SEVENTH FRAMEWORK PROGRAMME

CAPACITIES

Research Infrastructures

INFRA-2009-1 Research Infrastructures

OpenAIREplus

Grant Agreement 283595

"2nd-Generation Open Access Infrastructure for Research in Europe

OpenAIREplus"

OpenAIRE Data Model Specification

Deliverable Code: D6.1

Document Description

Project	
Title:	OpenAIREplus, 2nd Generation Open Access Infrastructure for Research in Europe
Start date:	1st December 2011
Call/Instrument:	INFRA-2011-1.2.2
Grant Agreement:	**283595**

Document	
Deliverable number:	D6.1
Deliverable title:	OpenAIRE Data Model Specification
Contractual Date of Delivery:	30th of April, 2012
Actual Date of Delivery:	21st of April 2012
Editor(s):	Paolo Manghi
Author(s):	Paolo Manghi, Marko Mikulicic, Claudio Atzori
Reviewer(s):	Natalia Manola, Katerina Iatropulou, Antonis Lempesis, Jochen Schirrwagen, Mathias Loesch, Mateusz Kobos, Linda Reijnhoudt, Eko Indarto, Arjan Hogenaar, Wilko Steinhoff, Lars Nielsen
Participant(s):	Nikos Houssos, Keith Jeffery, Brigitte Joerg, Marko Mikulicic
Workpackage:	WP6
Workpackage title:	OpenAIREplus data model and content management services
Workpackage leader:	CNR
Workpackage participants:	NKUA, UNIBI, DTU/DataCite, CERN, UNIWARSAW, EKT-NHF/EuroCRIS, EBI-EMBL, KNAW-DNAS, STFC-BADC
Distribution:	Public
Nature:	Deliverable
Version/Revision:	2.0
Draft/Final:	Final
Total number of pages:	

(including cover)	
File name:	
Key words:	data model

Disclaimer

This document contains description of the OpenAIREplus project findings, work and products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium head for approval.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the OPENAIRE consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu.int/)

OpenAIREplus is a project funded by the European Union

Table of Contents

Table of Figures

Summary

The OpenAIREplus web site will offer functionalities for administrators, anonymous and registered users to manage an Information Space of publications, together with their connections with funding projects (from the EC and national agencies) and research datasets. The aim of this document is to describe the conceived structure and semantics of this Information Space, i.e., the *OpenAIREplus data model*, by providing an abstract definition of its main entities and the relationships between them.

In this definitional process, the interaction with the EuroCRIS initiatives, several scientific institutes (i.e., KNAW-DANS, EBI-EMBL, BADC) as well as inspiration from DataCite and LinkedData play an important role in the specification of project data, i.e., how project data should be described, stored and exported in OpenAIRE, dataset metadata, how datasets should be described, and in the specification of how such interconnected entities can be made available and consumable by third-party systems.

The data model will be subject to changes in the future and therefore result in further versions. Such changes will be described in the following Section, in order to summarize to the reader the differences form the previous versions.

Log of Changes

Deliverable Version	**Date**	**Changes** (description, section and pages)
1.2	27.06.12	Ordering of authors: inserted new attributes in linked entity Person_Results
1.3	18.09.2012	Added property Name Space Prefix to Data Source entities Added EntityRegistry entity to entity table Removed Discipline entity (use inly Subject entity instead) Addition of Identity entity in the E-R diagram and entity table Addition of Title and Date entities in the entity table (fixed direct and inverse relationships)
2.0	04.10.2012	Updated section 7 to shape up the HBase solution we have developed (Cassandra was removed as an option)
2.0	26.10.2012	Associating licenses and classType classScheme (i.e., article, pre-prints) to instances rather than results (in order to support merges of different versions of files for the same result). Adding journal attribute to class Publication. Moving publisher attribute from Publicaiton to Result class.
2.0	7.11.2012	Added the notion of *OpenAIREcompatibility* to the DataSource entity Removed TypeClass and Scheme from Dataset entity: they will be described as instanceTypeClass and Scheme in the relative Instance entity Added contractTypeClass/Scheme to Project entity Added EC_International Organization flag in Organization entity, it was missing DataSource, Instance, Identity entities: attribute persistentIdentifer renamed as PID Adding the flag *ForMining* (administrative fields): the flag states that the piece of information (be it an object or a relationship) should not be fed to the index but to the Information Inference Service only

1 Scenario

The OpenAIREplus web site will offer functionalities for administrators, anonymous and registered users to manage an Information Space of open access and non-open access publications, together with their connections to funding projects (from the EC and national agencies) and research datasets. In particular:

- *Anonymous users* will be able to search and consult the Space;
- *Registered users* will be able to:
    - Give feedbacks to improve the quality of the Space;
    - Claim publications or datasets into the Information Space;
- *Administrators (data curators)* will have full access and rights to such Space to:
    - Collect data from data sources;
    - Edit properties of publications, datasets, persons, etc in the Information Space;
    - Validate/invalidate insertions/deletions/updates suggested by registered users or by automatic inference processes.

1.1 Data Model: Information Space Entities and Relationships

In our reasoning we generalize the concept of datasets and publications to that of **project result**, so as to be able of including further kinds of research outputs. OpenAIRE initially proposes two kinds of results: **datasets** (e.g., experimental data, software products) and **publications**. But others can be added in the future (e.g., patents). Besides, project results are always associated to one or more *instances* of the results, in the sense that different "physical representations" of the same result may exist. For example, the same publication may be kept in two different repositories, both exposing the payload file (e.g., PDF) at different internet locations (URLs). Morover, an instance of a result is represented as a combination of one or more *web resources* relative to the sub-parts of the result and of the internet data sources from which such resources are made available.[1]

Similarly, we extend the notion of authors of publications or datasets to that of **persons**, to include in the same set people connected to project fundings or organizations. For example "authorship" relationships between results and persons, which represent the fact that a given *person* has (co-)authored a given *result* while being affiliated with a given *organization*.

Organizations include companies, research centers or institutions involved as project partners or as responsible of operating data sources. Information about *organizations* will be initially collected from CORDA and CRIS systems, as being related to projects, or be ingested by users, for example to complete authorships information in the database.

Of crucial interest to OpenAIREplus is also the identification of the **funding programmes** which co-funded the **projects** that have led to a given result:

[1] The purpose of the project result-instance-web resource model is to capture a list of internet pointers relevant to the project result and not that of capturing the compound object structure which some results may have. If a project result is a compound object (e.g. ORE aggregation), its instances will likely be associated to web resources embodying its compound object nature (e.g., ORE resource maps).

- EC FP7 programme projects data will be fetched from the authoritative EC CORDA database, together with the *organizations* or *persons* which are *participants* of such projects.
- Data relative to National funding schemes and relative projects will be instead fetched from CRIS systems, together with other entities which may be typically kept within a CRIS system (e.g., publications, datasets, projects, organizations, people, etc).

Finally, OpenAIRE entity instances are created out of data collected from various **data sources** of different kinds, such as publication repositories, dataset archives, CRIS systems, etc. Data sources export **information packages** (e.g., XML records, HTTP responses, RDF data) which may contain information on one or more of such entities and possibly relationships between them. It is important, once each piece of information is extracted from such packages and inserted into the information space as an entity, for such pieces to be linked to the originating data source. This is to give visibility to the data source, but also to enable the reconstruction of very the same piece of information if problems arise. Inititially, information relative to repositories will be collected by the OpenDOAR data source, which will act as main entity registry for (literature) repositories in Europe, but other data sources may join OpenAIRE in the future. The same centralized directory is instead not available for dataset archives and CRIS systems, whose managers/administrators will have to provide to OpenAIREplus while registering their data sources.

