citation for a given dataset or a part of it or a specific version-instance of the dataset. A Diachronic citation is a reference schema (such as a URI) that can be used by the platform’s user as a reference mechanism to the contents of the archive. These citations must also be dereferenceable to the actual data, so this module provides the dereferencing mechanism for resolving the citations to platform-internal identifiers. Diachronic citations will be studied and implemented as part of Work Package 2.

2.2.7 Quality Evaluation Module

This module performs quality evaluation on a Dataset or a group of Datasets. Various quality dimensions and corresponding metrics and indicators can be calculated. These will be further defined and implemented in Work Package 5.

2.2.8 Crawling Module

This module provides to the platform the mechanism to reach the World Wide Web in order to retrieve a dataset from a single source or to crawl web pages or the LOD cloud in order to construct a new dataset. The crawling module is also scheduled to be implemented in Work Package 5.

2.3 Integration Layer - Service Composition

This layer is a middleware layer with a dual role. It encapsulates the services of the core layer and re-exposes the DIACHRON functionality by composing more complex services and by homogenizing the service contracts and endpoints. Additionally, it provides common functionality such as security and auditing control, logging, storage area for configuration information, etc.

The components of this layer act as orchestrators of the core functionality provided by the platform layer. Their role is to execute specific workflows that usually involve more than one core service from the platform layer in order to create service compositions and expose more composite functionality to the pilot applications that use them. Special care must be taken in the design of these workflows so as to provide when possible alternative workflows in case some of the core modules are unavailable (either due to malfunction or due to deployment scenarios).

Additionally this layer provides messaging coordination and synchronization for the communication of the service composition with the basic services of the platform layer.

This layer will be designed in detail and implemented as part of Work Package 6.
2.3.1 Query API

This component exposes an API and acts as the primary data endpoint of the platform. It performs queries to the archive using the DIACHRON query language and also uses the citation service in order to dereference and return datasets to the users.

2.3.2 Archiving Service

This component is the primary data input channel, as datasets or temporal versions of them that need to be stored in the archive must go through this component. As a service composition it uses the change detection service to detect changes and feed them to the Data storage module in case of an already existing dataset. In case of a new dataset it just pushes the dataset to the Archive.

2.3.3 Propagation Service

The propagation component is responsible for orchestrating the propagation of changes to previous versions of a dataset or to another linked dataset. The component is also responsible for acquiring from the user the propagation rules that are necessary for this action and also stores them for future reuse.

2.3.4 Monitoring Service

The monitor composite service uses the monitor service of the platform along with the crawling, the change detection and the archive modules in order to expose the complete monitoring functionality to the user where the user registers the dataset, the service periodically retrieves the dataset and performs change detection with its previous version and notifies the user accordingly.

2.3.5 Validation and Repairing Service

This component is responsible for validating and, if needed, repairing datasets according to sets of validation and repairing rules. The component is also responsible for storing these rules in case they are meant to be reused in several occasions.

2.3.6 Quality and Ranking Service

This component exposes the functionality of the quality evaluation module and also provides datasets ranking functionality based on the quality evaluations of a group of datasets or the series of temporal versions of the same dataset.

2.3.7 Crawling Service

Provides functionality for guiding the Crawling module to more composite tasks such as creation of a dataset from the WWW according to specific crawling scenarios and specifications. The component also acts as storage mechanism for these crawling configurations.

2.3.8 Authentication and Security Service

This component provides authentication functionality for the system and also, if needed, more complex security and auditing services (selective access to dataset, logging, etc.). Depending on the pilot deployment scenario, various user roles must be defined with varying granularity in access rights to the system services.
2.4 **Pilot Applications**

Pilot applications will be using the Services layer to offer DIACHRON functionality to the end users. Each pilot application should follow its own architecture in order to adapt better to the special needs of each pilot. The architecture for this layer will be different each pilot and will be designed as part of the corresponding work package for each use case scenario of the system (WP 7-9). Figure 1 depicts indicative architectures for each pilot. It must be noted here that the initial deployment plan at this phase is to deploy one DIACHRON system for each use case scenario. This option was decided in order to provide better customisation of the system through appropriate configurations and to address legal and security issues of private and proprietary datasets that the pilot partners are not allowed to disclose in a remote location.
3 THE DIACHRON DATASET MODEL

3.1 OBJECTIVES

This chapter presents the DIACHRON Dataset Model. The model is intended for use in the integration layer of the platform and possibly within the archive (WP4) as a way to store, monitor, query and retrieve datasets from various heterogeneous sources and origins. Considering the DIACHRON DoW document, the key features of such a design should include format-independence, data traceability and reproducibility, facilitation of the DIACHRON platform services and an overall common denomination for data that originate from different models (e.g. triple-based, tuple-based, multidimensional etc.).

Furthermore, the model intends to support a format-independent platform that maintains syntactic integrity. The original datasets must be reproducible from the DIACHRON platform using the same mechanisms independently of the source formats. Therefore the model retains the information as modelled on the datasets and at the same time takes advantage of information-rich content and implicit or explicit semantics in incoming datasets. The model provides a framework that can be used to define semantically richer entities on top of the semantics already depicted by the source datasets.

Moreover it sets the unified environment on which the DIACHRON objectives of data evolution and provenance annotation can be seamlessly managed at various levels of data granularity.