Entity instances, relative to persons, projects, organizations, results, and data sources will be instantiated from information packages collected, inferred, feed-backed, claimed from two main collection workflows: automated fetching from registered data sources and expert-provided information.

- *OpenAIREplus* compliant data sources: these include all data providers willing to authoritatively provide content to OpenAIRE,
  - *CRIS systems*: CERIF compatible data sources;
  - *Repositories*: institutional and thematic;
  - *Dataset Archives*: intended here as dataset providers from several subject areas;
  - *Data Source Aggregators*: intended here as systems federating several data sources of the same kind (e.g., repository federations);
  - *Entity Registries*: intended here as external data sources whose purpose is to provide a unique identity and reference for given entities (e.g., OpenDOAR for repositories, ORCID for persons)
- *OpenAIREplus expert-validated entity pool*: entities may reach the information space through human-driven workflows, which deliver entities into an expert-validated entity pool. Such a pool acts like a special data source, from which experts of various kinds (e.g., authors of results, project coordinators, data curators) can inject authoritative information into the OpenAIRE information space:
  - Authors (or others on their behalf) can "claim publications" into OpenAIREplus through the portal by:
    - Providing CrossREF DOIs and enriching them with project and license information;

        - Searching publication metadata through external information spaces (e.g., BASE search engine) and selecting/enriching (project and license information) the ones they claim to be related to a project;
    - Registered users can provide end-user feedbacks through the portal, to suggest data corrections or enrichments trough "editing" actions to be validated by OpenAIRE data curators;
    - EC project coordinators can confirm relationships between publications and EC projects automatically inferred by dedicated services;
    - Data curators can validate guesses made by end-users through feedbacks or validate *inference actions* (see below) to make them persistent;
    - Data curators can perform edit actions, such as entity addition, removal, or updates.

Such data sources remove, delete, update in the OpenAIRE information space information about one or more of the entities, as well as relationships between them. For example, some archives provide dataset metadata which also includes links to publications relevant for the dataset or vice versa. Such cross-entity and cross-sources data integration brings in data inference issues, which have mainly to do with information absence, duplication, and versioning (intended as replicas of the same entity). For example, many relationships (instances of) may not be available from data sources or may not be considered at all as valuable (e.g., not of interest to the specific research domain). Moreover, the same publication metadata may be collected from several resources, including repositories or CRIS systems. These issues will push into the data model a number of entities, properties, and relationships whose aim is to deliver to data curators the tools to maintain a clean, uniform, and consistent information space.

1.2 Extending Data Model to Support Data Inference

As planned in the DoW the OpenAIRE infrastructure will feature a number of services capable of curating the information space by disambiguating and enriching its entities, namely:

- *Duplicate inference*: different records representing the same entity (e.g., Result, Persons, Organizations, Projects) may be merged to disambiguate the information space.
- *Relationship inference*: new relationships between entities (e.g., citations, similarity semantics) may be inferred and added to the information space.
- *Attribute inference*: attribute values, such as titles or author names, will be inferred from the original full-text or digital files.

The OpenAIRE data model will therefore require to be extended in order to capture the entity information needed to cope with the distinction between **original entities**, which are either collected from data sources or provided by experts, and **inferred entities**, which can be instead re-calculated any time starting from the former set of entities and therfore have a "lower level of trust". As shown in Figure 1, end-users and application access an information space which consists of the pool of original entities as collected from data sources and provided by experts, deduplicated and enriched by the data inference process through a layer of inferred information. We shall see that in some cases, inferred entities may become original entities, when an expert validates them and inserts them into the expert-validated entity pool. In this case, such entities enter the information space as original entities and therefore become input to the data inference process.

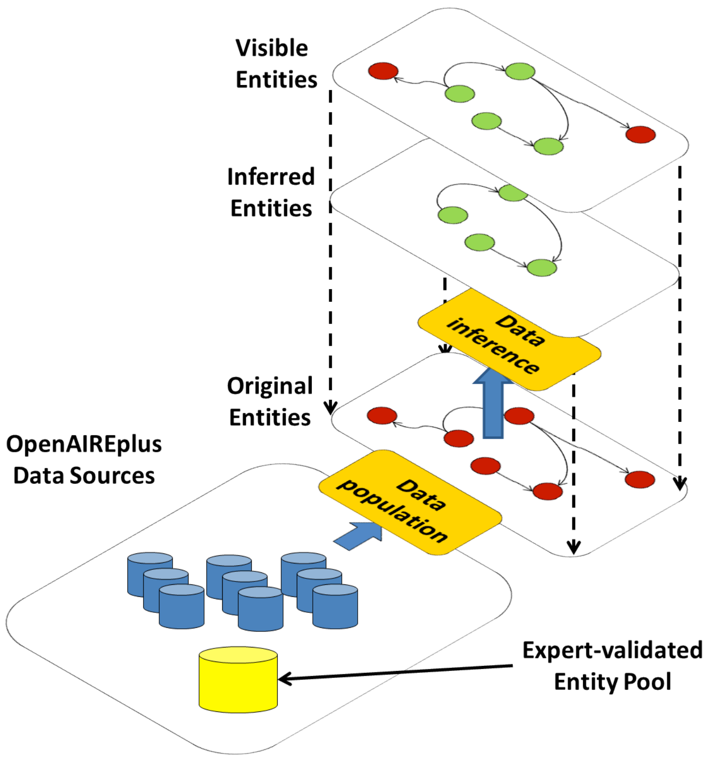

Figure 1 – Entity data flow

1.3 Outline

In the following, Section 2 provides a detailed description of the main entities that come into play by providing the relative Entity Relationship model. Section 3 describes the workflow of data population that is how data is collected from data sources and and ingested into the information space according to the data model. Section 3 describes the issues encountered when introducing the concept of inferred entities into the data model and presents the entailed model changes. Finally, Section 5 describes the changes to be applied to the data model in order to cope with data management, from the an administrative perspective.

The data model will be subject to changes in the future and therefore result in further versions.

2 Main data model entities

This section provides a detailed description of the OpenAIREplus data model. Since the data model marries the notion of "semantic layer" as proposed by the CERIF model,[1] we shall first describe its abstraction mechanisms and then provide the details of OpenAIRE entities, their properties and their relationships.

2.1 CERIF Semantic Layer

According to this notion, (*i*) "horizontal" classification of entities (e.g., by vocabularies of terms) is not modeled through properties associated to given controlled vocabularies and (*ii*) semantic relationships between entities are not modeled by adding dedicated relationships. In both cases, CERIF introduces a flexible modeling mechanism which allows injecting classification semantics into "semantics-agnostic" entities and relationships. The mechanism is obtained by introducing two entities *Schemes* and *Classes* such that (see Figure 2):

Class A *Class* represents one term of a classification, e.g., vocabulary, taxonomy. As such it is characterized by the following properties: a *Code*, which represents the persistent identifier associated to the term (e.g., real-world classifications, such as ISO vocabularies for countries, have a standard identification code for terms), a *name*, an *acronym*, a *description*, a *StartDate*, and an *EndDate*. A *Class* is characterized by the following relationships with other entities:

(i) *scheme*: the set of *Scheme*s to which the *Class* belongs to (typically a *Class* describes a term which belongs to one *Scheme*, but there are cases where the same term can be shared across several *Schemes*);

(ii) *related Classes* (inverse of relationships *Class1* and *Class2* from *Classes_Classes* entities): the *Classes* related with the *Class* through the relationships entities *Classes_Classes*; the semantics of these associations is specified in the Classes_Classes entities (e.g., "partOf", "parent", "child");

Scheme A *Scheme* identifies the existence of a classification scheme, which is modeled as a set of interrelated *Class* entities. A *Scheme* is characterized by the following properties: a *Code*, which represents the persistent identifier associated to the *Scheme* (e.g., real-world schemes, such as taxonomies, may be have a standard identification code), a *name*, an *acronym*, a *description*, a *StartDate*, and an *EndDate*. A *Scheme* is characterized by the following relationships:

(i) *related Classes* (inverse of the relationship *schemes* of *Class* entities): the *Classes* associated to the *Scheme*;

(ii) *entryPoints*: the *Classes* at the first level of the *Scheme*.

Class_Class Such entities represent associations between different terms (*Classes*), they are characterized by the following properties: a *StartDate* and an *EndDate*. They are characterized by the following relationships:

[1] CERIF data model: http://www.eurocris.org/Index.php?page=CERIFreleases&t=1

(iii) *Class1* and *Class2*: which identify the two *Class* entities to be associated;

(iv) *semanticsClass* and *semanticsScheme*: which respectively specify the semantics of this association through a given classification scheme.

Scheme_Scheme Such entities represent associations between different Schemes, they are characterized by the following properties: a *StartDate* and an *EndDate*. They are characterized by the following relationships:

(i) *Scheme1* and Scheme*2*: which identify the two *Scheme* entities to be associated;

(ii) *semanticsClass* and *semanticsScheme*: which respectively specify the semantics of this association through a given classification scheme.

The mechanism allows the adoption of new classification schemes of arbitrary complexity: flat structures, such as vocabularies of terms for country (ISO 3166-1), tree structures, such as the EC FP7 funding scheme (http://cordis.europa.eu/fp7) or the WoRMS taxonomy of marine species (http://www.marinespecies.org), and graph structures such as the gene ontology (http://www.geneontology.org).

Table 1 – Properties and relationships for Class, Scheme, Classes_Classes, and Schemes_Schemes entities

Class • code • name • acronym (optional) • description (optional) • startDate • endDate ➔ class1^{-1} (0 or N Class_Class) ➔ class2^{-1} (0 or N Class_Class) ➔ schemes (1 or N Scheme) *Notes:* For simplicity, inverse relationships with other entities are not reported	**Scheme** • code • name • acronym • description • startDate • endDate ➔ entryPoints (0 or N Class) ➔ schemes^{-1} (1 or N Class) ➔ scheme1^{-1} (0 or N Scheme_Scheme) ➔ scheme2^{-1} (0 or N Scheme_Scheme) *Notes:* For simplicity, inverse relationships with other entities are not reported
Class_Class • startDate • endDate ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) ➔ class1 (1 Class) ➔ class2 (1 Class)	**Scheme_Scheme** • startDate • endDate ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) ➔ scheme1 (1 Scheme) ➔ scheme2 (1 Scheme)

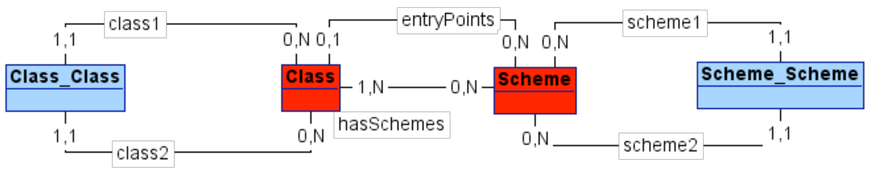

Figure 2 – E-R model: semantic layer entities

The OpenAIREplus data model introduces semantic-agnostic relationships between *publications-results, publications-publications, datasets-datasets, publications-datasets, and organizations-projects*. Their intended semantics will be injected thanks to a Class entity of a Scheme entity. Similarly, whenever entities need to be classified based in a property value (e.g., nationality of a person), property and values are modeled by an association to a *Class* (e.g., *nationalityClass*) and one to the relative *Scheme* (e.g., *nationalityScheme*). The benefit of the approach is that applications can be written in such a way they cope with the dynamic addition, removal, or deletion of *Classes* and *Schemes*. This is indeed the case in OpenAIREplus, where the intended entities and relationships will be subject to changes based on the results of Joint Research Activities.

2.2 Entities description

The entities in the data model can be grouped in the following way:

- *Main* entities: the entities whose information is continuously and incrementally fed to the information space; namely Result (Publication and Dataset), Person, Organization, DataSource (Repository, Dataset Archive, CRIS, Aggregator, Entity Registry), Projects;
- *Structural* entities: the entities added to the model to represent complex information about an entity; namely Instances, WebResources, Titles, Dates, Identities, and Subjects;
- *Static* entities: entities whose content is inserted in the information space at some point in time; namely Funding, Class, and Scheme;
- *Linked* entities (CERIF notation): relationship entities, used to connect in a semantic-agnostic way two or more main entities; namely, those denoted by an Entity1_Entity2 notation.

2.2.1 Main Entities

Main entities are characterized by a provenance relationship *collectedFrom*, which indicates the DataSource entity (if it exists) from which entity information was collected. The conceptual representation of the schema is illustrated in Figure 3, where main entities, static entities and linked entities are represented, in Figure 4, where Result entities and their sourroundings are represented, and in Figure 5, which groups all entities involved in the *collectedFrom* relationship.

Result A *Result* is here intended as the (metadata) description/representation of a scientific output (possibly) resulting out of one or more projects. A result is characterized by the following properties: a *date of acceptance*, a *publisher* (optional), a *description* (optional), and an *embargo end date* (empty if the *licenseClass* does not imply an embargo). A Result is characterized by the following relationships with other entities:

(i) *PIDS* (optional): which is the list of unique and persistent identifiers used to identify the result together with the relative identification agency, e.g., EPIC, CrossRef, DataCite;

(ii) *Titles* (mandatory): the *Titles* of the Result, represented as *Class* entities of a Scheme entity, e.g., original, alternative, subtitle, etc.

(iii) *Creators* (mandatory): the creators of the *Result*, which are *Person* entities connected to the result through relative *Person_Result* entities;

(iv) *Instances* (mandatory): the *Instances* of the *Result*, which represent the locations (*DataSource* entities) where the *Result* files (*web resources* entities, e.g., DOIs) can be found;

(v) *languageClass* and *languageScheme* (optional): the language used in the description or body of the Result, specified according to a given classification of languages, respectively described as a *Class* entity and a *Scheme* entity;

(vi) *collectedFrom* (mandatory): the *DataSource* entity from which the information relative to the Result entity was collected;

(vii) *relevantDates* (optional): a list of dates relevant to the Result;

(viii) *fundingProjects* (optional): the *Projects* which co-funded the research underlying the *Result*;

(ix) *subjects* (optional): the scientific disciplines (represented as *Class* entities of a Scheme entity) covered by the *Result;*

Note that the difference betweem *collectedFrom* and *hostedBy* is introduced to encode the peculiar notion of *Aggregation* data sources, whose entities are obtained by federating a set of DataSource entities. In OpenAIRE aggregators are DataSources from which entities are "collected", while the DataSources they aggregate "host" instead such entities. As such, Aggregators differ from the data sources they aggregate, but play an equally important role in delivering the entities to OpenAIRE, and should therefore be given visibility. In the case of other DataSources, e.g., repositories, collectedFrom and hostedBy refer to the same DataSource.