3.2 ENTITIES

DIACHRON entities are identifiable and citable objects, both within DIACHRON and outside the platform. In this sense, the following types of objects are considered as entities:

- Diachronic Datasets (time-agnostic entity that represents a dataset)
- Datasets (time-specific instantiation of a diachronic dataset)
- Schema Objects (e.g. classes and properties in ontologies, or tables and columns in relational databases)
- Diachronic Resources (e.g. ontology instances, linked data resources etc.)
- Records (individual dataset entries as collections of facts about resources, statistical observations etc.)
- Changes

These entities are modelled in such a way that their basic information is captured at the following three levels, as can be seen in Figure 3:

- Identification
- Attributes (things that describe the entity or that are linked with the entity in a direct way)
- Provenance

This offers the advantage of facilitating the various functionalities of DIACHRON on many different levels, as well as using the same mechanisms in change detection and provenance tracking independently of what is being monitored, thus providing fine granularity at different syntactic and semantic levels.
Every type of entity can hold its respective provenance metadata; every entity can be compared to entities of the same type for changes, not only across datasets, but across different versions of the same dataset as well.

### 3.2.1Entity Identification

In any case, there should be a uniform way of providing identifiers within DIACHRON, given that entities are identified in different ways across formats (e.g. resource URIs vs. primary keys, property URIs vs. column names). For this reason, a level of abstraction is needed in order to bring the identifiers to the same level. One way to achieve this is to provide an Identifier concept that will reify the information about the identifier itself as well as the identifier’s nature (functional as well as non-functional metadata). Furthermore, using this level of abstraction the identifiable entity can be associated not only with its identifier(s), but with a set of DIACHRON-related metadata such as temporality, provenance and so on. This can be seen in Figure 3.

![Figure 3: DIACHRON Entities](image)

### 3.2.2Temporal and Provenance Annotations

Following the aforementioned approach, it is possible to provide metadata on DIACHRON entities in a multitude of levels. For example, assertions on the temporal characteristics (e.g. lifetime, temporal version, etc.) can be given on all types of entities described above. A particular dataset instantiation can be associated with information such as the time it became relevant and/or obsolete, which version preceded/succeeded it (if any) and so on. Similar assertions can be given for a particular resource or schema object. Query provenance can be tracked directly on particular records (or record sets) in a dataset, in schema objects and so on. The representation of such information is subject to a decision on
the representation model (e.g. W3C PROV ontology for provenance [6]). This will be directly pluggable in the provenance subgraph that can be seen in Figure 3.

3.2.3 Attributes

The term attributes is used to (informally) refer to descriptional specificities that are dependent on the particular entity type. The differentiation is for representational purposes and should not be considered as something hard-coded in the data model. However, it provides useful conceptual boundaries of the actual information that an entity holds from the provenance and identification sub graphs of the entity at hand (Figure 3). The specificities of attributes for each entity are described in the following paragraphs.

3.3 Entity Types

3.3.1 Diachronic Datasets

Diachronic datasets are defined as conceptual entities that represent a particular dataset from a time-independent point of view. Different temporal instantiations of the same diachronic dataset are linked to the latter in the data model. Furthermore, diachronic dataset metadata are comprised of information that is not subject to change, such as diachronic identifiers and (non-changing) labels. The identifiers of diachronic datasets will serve as ways to refer to the datasets in a time and/or version unaware fashion (i.e. diachronic citations).

3.3.2 Datasets (Dataset Instantiations-versions)

Instantiations of Diachronic Datasets are considered DIACHRON entities in the sense meant in this document. They will be referenced as datasets throughout this document, thus they are given identifiers and metadata descriptions. These data hold information on how and when a particular dataset instantiation was relevant, active, trusted and so on. In essence they define (temporal) versions of diachronic datasets. The instantiations do not contain their records, rather they are linked to them via one or more Record Set objects, which can either be explicitly created or inferred. Record sets are, in essence, collections of records that exist within a particular dataset instantiation and they are comprised of one or more records, as described in Figure 4. The motivation is that, given a particular record set and the dataset’s metadata information, the dataset instantiation can be recalled on demand and reproduced in its original form, both information-wise and syntax-wise. Furthermore, past instantiations of datasets can be interpreted using different schemata, not only the one(s) relevant to the time the instantiation was current.

3.3.3 Schema Objects

This type of entity is used to cover a variety of schema-related objects from different formats/data models/paradigms. For instance, given an ontology, and in particular an ontology’s T-Box, the classes along with their class restrictions, the properties and their definitions (domains, ranges, meta properties depending on the expressivity) are modelled as DIACHRON entities. The motivation behind this lies in the facts that they become referenceable and reusable across datasets within the context of DIACHRON and its universal data model and, more importantly, they can be monitored for changes and evolution as distinct entities or sets of entities. Furthermore, schema versions become pluggable (i.e. can be applied) on datasets independently of whether they co-existed in a dataset instantiation or not.
When changes are identified between two temporally different versions of the same schema object (e.g. a class) or set of objects (e.g. a set of classes and properties), rules can be triggered in order to propagate the implications of the evolving schema to the rest of the DIACHRON entity levels that are affected by the changes. However, this is relevant to the change propagation functionality and not an inherent functionality of the data model.

![Diagram of datasets](image)

**Figure 4. Datasets**

### 3.3.4 Records

Records are described as format-independent, indivisible entries of data that are stored within the archive. For example, tuples, triples or rows can all be considered as DIACHRON records. Unique identifications of records will be created in order to make annotation feasible on record level. This unique identification in order to match them across temporal versions might not always be possible (what qualifies as record equality is an open question in the context of this model), therefore this still remains a subject open to discussion. Records are considered as super-types of the originally model-dependent entries that originate from their respective datasets (e.g. tuples, triples) in order to attribute provenance, temporality and changes on them. Given that a record can be comprised of one or more binary relationships, depending on the original format, it can be thought of as a bag holder of one or more record attributes, as can be seen in Figure 5. However, in an actual instantiation the relations become edges, such as RDF properties, as can be seen in the examples that will follow.