Dataset A *Dataset* is a *Result* further characterized by the following optional properties (ref. DataCite initiative v2.2): *resource type, size, format, version, last metadata update,* and *metadata version number*. A *Dataset* is characterized by the following relationships:

(i) *the set of Datasets it is related with* (inverse of relationships *dataset1* and *dataset2* of *Dataset_Dataset* entities): the semantics of such relationships is injected in *Dataset_Dataset* entities through a *semanticsClass* and *semanticsScheme* relationships);

(ii) *the set of publications it is related with* (inverse of relationships *dataset* of *Publication_Dataset* entities): the semantics of such relationships is injected in *Publication_Dataset* entities through a *semanticsClass* and *semanticssScheme* relationships);

(iii) *resourceTypeClass* and *resourceTypeScheme*: the type of the *Dataset* according to a given classification of *Dataset* types, respectively described as a *Class* entity and a *Scheme* entity; The resourcetypeScheme "DataCite_resource" defines the

following typeClass values: Collection, Dataset, Event, Film, Image, InteractiveResource, Model, PhysicalObject, Service, Software, Sound, Text.

Publication A *Publication* is a *Result* further characterized by the following properties: *journal*. A *Publication* is characterized by the following relationships:

(i) *the set of publications it is related with* (inverse of relationships *publication1* and *publication2* of *publication_publication* entities): the semantics of such relationships is injected in *publications_publications* entities through a *semanticsClass* and *semanticsScheme* relationships);
(ii) *the set of datasets it is related with* (inverse of relationships *Publication* of *Publication_Dataset* entities): the semantics of such relationships is injected in *Publication_Dataset* entities through a *semanticsClass* and *semanticssScheme* relationships)

Person A Person is characterized by the following properties: *firstName*, *secondNames*, *infixName*, *fax, email, phone*, *title*, and *gender*. A Person is also characterized by the following relationships:

(i) *PIDS* (optional): which is the list of unique and persistent identifiers used to identify the person together with the relative identification agency, e.g., ORCID;
(ii) *creations* (optional): the creations of the *Persons,* which are *Results* entities connected to the result through relative *Person_Result* entities;
(iii) the participations of the *Person* to *Projects* (optional): inverse of *contactPerson* relationships of *Participants* entities;
(iv) nationalityClass and *nationalitycheme*: the nationality of the *Persons* according to a given classification of nationalities, respectively described as a *Class* entity and a *Scheme* entity.
(v) *collectedFrom*: the DataSource entity from which the information relative to the Person entity was collected;

Project A *Project* is characterized by the following properties: an *persistent identifier* (which is the unique and persistent identifier used to identify the project by its funding agency, e.g., grant agreement number for the EC), a *title*, an *acronym*, a *web site* (e.g., in the case of EC projects the project page at CORDIS), a *start_date*, an *end_date*, a *duration* (derived by start and end dates), a *project call identifier*, and a list of *keywords*. It also features a special flag *EC_SC39*, indicating whether or not an EC project is subjected to clause 39. A Project is also characterized by the following relationships:

(i) *PIDS* (optional): which is the list of unique and persistent identifiers used to identify the project across several institutions, together with the relative identification agency;
(ii) *participants*: which is the set of Participants participating to the Project;
(iii) the set of Results whose research was co-funded by the Project (inverse of the relationship *fundingProjects* of the *Result* entities);
(iv) fundedBy: the set of *Fundings* entities indicating which grants co-funded the *Project*;
(v) *collectedFrom*: the DataSource entity from which the information relative to the Project entity was collected;

(vi) *contractTypeClass* and *contractTypeScheme*: the contract type of the *Project* according to a given classification of contracts, respectively described as a *Class* entity and a *Scheme* entity.

Organization An Organization is characterized by the following properties: an *original identifier* (persistent identifier, if any, made available by the source providing organization information), a *legal short name*, a *legal name*, an *URL of its web site*, and an *URL of the logo* (if available). An organization is further characterized by a number of flags, which are always present in the case of *Organization* information provided by the EC: *legal body, legal person, non profit, research organization, higher education, international organization with Eur interests, international organization, enterprise, SME validated* flag, and *NUTS code* (Nomenclature of Territorial Units for Statistics). An *Organization* is characterized by the following relationships:

(i) *PIDS* (optional): which is the list of unique and persistent identifiers used to identify the organization across several identification agencies;
(ii) the set of *Persons* which have created a *Result* while affiliated with the *Organization* (the inverse of the relationship *creatorAffiliation* of *Person_Result*);
(iii) *dataSources*: the *DataSource* entities under the responsibility of the *Organization*;
(iv) the participations of the *Organization* to *Projects* (inverse of *respOrganization* relationships of *Organization_Project* entities);
(v) *countryClass* and *countryScheme*: the country of the *Organization* according to a given classification of countries, respectively described as a *Class* entity and a *Scheme* entity;
(vi) *collectedFrom*: the DataSource entity from which the information relative to the Organization entity was collected.

DataSource A *DataSource* is characterized by a *persistent identifier* (if available, as released by the agency or organization providing *DataSource* information, e.g., OpenDOAR for repositories), a *name space prefix* (used to generate unique identifiers for entities generated/provided by the data source in a intelligible manner), an *official name*, an*English name* (if available), a *URL of its web site*, an *URL of the logo*, a *description*, a *contact email* (if available), and and *access information package*, which contains API information useful to access the data source. It is characterized by the following relationships:

(i) the set of *Result Instances* it is hosting (inverse of the relationship *hostedBy* of *Instance* entities);
(ii) the set of *Result Instances* which were collected from it (inverse of the relationship collectedFrom of *Instance* entities)
(iii) *OpenAIREcompatibilityClass* and *OpenAIREcompatibilityScheme*: the degree of compatibility to the OpenAIRE guidelines for content providers of the data source, according to a given classification of compatibility types
(iv) the set of *Organizations* responsible for the *DataSource* (inverse of the relationship *respOrganization* of the *Organization* entities)
(v) *collectedFrom*: the DataSource entity from which the information relative to the DataSource entity was collected.

Repository A *Repository* is a *DataSource* characterized by the further properties inherited from OpenDOAR description: *numberOfItems, numberOfItemsDate, subjects, policies, languages,* and *contentTypes*.

CRIS system/Data Archive Both entities are *DataSources*.

Aggregator An *Aggregator* is a *DataSource* characterized by the following relationships:

(i) *typeClass* and *typeScheme*: the type of *Aggregator* according to a given classification of *Aggregator* types (i.e., the kind of data sources they aggregate, such as repositories, dataset archives, etc), respectively described as a *Class* entity and a *Scheme* entity.

Entity Registry An *Entity Registry* is a *DataSource* characterized by the following relationships:

(i) *typeClass* and *typeScheme*: the type of *Entity Registry* according to a given classification of *Registry* types (i.e., the kind of entities they contain), respectively described as a *Class* entity and a *Scheme* entity.

2.2.2 Structural entities

Title A *Title* is a title of a Result charachterized by a *Name* and a type provided by a *titleTypeScheme* and a *titleTypeClass* properties. The title TypeScheme "DataCite_title" defines the following typeClass values: Alternative Title, Subtitle, Translated Title. A Title is characterized by the relationships *titles*$^{-1}$, which refers to the *Result* to which it belongs.

Date A *Date* is a date different from the "date of acceptance" for Results. It is characterized by a property *Date* and a type provided by a *dateTypeScheme* and a *dateTypeClass* properties. The typeScheme "DataCite_date" defines the following *dateTypeClass* values: Available, Copyrighted, Created, EndDate, Issued, Start Date, Submitted, Updated, Valid. A Date is characterized by the relationships *otherDates*$^{-1}$, which refers to the *Result* to which it belongs.

Identity An *Identity* is a combination of a *PID (persistent identifier)* used to uniquely refer to the individual together with the relative *identifier scheme* (issuer) provided by a *issuerTypeScheme* and a *issuerTypeClass* properties.; e.g., ORCID, ISNI. An Identity is characterized by the relationships *PIDS*$^{-1}$, which refers to the *Person*, the *Organization* or the *Project* to which is assigned (see Figure 6).

Instance An *Instance* represents the combination of the *Web Resources* (i.e., URLs relative to files or file locations, such as "splash pages") associated with a *Result* and the *DataSource* where such *Web Resources* are hosted. An Instance also contains the persistent identifier of the Result, if available. As such, an Instance is characterized by the following relationships:

(i) the *Result* of which it is an *Instance* (the inverse of the relationship *instances* of *Result* entities);

(ii) *webResources*: the set of *WebResources* associated to the *Instance*;

(iii) *instanceTypeClass* and *instanceTypeScheme*: the type of the Result according to a given classification of Result types, respectively described as a *Class* entity and a *Scheme* entity.

(iv) *licenseClass* and *licenseScheme* (mandatory): the license of the *Result* according to a given classification of *Dataset* licenses, respectively described as a *Class* entity and a *Scheme* entity; the Classes should include "Unknown".

Web Resource A *WebResource* is characterized by *its unique URL* and by its relationship with the associated *Instance* (inverse of the relationship *webResources* of *Instance* entities).

Subject *Subjects* are scientific disciplines associated to one *Result*, together with a given semantics. As such they are characterized by the following relationships:

(i) the *Result* associated to the subject (the inverse of the relationship subject of *Results entities*);

(ii) *semanticsClass* and *semanticsScheme:* the semantics of the Subject according to a given classification of Subjects, respectively described as a *Class* entity and a *Scheme* entity.

2.2.3 Static entities

Funding A *Funding* entity represents funds that can be granted to *Projects*. As such it is characterized by the following properties: *name*, *description*, *keywords,* and a *persistent identifier* (if available)*.* Moreover, by the following relationships:

(iii) the *Projects* granted the funding (inverse of relationship of *Project* in *Projects_Funding* entities);

(iv) *semanticsClass* and *semanticsScheme:* the semantics of the Fundings (e.g., in FP7: Supporting and Coordination Actions, I3, STREP) according to a given classification of Fundings semantics (e.g., EC-FP7 contract schemes), respectively described as a *Class* entity and a *Scheme* entity.

2.2.4 Linked entities

Publication_Publication *Publication_Publication* entities represent relationships between *Publication* entities, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Publication1 and Publication2*: the two interrelated *Publication* entities;

(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between Publications, respectively described as a *Class* entity and a *Scheme* entity.

Funding_Funding *Funding_Funding* entities represent relationships between *Funding* entities, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Funding1 and Funding2*: the two interrelated *Publication* entities;

(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between Fundings, respectively described as a *Class* entity and a *Scheme* entity.

Project_Funding *Project_Funding* entities represent relationships between *Projects* and *Fundings* entities, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Project* and *Funding*: the two interrelated entities;
(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Funding* and *Projects*, respectively described as a *Class* entity and a *Scheme* entity.

Organizations_Funding *Organization_Funding* entities represent relationships between *Organizations* and *Fundings* entities, together with the duration and the semantics of the relationship. For example, the EC is the organization behind FP7 funding (Class = funding organization). As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Organization* and *Funding*: the two interrelated entities;
(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Funding* and *Organization*, respectively described as a *Class* entity and a *Scheme* entity.

Organization_Project *Organizations_Projects* entities represent relationships between *Publication* and *Organization* entities, together with the semantics of the relationship. Examples of such semantics, as suggested by CERIF vocabularies, are: subcontractors, principal investigating, exploitation, coordinators, participant. Example is a formal beneficiary of fundings within a *Project*. It is characterized by an *Original Identifier* (e.g., for EC projects it is the participant number, a progressive integer 1,2,3,4, etc., where 1 is assigned to the project coordinator), a *start date* and an *end date*. A Participant is characterized by the following relationships:

(i) the *Project* involved in the *Organization_Project* entity;
(ii) *respOrganization* involved in the *Organization_Project* entity;
(iii) *contactPerson*: the *Person* recorded as contact point for the responsible *Organization*;
(iv) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between Organizations and Projects, respectively described as a *Class* entity and a *Scheme* entity.

Dataset_Dataset *Dataset_Dataset* entities represent relationships between *Dataset* entities, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *dataset1* and *dataset2*: the two interrelated *Dataset* entities;
(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between Datasets, respectively described as a *Class* entity and a *Scheme* entity.

Publication_Dataset *Publication_Dataset* entities represent relationships between a *Publication* and a *Dataset*, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Publication and Dataset*: the two interrelated *Publication and Dataset* entities;

(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Publications* and *Datasets*, respectively described as a *Class* entity and a *Scheme* entity.

Person_Result *Person_Result* entities represent relationships between a *Person* and a *Result*, together with the duration, the semantics of the relationship and the ranking (i.e., order of importance) of the relationship (e.g., to model author ordering). As such it is characterized by the properties *start date*, *end date*, *ranking*, and by the following relationships: *Person* and *Result*: the two interrelated *Person* and *Result* entities;

(i) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Persons* and *Results*, respectively described as a *Class* entity and a *Scheme* entity;

(ii) *personAffiliation*: the *Organization* to which the *Person* creator of the *Result* was affiliated to at the time of *Result* creation.

Result_Project *Result_Project* entities represent relationships between a *Result* and a *Project*, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(iii) *Result and Project*: the two interrelated *Result and Project* entities;

(iv) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Results* and *Projects*, respectively described as a *Class* entity and a *Scheme* entity.

Result_Organization *Result_Organization* entities represent relationships between a *Result* and a *Project*, together with the duration and the semantics of the relationship. As such it is characterized by the properties *start date* and *end date* and by the following relationships:

(i) *Result and Organization*: the two interrelated *Result* and *Organization* entities;

(ii) *semanticsClass* and *semanticsScheme*: the semantics of the relationship according to a given classification of relationships between *Results* and *Organizations*, respectively described as a *Class* entity and a *Scheme* entity.

2.3 Entity-Relationship model

Table 2 reports the attributes and the named associations for the classes in the E-R schema in Figure 3.