In this sense, a tuple from a relational table is considered to be a record in the DIACHRON model. In the same manner, an RDF triple can be considered to be a record of its own, considering that subjects, predicates and objects are treated the same way as columns in a table. Records, when seeing everything as rows, can be considered as distinct rows. This, however, has the following consequence: while a tabular record can have many record attributes, an RDF record can only have a fixed amount of record attributes,
as its number of columns is immutable (subject-predicate-object and potentially a fourth column representing the named graph). Given the above, an RDF-based DIACHRON platform will find the use of “Record Attributes” nodes redundant.

**F**igu**r**e 5. **R**ecords inherit the model of DIACHRON entities but are also described by one or more record attributes.

### 3.3.5 Diachronic Resources

Diachronic Resources are *identifiable concepts that conform to a formal definition*. They essentially provide a means to relate high level changes to contexts they refer to. The definition can be the result of curation or a default depending on the original data model. In any case, it consists in two parts, a *Resource identification definition* and a *Resource description definition*, as will be discussed in the following section. Given a particular dataset instantiation, the evaluation of a resource is defined as the application of the resource description definition to a particular identified Resource. This process outputs the *evolving context* of a Resource.
Resources can vary in nature across datasets and data formats. For example, given an ontology and its instantiation, all instances of classes can be identified as Resources. Given a table of employees in a relational database, a Resource in this sense can be a particular employee identified by his primary key. In the case of RDF, a Resource can be anything with a URI. A proper rule-based mechanism to identify Resource contexts requires two types of input from the user:

1. Rules for Resource identification. Examples:
   a. (RDF) Everything with a URI
   b. (RDF) Everything with a URI that appears as subject in at least one triple
   c. (RDF) All instances of class A
   d. (Multidimensional) All Observations
   e. (Multidimensional) Observations that measure unemployment in two fixed dimensions (e.g. fixed region and time period).
   f. (Multidimensional) A single observation that measures unemployment with all available dimensions fixed

2. Rules for Resource description. Examples:
   a. (RDF) All first level triples (i.e. all triples with the identified resource as a subject)

---

1 [http://www.w3.org/TR/vocab-data-cube/#reference-observations](http://www.w3.org/TR/vocab-data-cube/#reference-observations)
b. (RDF) All first and second level triples

c. (RDF) All first level triples, only the second level triples that come after property p1

d. (Multidimensional) All the values of the measures of the observations

Rules can be defaulted to generic assertions (such as cases 1a and 2a above) in case no input is provided from the user.

The persistence of Resources is dataset independent, however they need to appear at least once in DIACHRON—in order to be eligible to be considered as Diachronic Resources. Moreover, the same Resource might appear with many identifiers/instantiations (e.g. two URIs linked with owl:sameAs). Despite their time-independent nature, Resources manifest themselves in different ways across instantiations of the same dataset, or even across datasets.

As a first approach, the evolution of Resources across time will be recorded in each dataset they appear in separately, rather than fuse the data of conceptually same Resources. For example, if a Resource describing the city of Athens appears in a diachronic dataset of the statistics authority, the evolution of the city of Athens will be monitored as it appears in the particular dataset, even though it might also appear in another diachronic dataset from a different source.

In any case, same Resources from different datasets get aggregated and placed under a dataset-independent space, in a form of Resource index that holds information on the appearance of entities, their provenance metadata, as well as the particular appearances that are aggregated under them (e.g. different URIs with owl:sameAs links). Some aspects of this approach can be seen in the figure that follows. There, the DIACHRON Resource set exists outside of the space defined by the datasets and their records, but particular Resource instantiations are linked to the records they appear in.
3.4 Changes

Changes are subject to the change representation language/model that will be used by the DIACHRON components. The main idea, though, is that changes are only relevant to sets of two DIACHRON entities of the same type (e.g. changes between two Resources, two datasets, two schema objects etc.). In a time-oriented versioning scenario, changes come in Change Sets between two dataset instantiations of the same diachronic dataset. These are comprised of change entities that refer to changes between particular DIACHRON entity types (e.g. changes between record sets, changes between schemata and so on) in the two datasets under comparison. An example of this can be seen in Figure 8.
3.4.1 Resources as Change Contexts

Resources can be explicitly defined and curated in DIACHRON datasets. This way, DIACHRON users can explicitly state what they want to be able to track across time, as this provides added value for datasets and ground for innovation within DIACHRON. This, however, is not necessary. When the user is not interested in more semantic functionality that requires a certain amount of manual curation, Resources within the DIACHRON data model will be used as a concept to refer to already identified objects.

3.5 DIACHRON Data Model Instantiations Examples

In this chapter, two examples are presented on how datasets from the pilots are going to be mapped to the DIACHRON Dataset Model. The first one is from the Open Data scenario and the second from the Scientific Data Scenario.

3.5.1 Multidimensional Data: Eurostat Data

Namespaces

<table>
<thead>
<tr>
<th>qb</th>
<th><a href="http://purl.org/linked-data/cube#">http://purl.org/linked-data/cube#</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>eg</td>
<td><a href="http://example.com/">http://example.com/</a> (used as an example namespace)</td>
</tr>
</tbody>
</table>
Eurostat datasets mainly consist in multi-dimensional data that can generally be described using a multitude of existing schemata and vocabularies, such as the RDF data cube vocabulary\(^2\). This example will use Eurostat data on projected populations of EU members at 5-year intervals from 2015 to 2050\(^3\). A brief description drawn from the source URL:

*The Europop2010 (Eurostat Population Projections 2010-based) convergence scenario contains statistical information on population projections at national level, namely:*

- projected 1\(^{st}\) January population by sex and 5-year age group, by 5-year time interval,
- assumptions on total fertility rates (TFR), life expectancy at birth by sex and net international migration.