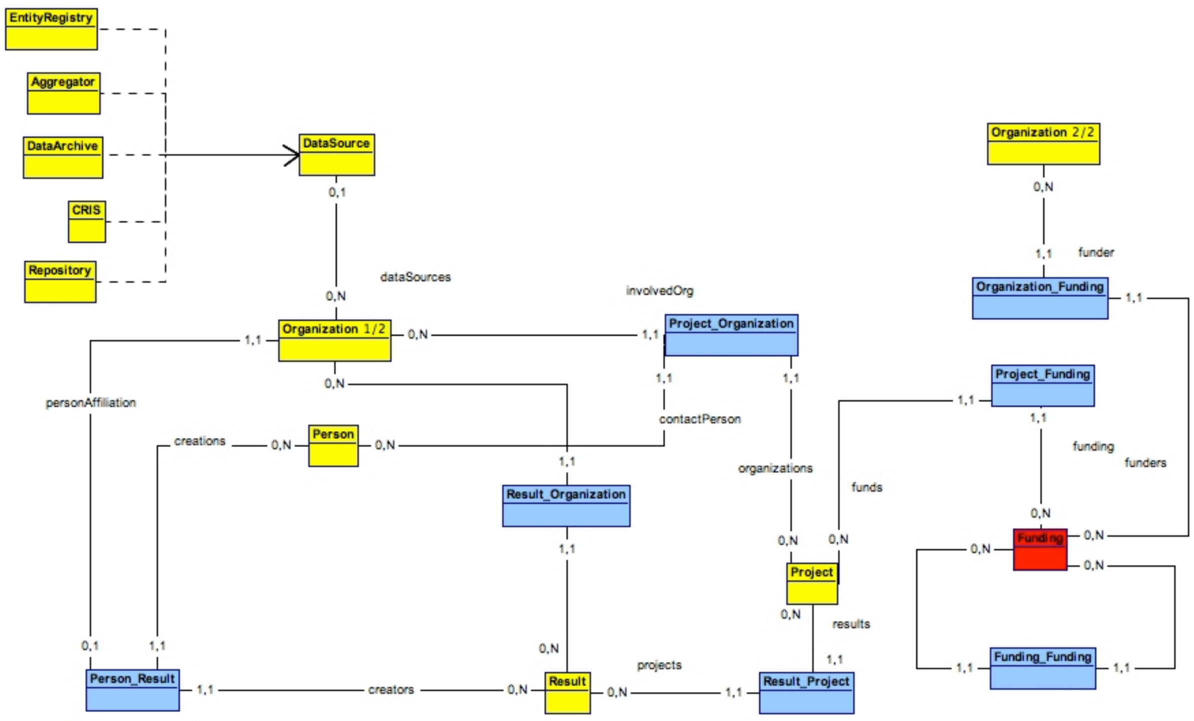

Figure 3 – E-R model: main, linked, static and structural entities

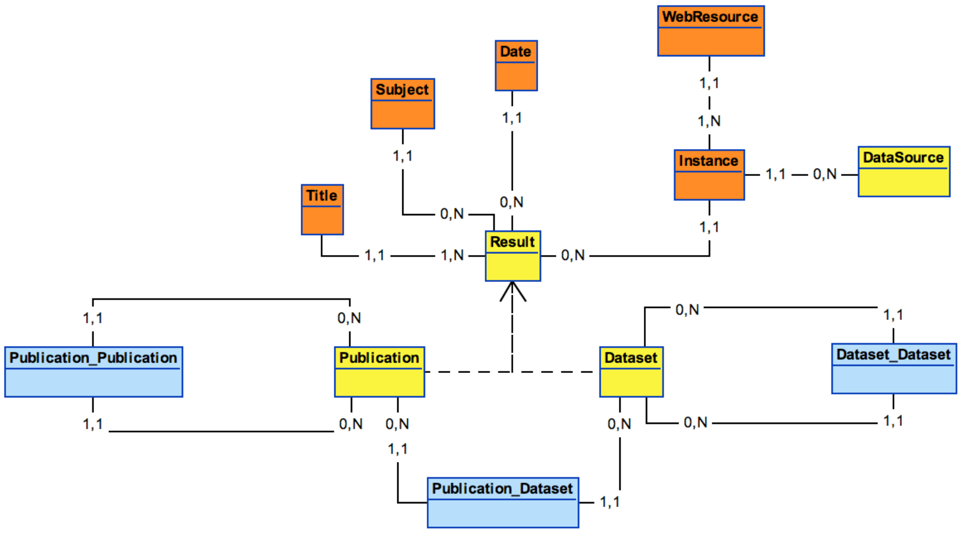

Figure 4 - E-R model: Result entities

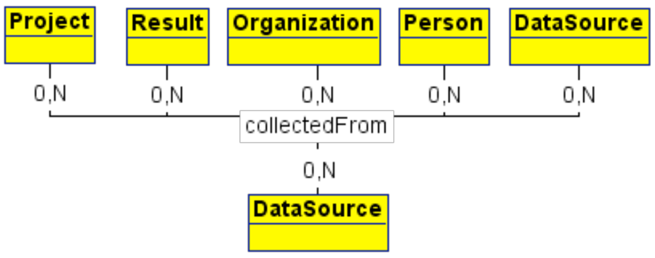

Figure 5 – E-R model: provenance relationships

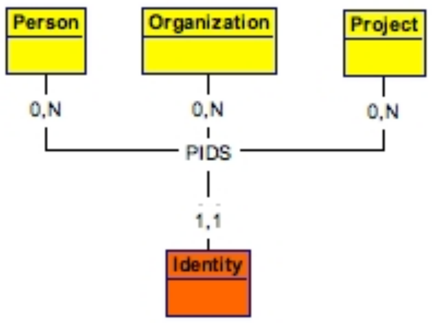

Figure 6 – E-R model: persistent identifiers of main entities

Table 2 – E-R Schema: class properties

Result • publisher (optional) • dateOfAcceptance • description ➔ PIDS (0 or N Identity) ➔ titles (1 or N Title) ➔ creators (0 or N Person_Result) ➔ result^{-1} (0 or N Result_Organization) ➔ instances (0 or N Instance) ➔ subjects (0 or N Subject) ➔ collectedFrom (0 or 1 Data Source) ➔ languageClass (1 Class) ➔ languageScheme (1 Scheme) ➔ relevantDates (0 or N Date)	**Person** • firstName • secondNames • fax • email • phone • creations (0 or N Person_Result) • contactPerson^{-1} (0 or N Project_Organization) • nationalityClass (0 or 1 Class) • nationalityScheme (0 or 1 Scheme) ➔ collectedFrom (0 or 1 DataSource) ➔ PIDS (0 or N Identity) *Constraints:* • This.nationalityClass must be associated to This.nationalityScheme	**Dataset** **(isA Result)** • device (optional) ➔ dataset1^{-1} (0 or N Dataset_Dataset) ➔ dataset2^{-1} (0 or N Dataset_Dataset) ➔ dataset^{-1} (0 or N Publication_Dataset) ➔ typeClass (1 Class) ➔ typeScheme (1 Scheme) *Constraints:* • This.typeClass must be associated to This.typeScheme
Publication **(isA Result)** • journal (optional) • embargoEndDate (optional) ➔ publication1^{-1} (0 or N Publication_Publication) ➔ publication2^{-1} (0 or N Publication_Publication) ➔ publication^{-1} (0 or N Publication_Dataset) *Constraints:* • This.typeClass must be associated to This.typeScheme	**Project** • code • webSiteURL (optional) • acronym • title • start_date • end_date • call_identifier (optional) • keywords (optional) • duration (derived: from start_date and end_date) • EC_SC39 (optional) • organizations (1 or N Project_Organization)	**Organization** • legal short name • legal name • webSiteURL • logoURL (optional) • EC_LegalBody (boolean) (optional) • EC_LegalPerson (boolean) (optional) • EC_NonProfit (boolean) (optional) • EC_ResearchOrganization (boolean) (optional) • EC_HigherEducation (boolean) (optional)