*Data comprise the EU27 Member States and the EFTA countries.*

These projections are calculated based on a set of assumptions that can also be considered as multidimensional data. The particular dataset is an interesting example of “versionable” statistical data given that future projections can change and not be considered as a new dataset, as is often the case with statistical data. Furthermore, given the linkage between the actual population projections and the underlying set of assumptions, the example will try to fit into the data model the same projections as seen in two temporally different versions, where some of the assumption data has changed because of re-evaluations.

The curator of the example defines the Resources he wants to monitor as the *statistical observations\(^4\) of each projection for each country for each reference year*. The Resource description (i.e. the change context) of observations is defined as the projected population observations along with the assumption observations. Evaluating the defined Resource would yield its first-level triples (the triples where the Resource is a subject in RDF terms) along with the triples that describe the assumptions used to calculate the dataset’s projections. This means that the curator sees an observation Resource as changed if one or both of the following cases take place:

1. The population projection measure value changes without some change in the assumptions (e.g. fixing typos).
2. A subset of the assumptions changes, thus requiring recalculation of the projections.

Notice that in case (1) the curator does not consider that a Resource change is triggered if some dimension value changes, because this is essentially a different observation.

\(^2\) [http://www.w3.org/TR/vocab-data-cube/](http://www.w3.org/TR/vocab-data-cube/)


\(^4\) The term observation refers to the value of a particular measure for particular dimension values.
3.5.2 POPULATION PROJECTIONS

In Data Cube terms, the Data Structure Definition of the projected population part of the dataset is as follows:

```tcl
# The dataset
eg:pop_proj2050 a qb:DataSet;
    qb:structure eg:dsd-pop_proj2050 .

# The Data Structure Definition
eg:dsd-pop_proj2050 a qb:DataStructureDefinition;
    # Dimensions
    qb:component [ qb:dimension eg:refArea];
    qb:component [ qb:dimension eg:refPeriod];
    # The measure
```

A TTL-serialized example Data Structure Definition for the particular dataset, as defined in the Data Cube specification.

As can be seen from the Data Structure Definition, there are two dimensions, namely eg:refArea and eg:refPeriod. They respectively capture the EU member state and projection year for a particular observation. Moreover, there is one measure defined, namely eg:projectedPopulation, which captures the actual value of the projected population for fixed dimension values. These are all captured using an observation Resource, as can be seen in the following:

```tcl
eg:obs1a a qb:Observation;
    qb:dataSet eg:pop_proj2050;
    eg:refArea "Greece";
    eg:refPeriod "2020";
    eg:projectedPopulation "11526085" .
```

A TTL-serialized example Observation Resource, essentially capturing the information that the projected value of the population of Greece for the year 2020 is 11,526,085.

3.5.3 ASSUMPTIONS

The part of the dataset containing the assumptions carries information about several indicators that are used in order to calculate the actual projections. Namely, these are:

- Net migration (total)
- Fertility rate (total)
- Life expectancy at birth (males)
- Life expectancy at birth (females)

The assumptions part of the dataset (here considered as a different dataset) is a multi-measure dataset and its Data Structure Definition can be seen in the following figure:

```tcl
# The dataset
eg:pop_proj2050_assumptions a qb:DataSet;
```
A TTL-serialized example Data Structure Definition for the particular dataset, as defined in the Data Cube specification.

An example observation concerning the aforementioned assumption measures for the case of Greece for 2020 is as follows:


Therefore, the curator tracks the two datasets for changes and notices that a few months after the initial publication of the datasets, the fertility rate value for Greece concerning the year 2020 changes from 1.55 to 1.60 due to new, more conclusive data. The DIACHRON platform evaluates the Resource descriptions of the defined Resources (i.e. all observations concerning projected populations in EU member states) and concludes that between the two versions, the observation for the projected population of Greece for 2020 is different. The instantiation of the model can be seen in the following figure.
FIGURE 9. INSTANTIATION OF THE DATA MODEL FOR OBSERVATIONS 1 AND 2 AT TIMES $t_m$ AND $t_n$.

Evaluating the Resource according to its description at times $t_m$ and $t_n$ will create a complex change of type “Projection change due to assumption change” that can be traced back to two low-level deltas, specifically:

- **Update Record 1**
  ```
  Delete eg:Obs1 eg:projectedPopulation "11526085".
  Insert eg:Obs1 eg:projectedPopulation "11482100".
  ```

- **Update Record 2**
  ```
  Delete eg:Obs1 eg:fertilityRateTotal "1.55".
  Insert eg:Obs1 eg:fertilityRateTotal "1.60".
  ```

The following figure shows how DIACHRON captures changes and relates them to changes in records.
Figure 10. Going from Resources (Observation 1) to evaluated contexts (depicted as documents with rows of records), to differences between evaluated contexts.

3.6 Case: Generic Multidimensional Data expressed with the Data Cube Vocabulary

The RDF Data Cube Vocabulary is intended for use in scenarios where multidimensional data need to be published as Linked Data, following the RDF paradigm. The vocabulary itself is a predefined schema of terms that describe multidimensional components. It is to be used in several DIACHRON pilot datasets and it provides a solid basis for an example of a default definition of Resources. The key classes of the vocabulary are represented by the following terms:

- **qb:Observation**: represents observation instances, that have fixed values for their dimensions, measures and attributes.
- **qb:Dataset**: represents a collection of observations, along with a defined multidimensional structure of components (dimensions, measures and attributes).
- **qb:DataStructureDefinition**: represents the defined structure that observations must conform to. For example, all observations must have one dimension to describe a reference area (e.g. country), one dimension to describe a reference period (e.g. fiscal year) and one measure for the actual measured value (e.g. projected net income).
- qb:Slice: represents a subset of the dataset with some fixed dimension values or other component property values.
- qb:DimensionProperty: represents dimensions as rdf properties
- qb:MeasureProperty: represents measures as rdf properties
- qb:AttributeProperty: represents attributes as rdf properties

In any case, the level of expressivity is dependent on the provided data and the desired modelling.