• This.licenseClass must be associated to This.licenseScheme	➔ funds (0 or N Project_Funding) ➔ collectedFrom (0 or 1 DataSource) ➔ projectPIDS (0 or N Identity) ➔ contractTypeClass (0 or 1 Class) ➔ contractTypeScheme (0 or 1 Scheme) *Notes* For EC projects *OriginalIdentifier* contains the grant agreement number. *Constraints:* This. contractTypeClass must be associated to This. contractTypeScheme	• EC_InternationalOrganization EurInterests (boolean) (optional) • EC_InternationalOrganization (boolean) (optional) • EC_Enterprise (boolean) (optional) • EC_SMEValidated (boolean) (optional) • EC_NUTScode (optional) ➔ organization^{-1} (0 or N Result_Organization) ➔ personAffiliation^{-1} (0 or N Person_Result) ➔ dataSources (0 or N Data Sources) ➔ funder^{-1} (0 or N Organization_Funding) ➔ involvedOrg^{-1} or N Project_Organization) ➔ countryClass (0 or 1 Class) ➔ countryScheme (0 or 1 Scheme) ➔ collectedFrom (0 or 1 DataSource) ➔ organizationPIDS (0 or N Identity) *Constraints:* This.countryClass must be associated to This.countryScheme
Data Source • PID • nameSpacePrefix (unique) • officialName • englishName (optional) • webSiteURL • logoURL • contactEmail • accessInfoPackage ➔ OpenAIREcompatibilityClass (1 Class) ➔ OpenAIREcompatibilityScheme (1 Scheme) ➔ hostedBy^{-1} (0 or N Instance) ➔ collectedFrom^{-1} (0 or N Instances) ➔ dataSources^{-1} (0 or N Organizations) ➔ collectedFrom^{-1} (0 or N Person, DataSource, Organization, Project, Instance)	**Instance** • PID ➔ instances^{-1} (1 Results) ➔ hostedBy (1 Data Source) ➔ licenseClass (1 Class) ➔ licenseScheme (1 Scheme) ➔ instanceTypeClass (1 Class) ➔ instanceTypeScheme (1 Scheme)	**Web Resource** • Web Resource URL ➔ webResources^{-1} (1 Instances)

Repository **(isA DataSource)** • OD_description (optional) • OD_numberOfItems (optional) • OD_numberOfItemsDate (optional) • OD_subjects (optional) • OD_policies (optional) • OD_languages (optional) • OD_contentTypes (optional)	**EntityRegistry** **(isA DataSource)**	**DataArchive** **(isA DataSource)** **CRIS** **(isA DataSource)**
Aggregator **(isA DataSource)** ➔ typeClass (1 Class) ➔ typeScheme (1 Scheme) *Constraints:* • This.typeClass must be associated to This.typesScheme	**Subject** ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) ➔ subjects^{-1} (1 Result)	**Funding** • persistentIdentifier • name • description • keywords • funding1^{-1} (0 or N Funding_Funding) • funding2^{-1} (0 or N Funding_Funding) • funders (0 or N Organization_Funding) • funding^{-1} (0 or N Project_Funding)
Person_Result • startDate • endDate • ranking ➔ creators^{-1} (1 Person) ➔ creations^{-1} (1 Result) ➔ creatorAffiliation (0 or 1 Organizations) ➔ personRoleClass (1 Class) ➔ personRoleScheme (1 Scheme) *Constraints:* • This.roleClass must be associated to This.roleScheme	**Publication_Publication** • startDate • endDate ➔ publication1 (1 Publication) ➔ publication2 (1 Publication) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* • This.semanticsClass must be associated to This.semanticsScheme	**Organization_Project** • participantNumber (optional) • startDate • endDate ➔ participants^{-1} (1 Projects) ➔ contactPerson (0 or 1 Persons) ➔ respOrganization 1. Organizations) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Notes* • For EC projects, *OriginalIdentifier* is the participant number (1 for project coordinators) • For other projects, unless they support the concept of

		participant, the OriginalIdentifier is empty.
Dataset_Dataset • startDate • endDate ➔ dataset1 (1 Dataset) ➔ dataset2 (1 DataSet) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* • This.semanticsClass must be associated to This.semanticsScheme	**Publication_Dataset** • startDate • endDate ➔ publication (1 Publication) ➔ dataset (1 DataSet) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* • This.semanticsClass must be associated to This.semanticsScheme	**Funding_Funding** • startDate • endDate ➔ funding1 (1 Funding) ➔ funding2 (1 Funding) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* • This.semanticsClass must be associated to This.semanticsScheme
Organization_Funding • startDate • endDate ➔ funder (1 Organization) ➔ funders^{-1} (1 Funding) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* This.semanticsClass must be associated to This.semanticsScheme	**Project_Funding** • startDate • endDate ➔ funding (1 Funding) ➔ funds^{-1} (1 Projects) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* This.semanticsClass must be associated to This.semanticsScheme	**Project_Organization** • startDate • endDate ➔ involvedOrg (1 Organization) ➔ organizations^{-1} (1 Project) ➔ contactPerson (1 Person) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* This.semanticsClass must be associated to This.semanticsScheme
Result_Project • startDate • endDate ➔ project (1 Project) ➔ result (1 Result) ➔ semanticsClass (1 Class) ➔ semanticsScheme (1 Scheme) *Constraints:* • This.semanticsClass must be associated to This.semanticsScheme	**Title** • name ➔ titleTypeClass (1 Class) ➔ titleTypeScheme (1 Scheme) ➔ titles^{-1} (1 Result)	**Date** • date ➔ dateTypeClass (1 Class) ➔ dateTypeScheme (1 Scheme) ➔ relevantDates^{-1} (1 Result)
Identity • PID ➔ issuerTypeClass (1 Class) ➔ issuerTypeScheme		

(1 Scheme) ➔ projectPIDS^{-1} (1 Projects) ➔ personPIDS^{-1} (1 Person) ➔ organizationPIDS^{-1} (1 Organization)		

3 Data population

As shown in Figure 1 the OpenAIREplus infrastructure collects entity information from an expert-validated entity pool and from data sources of different typologies, such as repositories, CRISs, dataset archives, aggregators, entity registries. All these contain information relative to different interrelated entities of the OpenAIRE data model. In particular, as shown by **Error! Reference source not found.**, data sources of the same typology may deliver metadata records which contain information relative to different entities. For example, an OpenAIRE compliant repository delivers information packages (i.e., metadata records) which contain information about the publication result, the persons who created such result, the projects funding such result, and the instances relative to the result; while a DRIVER compliant repository does not contain information about project entities.

In the following we shall call **original entities** the entities collected from data sources, hence from "authoritative" providers of data, onto the Information Space. The layer of original entities includes entities and relationships between them as collected from the different data sources (i.e., mapped from their original structure noto the one of the OpenAIRE data model). The layer is "stateless", in the sense that entities have "reproducible" identifiers derived by combining identifiers of the entities in the original data sources with a data source identifier assigned by OpenAIRE. In other words, if the same entity or relationships is collected more than once from the same data source, it will be transparently overridden.