DIACHRON can make use of the defined semantics of this vocabulary and define Resources tailored for such datasets. In essence, it is taking advantage of a fixed set of semantics, provided that multidimensional datasets are expressed in this vocabulary. Therefore, in this case, it will be examined how Resources can be defaulted to more semantically rich concepts without any curation, as is the case in other examples of this document.

Observations are actually reified statements that measure one or more qualities of a property over one or more dimensions. They express multidimensional data efficiently and allow consumption and exploration. The actual URI of an observation is not necessarily meaningful or representative of the actual instantiation of cube components. In order to identify an observation in more concise, consistent and reversible ways, and more importantly, in a way that makes its evolution traceable, it is not enough to use its URI as an identifier. It is more reasonable to delve into the information it carries. For example, an observation can be identified across temporal versions of a dataset by gathering the dimension properties, dimension values and measure properties (and associated attributes) and then referring to the outcome as an observation Resource (in DIACHRON terms).

Given a dataset that measures unemployed people in EU member states per year for the past 10 years, the total number of observations will be the number of dimensions times the number of values the dimensions must have in the dataset, which in this example amounts to 28(member states) x 10(years) = 280 unique observations. The dataset, data structure definition, dimensions, measures and an example observation can be seen in the following figure.

```r
# The dataset
e:g:unemployment a qb:DataSet;
    qb:structure e:g:dsd-unemployment .

# The Data Structure Definition
e:g:dsd-unemployment a qb:DataStructureDefinition;

    # Dimensions
    qb:component [ qb:dimension e:g:refArea ];
    qb:component [ qb:dimension e:g:refPeriod ];

    # The measure
    qb:component [ qb:measure e:g:unemployedTotal ] .

    # Dimension definitions
e:g:refPeriod a rdf:Property, qb:DimensionProperty;
    rdfs:label "reference period"@en;
    rdfs:subPropertyOf sdmx:dimension:refPeriod;
    rdfs:range interval:Interval.
```
eg:refArea a rdf:Property, qb:DimensionProperty;
  rdfs:label "reference area"@en;
  rdfs:subPropertyOf sdmx-dimension:refArea;
  rdfs:range adm:geo:UnitaryAuthority.

# Measure definitions
eg:unemployedTotal a rdf:Property, qb:MeasureProperty;
  rdfs:label "Total number of unemployed people"@en;
  rdfs:subPropertyOf sdmx-measure:obsValue;
  rdfs:range xsd:integer .

# Example observation
eg:obs154 a qb:Observation ;
  qb:dataSet eg:unemployment ;
  eg:refArea eg:Greece ;
  eg:refPeriod eg:2000-2001 ;
  eg:unemployedTotal "500,000"^^xsd:integer .

A TTL-serialized example Data Structure Definition for the particular dataset, as defined in the Data Cube specification.

The example observation with URI eg:obs154 can be defined and retrieved by a subset of its components, instead of referring to its URI, which can change across versions. Specifically, it is defined by the fact that “total unemployment in Greece for the period 2000-2001 was 500,000”. What is measured in the above statement is completely dependent on the dimensions, as well as the dimension values of the measurement. The particular observation can be rebuilt if it is known that it is defined by a reference area, in particular Greece, a reference period, in particular 2000-2001, and a measure of total unemployment. The actual measured value is not part of the description definition itself, but the measure property can be used in order to compute the observation’s description. This becomes clearer upon inspecting the corresponding description query in SPARQL:

CONSTRUCT {
  ?observation ?p ?o
} WHERE {
  ?observation a qb:Observation ;
  qb:dataSet eg:unemployment ;
  eg:refArea eg:Greece ;
  eg:refPeriod eg:2000-2001 ;
  ?p ?o .
}

A SPARQL query constructing a graph that contains the description of observation eg:obs154, without referring to its URI.

This description definition is the result of the dataset initialization mechanism of DIACHRON, which uses the strict semantics of the Data Cube Vocabulary in order to identify the cube components (dimensions, measures etc.) associated with the dataset at hand and in turn build the appropriate definitions for
observations. The same procedure can be applied to all the terms that are defined in Data Cube. The procedure, in steps, is as follows:

1. For each incoming qb:dataset find its associated qb:DataStructureDefinition (hence dsd). This is a 1:1 relationship for valid cubes. Output: one dataset resource and one dsd resource and their associated definitions.
2. Given a dsd, find its components (dimensions, measures, attributes). Output: dimension resources, measure resources, attribute resources and their associated definitions. The resources created in steps (1) and (2) make up the schema part of the DIACHRON data model.
3. For each component find its distinct values that appear in the dataset and build a closed set of component-value combinations as prescribed by the dsd. Output: observation resources and their associated definitions.

After following the above steps the platform ends up having created Diachron Resources for all cube components of interest, which can then be tracked separately for changes, evolution and so on.

### 3.6.1 Ontological Data: Gene Expression Atlas

Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>dct</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
</tr>
<tr>
<td>atlas</td>
<td><a href="http://rdf.ebi.ac.uk/resource/atlas/">http://rdf.ebi.ac.uk/resource/atlas/</a></td>
</tr>
<tr>
<td>foaf</td>
<td><a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a></td>
</tr>
</tbody>
</table>

The Gene Expression Atlas will serve as an example instantiation of DIACHRON’s dataset model, and specifically how the various entities are used in order to model the data. Using the Gene Expression Atlas RDF Schema, the curator of the example decides that experiments (everything defined by the class atlas:Experiment) are Resources the changes of which she wants to track diachronically. She goes on to outline the definition of experiments, i.e. the change context they define, as follows:

An experiment ?x is described by the following sub graph (properties marked with an asterisk define 1:* relationships):

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>?x</td>
<td>dct:identifier</td>
<td>?identifier</td>
</tr>
<tr>
<td>?x</td>
<td>dct:description</td>
<td>?description</td>
</tr>
<tr>
<td>?x</td>
<td>atlas:isAbout</td>
<td>?organism</td>
</tr>
</tbody>
</table>

---

The description of an experiment according to the curator’s definition. The question mark notation is drawn from SPARQL. For more information refer to the specification document located at http://www.w3.org/TR/sparql11-overview/.

In other words, the curator decided that an experiment is described not only by its immediate first-level triples, but by an additional set of triples with different subjects than the experiment itself, namely the experiment’s assays and their samples/sample characteristics. This way, if a change occurs in one of the samples’ characteristics, the change is reflected back to the experiment as a change of the experiment itself.

Thus far, the curator has defined two things:

1. Experiment Resources are identified based on the fact that they belong to the class atlas:Experiment.
2. Experiment Resources are described at the triple level as follows:
   a. All triples that contain an experiment ?x as a subject.
   b. The atlas:Assay(s) that are related to the experiment, their respective atlas:Sample(s) and the samples’ sample characteristics, which are essentially reifications of properties.
   c. A query for creating the subgraph in the original dataset can be seen in the following figure and run in the Atlas endpoint. The returned result set is what triggers changes for the Resource. In other words, based on the provided definition, an experiment changes if any of its first-level triples is changed (added, deleted, modified etc.) **OR** if some of its assays, its assays’ samples and its assays’ samples’ characteristics change.

6 http://www.ebi.ac.uk/rdf/services/atlas/sparql The prefix atlas has to be changed to atlasterms in order to be consistent with the predefined prefixes in the endpoint. Furthermore, a limit should be given after the ending bracket of the WHERE clause.
An example of a SPARQL CONSTRUCT query the execution of which would yield the desired description in the Atlas SPARQL endpoint.

### 3.6.2 EXPERIMENT INSTANTIATION

For the purposes of this example, it is assumed that the dataset has been instantiated in DIACHRON. The experiment with identifier "E-GEOD-1085" that will be used in the present example can be dereferenced at its source simply by following its URI (http://rdf.ebi.ac.uk/resource/atlas/E-GEOD-1085). Briefly, it is described as follows:

> Transcription profiling of mouse BDC2.5 T cells from pancreas and pancreatic lymph nodes to understand what controls the aggressivity of the pancreatic infiltrate during type-I diabetes development.

The change context of “E-GEOD-1085” will be the result of the application of the curator’s definition, described in the previous section. In this case, the DIACHRON data model relevant to the experiment will be instantiated at time $t_m$ as follows.
**Figure 11. The context of “E-GEOD-1085” represented in the DIACHRON data model, at time $t_M$. Not all information is presented in the figure for brevity. However, it can be seen that there are Resources as subjects that are different than “E-GEOD-1085”, such as assays and samples.**

The Resource representing the experiment “E-GEOD-1085” can be seen in the left side of the figure.

At time $t_n$ a change occurs in the above context. Specifically, the sample characteristic `sample-characteristic-3BEF0279006F9DF8C896B3ED07F57D51` of sample `EDA27CC823B480C14A1226D9FF2B4A7B` changes its property value from “mus musculus” to some other animal (described by a literal in this case), which coincides with an already defined sample characteristic with ID `sample-characteristic-147CEC756B010B2B7DA10C5BF2C28B61`.

This means that the evaluation of the definition of Resource “E-GEOD-1085” will yield different results in $t_n$, triggering a change between the two versions. Again, it should be noted that the collection of records that are computed as the evaluation of a Resource’s definition does not necessarily contain triples that include “E-GEOD-1085” in the records. It includes, however, triples that in some primary way affect “E-
GEOD-1085” according to the curator. The following figure shows how DIACHRON captures changes and relates them to changes in records.

![Diagram](http://example.com/diagram.png)

**Figure 12. Going from Resources (http://rdfsources.com/resource/atlas/E-GEOD-1085) to evaluated contexts (depicted as documents with rows of records), to differences between evaluated contexts.**

### 3.6.3 Experimental Factor Ontology

The Experimental Factor Ontology (EFO) is developed at the EBI for the annotation of experimental variables in wide range of databases both inside and outside of the EBI. The ontology is an evolving document with monthly release cycles and has a number of dependencies both to other reference community developed ontologies, and also to the databases where it is used. Given the large number of dependencies it is important that the ontology developers and users can monitor changes in the ontology that may have potential implications to downstream data analysis processes.

The DIACHRON use case deliverable outlines several components of the ontology where change tracking is important. This document outlines how some of these components fit into the DIACHRON data model.
The life sciences have a wide range of specialised ontologies for certain domains such as anatomy, cell types, and disease. Where possible, EFO will reuse a term (or concept) from an external ontology rather than define a new term. Reuse is simply done by reusing the external term IRI in EFO. In some cases no relevant term is available in the external ontologies, in these cases EFO will mint an internal EFO id for the term. Over time as the external ontologies mature and evolve, the EFO developers may deprecate an EFO term IRI in favour of an external term. Reusing external terms is preferable as it facilitates more integration across databases when common terminologies are used.

An example of the typical scenario is that database A requests that EFO creates a term for “liver”. After searching the external reference ontologies no suitable concept can be found, so a new EFO term is created with an IRI of efo:EFO_0000887. After a year, the UBERON ontology, a reference ontology widely used to describe anatomy introduces a new term that represents the liver concept. At this point the EFO developers choose to obsolete the old EFO liver term (efo:EFO_0000887) and replace it with the term provided by UBERON (obo:UBERON_0002107). This series of events should then cause a cascade of changes to the databases that use EFO to update their annotation to the new preferred term id.

In this scenario the change is focused around an OWL class. At \(t_m\) the EFO class described the term “liver”. At \(t_n\), the class has been obsoleted in favour of an external class for “liver” from the UBERON ontology.

![Figure 13. Instantiation of the data model for efo:EFO_0000887 at times \(t_m\) and \(t_n\).](image)

### 3.6.4 ZOOMA

Zoo is the EBIs ontology annotation platform. This is where annotations between data and ontology terms are stored and curated. A typical scenario is that biological samples from one database, such as the Gene Expression Atlas (GXA), are semantically annotated with an ontology term. That can be used
further downstream to support querying and analysis of the data. This can be demonstrated with a real example from the database in order to show how the DIACHRON model captures the data. The assumption is that a curator wants to annotate a sample from experiment E-GEOD-1085. The sample is described as being a liver sample, and the curator wants to annotate this free text description with a class from the EFO ontology. At t1, an annotation is created in Zooma (zoomaannotations:_a1) that connects the sample (atlas:E-GEOD-1085#13.07-sample-characteristic-_1), which is the target of the annotation to two annotation bodies, the property value “liver”, and the semantic tag (efo:EFO_0000887). The annotation schema for Zooma comes from the Open Annotations Model that defines the relations between annotation, target and body.

**Figure 14. Instantiation of the data model for annotation _a1 at time t_m**
It can now be seen how the EFO ontology is connected to the data. Considering the scenario when the EFO class for livers becomes obsoleted and the new UBERON class is added to EFO, the accompanying Zooma annotation must also be updated. At t2 now it is shown how the Zooma annotation has been updated in its corresponding DIACHRON model. A new Zooma annotation is created (zoomaannotations:_a2) that is connected to the old annotation by the zooma:replaces property. The body of this new annotation has now been updated to use the new UBERON term (UBERON:UBERON_0002107).

3.7 SUMMARY

The figure shows a 2x2 grid of example instantiations of datasets in various degrees of details. The vertical dimension divides the grid into time-agnostic space and time-aware space, which provides a visual and conceptual means of differentiating between entities that evolve or are otherwise affected by time on the bottom half of the grid, and entities that are not affected by temporal progression in the top
The horizontal dimension divides the grid into (raw) data space and curated information space. The difference between these two lies in the fact that data from various datasets reside in the data space and do not hold more semantic information that what is already attributed to them at the time of deployment (as is the case with information that can be the result of curation), whereas in the curated information space one can find Resources that are defined with a non-generic mechanism such as the ones previously discussed. In the data space there are dataset instantiations with their records at different times. Dataset D1 is shown instantiated at times \( t_m \) and \( t_n \). On the top half diachronic entities can be seen.

The difference between a diachronic Resource (top right) and a non-diachronic Resource (bottom right) lies in the fact that the former is a time-independent abstraction of the latter, which is an instantiation of the former on a particular moment in time. A diachronic Resource can be connected with more than one non-diachronic Resource, as the same conceptual Resource can be found in many different temporal versions of the same dataset.
The next figure depicts the basic conceptual objects of Diachron. These concepts exist both in the pilot’s application model but also in Diachron. The system will execute its functionality in its own defined dataset model and changes representation model. A set of explicit mapping (transformations) will have to be defined in order to bridge the basic concepts as they are modeled in the pilots’ applications with the DIACHRON models, for example the mapping of the data model a pilot uses to the data model DIACHRON operates.

The definition and implementation of these mappings play the role of the platform’s configuration and customization and is dependent on each pilot’s modeling and technology principles. Therefore the platform configuration has to be made for each pilot separately. From the definition of the mappings derives a series of inferred translations (greyed boxes) that map the remaining concepts that do not have to be explicitly defined during the platform’s customization. The resulting scheme allows the pilots to define the Diachron concepts (Complex changes and Resources) during the operational phase of the system by using their own modeling concepts and principles. Additionally it provides a way for the pilots to retrieve data (Datasets, Resources and Changes) represented in models similar to the ones that they use already and thus by minimizing the effort of integrating the system’s functionality with their other systems, maximizes the potential benefits offered by DIACHRON.
FIGURE 18. BASIC MODELING CONCEPTS IN DIACHRON AND THEIR RELATIONSHIPS
4 HOW THE DESIGN MEETS THE REQUIREMENTS

The functional requirements of the DIACHRON system have been elicited and presented in deliverable D1.1 “Requirements of the DIACHRON Use Cases”. The following paragraphs and figures provide the correlation between the functional requirements and the architecture of the system in the form of cross functional diagrams that try to decompose the functional requirement to simpler functions that can be assigned to the various parts of the system as they were depicted in Chapter 2 of this document.

In following table, the first two columns are taken from “Table 1. Correlation between functional requirements, use cases and the system's added value” of the deliverable D1.1 and they are now associated with the corresponding workflow representations.

<table>
<thead>
<tr>
<th>DoW requirements</th>
<th>Functional requirements</th>
<th>Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-version Archiving</td>
<td>Add a dataset</td>
<td>Diagram 1</td>
</tr>
<tr>
<td></td>
<td>Update a dataset</td>
<td>Diagram 2</td>
</tr>
<tr>
<td></td>
<td>Delete a dataset</td>
<td>Not depicted</td>
</tr>
<tr>
<td>Longitudinal querying</td>
<td>Querying</td>
<td>Diagram 3</td>
</tr>
<tr>
<td>Diachronic citations</td>
<td>Diachronic Citations</td>
<td>Although not depicted explicitly in the diagrams, these requirements are part of diagrams 1 and 2. Further research that will be done during WP2 will further elaborate the correlation of this requirement with the architecture of the system.</td>
</tr>
<tr>
<td>Temporal and provenance</td>
<td>Annotations</td>
<td></td>
</tr>
<tr>
<td>annotations</td>
<td>Change recognition and propagation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change detection</td>
<td>This requirement is present as sub flow in Diagrams 2, 4 and 5</td>
</tr>
<tr>
<td></td>
<td>Change Monitoring</td>
<td>Diagram 4</td>
</tr>
<tr>
<td></td>
<td>Propagation of changes</td>
<td>Diagram 5</td>
</tr>
<tr>
<td>Cleaning and repairing</td>
<td>Validation – Repairing</td>
<td>Diagram 6</td>
</tr>
<tr>
<td>Ranking datasets</td>
<td>Quality assessment</td>
<td>Diagram 7</td>
</tr>
<tr>
<td></td>
<td>Dataset ranking</td>
<td>No depicted, trivial case using Diagram 7 workflow</td>
</tr>
<tr>
<td>Crawling datasets</td>
<td>Crawling</td>
<td>Diagram 8</td>
</tr>
</tbody>
</table>
The first Diagram depicts the insertion of a new dataset to the DIACHRON system. The dataset is retrieved from the World Wide Web and the workflow includes two core services of the platform layer.

Diagram 2 depicts the update of an existing dataset a result of the submission of a new a new version of the dataset by the user. This composition requires the use again of two core services.
Diagram 3 depicts a simple workflow for executing a query for a Dataset from the system.
Diagram 4 present the composite workflow of monitoring a dataset for changes. The dataset is downloaded from the World Wide Web, change detection is performed and the user gets notified when changes are found. The whole task is scheduled for periodic re-execution and a process could use it for the estimation of the time interval between new versions of the dataset.
Figure 22. Diagram 4. Monitoring for changes on through scheduling a task
Diagram 5 depicts the workflow executed when a temporal query involves data which their schema has changed but there is a way to apply these changes to the previous data so as to provide to the user a continuous temporal data series.

**Diagram 5. Propagation of schema changes to previous dataset versions as part of a temporal query execution**

Diagram 6 presents the workflow for validating and repairing a Dataset by using two sets of rules, one for validation and one for guiding the repairing process.
Diagram 7 depicts the quality assessment process of a dataset.
Finally, Diagram 8 presents a process to create a dataset through crawling on various user provided sources. The process also involves the quality assessment of the newly created dataset.
Creation of a dataset through crawling

<table>
<thead>
<tr>
<th>Pilot</th>
<th>Integration Layer</th>
<th>Platform Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>User records crawling sources and crawling topic</td>
<td>Crawling Service initiates crawling task</td>
<td>Crawling Module retrieves data</td>
</tr>
<tr>
<td></td>
<td>Crawling Service evaluates the results</td>
<td>Crawling result set</td>
</tr>
<tr>
<td>User curates the results</td>
<td></td>
<td>Quality module calculates quality indicators</td>
</tr>
<tr>
<td></td>
<td>Crawling Service creates new dataset for the results</td>
<td>Quality assessment results</td>
</tr>
<tr>
<td></td>
<td>Diachron Dataset</td>
<td>Data Access Module stores Dataset to Archive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Archive</td>
</tr>
</tbody>
</table>

**Figure 26. Diagram 8. Creation of dataset through crawling**
5  CONCLUSION

This document provides the conceptual model of the DIACHRON system and leaves the detailed design of the system, service contracts and messaging workflows, to be carried in task 6.1 “DIACHRON Development Environment and Detailed Design” and presented in deliverable “Development Environment and Detailed Design of DIACHRON platform” due in month 13 (M13) of the project.

During the phase of the functional requirements (deliverable D1.1) the four pilot partners presented their respective data models which were categorized in two major categories, namely multi-dimensional data and ontological data. The pilots also use various serialization formats and persistence storage technologies, varying from DSPL and custom OLAP implementations in RDBMS and Object Oriented Databases to RDF(S) and RDF triple stores.

The design approach to try to bring the DIACHRON functionality in the pilots’ data models directly was considered but no adopted. This would require the re-implementation of the DIACHRON functionality in more than three versions. Additionally, the research tasks would have to address not only the actual research challenges and objectives but also to the specificities of each case.

This would impose a high risk of having to compromise on the research objectives of the project. Moreover the integration task would be more dependent on each pilot scenario and more dependent on each research partner’s work. This would probably have result in implementing three significantly different versions of the system, one for each use case scenario, which, in addition to not being completely aligned with the project’s objectives as described in the DoW document, also would have imposed a great risk to the project.

In order to abstract the functionality from the specific data models of the pilots, it was decided to introduce the DIACHRON Dataset Model, which provides a common ground where the functionality can be designed and implemented, and reduces the integration risk to the transformation/mapping of the pilots’ data models to the DIACHRON Dataset Model. This risk is further minimized by the adoption of the service oriented architecture model, which provides a flexible ground for the partners to proceed with their assignments since it minimizes the dependencies between their research and implementation tasks.
6 REFERENCES

[4] SPARQL. http://www.w3.org/TR/sparql11-overview/