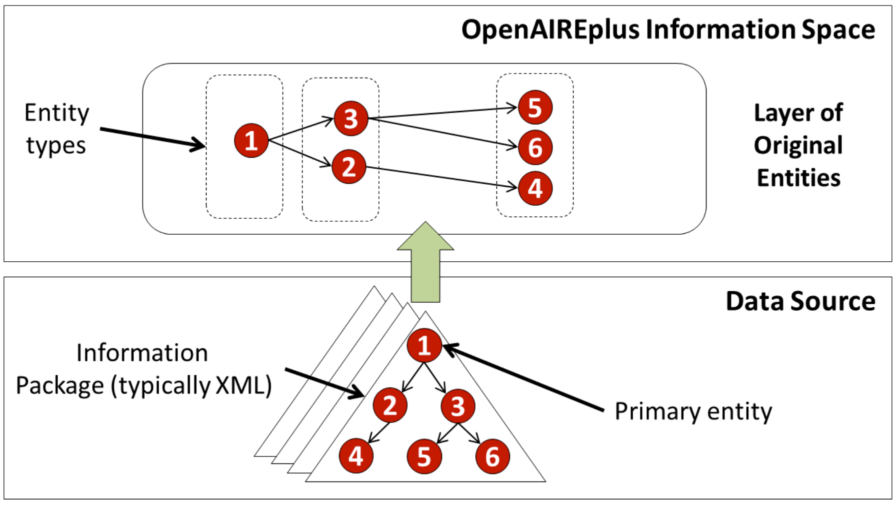
OpenAIREplus Information Space
Entity types
Layer of Original Entities
Data Source
Information Package (typically XML)
Primary entity
1
2
3
4
5
6

Figure 7 – Entity layers: orginal entities

With respect to data population, hence with the process of collecting data from data sources and map them onto the information space, this section we will introduce the notions of:

- **Information packages** and **population workflows**: entity ingestion from data sources into the information space;
- How to assign a **stateless (and permanent) identifiers** to the entities when they enter the information space;

For each of these aspects, we shall provide and explanation of the problem, an extension of the data model to handle the problem, and, where necessary, a solution to the problem using the updated model.

3.1 Information packages

In OpenAIRE entities are collected from external data sources in the form of "information packages". This notion aims at generalizing the OpenAIRE scenario of bibliographic metadata records import from repository data sources to other data source typologies and other types of (primary) entities. In particular, we shall call *information package* a file in some interpretable format (e.g. XML), which contains *identifier* and *information* (e.g., properties) relative to one entity, called *primary entity*, of a given entity type. An information package may contain information (but not necessarily the identifier) relative to other entities (of likely different entity types), called *sub-entities,* which must be directly or indirectly associated with the package primary entity. Figure 7 shows an example of an information package whose primary entity is 1: for example, an information package from OpenDOAR is relative to a repository data source and can be identified by the relative OpenDOAR identifier. Its sub-entities are those from 2 to 6: for example, an OpenDOAR package also contains information about the organization responsible for the repository data source.

3.2 Population Workflows

Original entities are collected from information packages originating from various data sources. We call *population workflow* the process that takes the information package from a data source, extracts its primary entity and its related entities, and stores them into the OpenAIRE information space. A workflow is therefore dependent on:

- The *data source typology* (including the expert-validate entity pool);
- The *access method*, namely (i) protocol required to get the data (e.g., OAI-PMH, JDBC, FTP) and (ii) relative access configuration (e.g., entry point, parameters, etc.);
- The *primary entity type* of the information packages;
- The *XML structure of the information packages* at hand, which depends on the primary entity type.

Note that data sources of the same typology may deliver information packages relative to different primary entity types (in general we can assume they will do it from different access points). For example, CRIS systems may expose through OAI-PMH both publication or project primary entities.

Information package structure (OpenAIREplus guidelines) The OpenAIREplus infrastructure will includes services capable of handling automated collection of entities from data sources according to given population workflows. To this aim, the OpenAIRE guidelines will describe which XML information packages structure should be expected for each population workflow triple

<datasource typology, access method, primary entity type> → XML information package structure

available to the system. WP6 will develop services to automatically process the information packages and insert the relative entities onto the information space.

Information package heterogeneity and harmoniztion Unfortunately, the "raw" information packages exported by data sources will likely not match the information package structures to be identified in the previous step. For example, CRIS systems generally support OAI-PMH harvesting of information, but may export information packages relative to the same entities (e.g., projects, publications) in different XML formats. To this aim, WP6 will update its transformation services in order to map the specific structures exposed by a data source through a given workflow so that they match the expected information package structure.

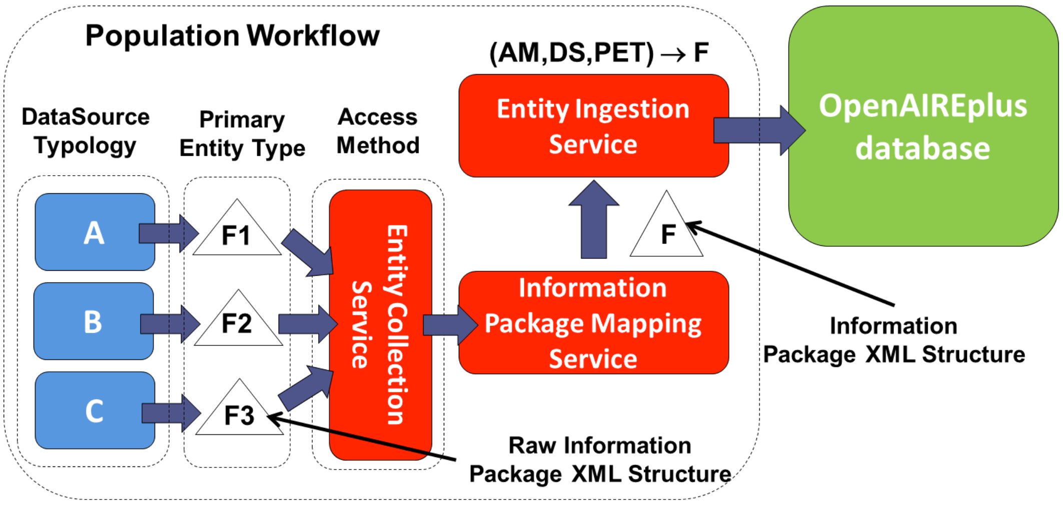

Figure 8 – Information Packages: ingestion workflows (AM = Access Method, DS = Data source typology, PET = Primary Entity Type, F = information package Format): data sources of the same typology export the same primary entity of the same type through different "raw" information package format structures.

3.3 Identity of original entities

Original entities reach the information space from different workflows. Once they enter the information space they must be assigned a unique "stateless" identifier. The data sources of such entities are not under the OpenAIRE infrastructure control and may in any moment decide to delete, update, or add new entities or relationships between them. Hence, it is particularly important to make sure such identifiers are generated from the incoming information packages in a stateless and stable way that is "if the same entity enters the information space at different times, it will be assigned the same identifier".

To this aim, the OpenAIRE infrastructure constructs indentifiers for primary entities and sub-entities in an information package by combining three levels of scope: data sources, relative primary entities, and sub-entities of such primary entities. More specifically:

- *Infrastructure scope:* all data sources are registered and assigned a unique identifier in OpenAIRE;
- *Data source scope:* information packages from the same data source contain one primary entity with an identifier which is unique in the context of the data source;
- *Primary entity scope:* information packages may contain a number of sub-entities relative to the primary entity; unlike primary entities, sub-entities may not necessarily come with an identifier (data source scope)[1] and can be generally

[1] Absence of identifiers may be due to the fact they do not have an identifier on the original data source, e.g., authors of publications in bibliographic metadata records, or to the structure of the information package export format, which focuses on the main entity only.

uniquely identified in the scope of the primary entity based on their properties. The process of identification of such "unique information" is very much dependent on the given information package structure.

Primary entity identifiers The process of generation of stateless identifiers for primary entities is based on a *data source scope strategy*. Independently of the workflows, the type of entity, and the data source kind, primary entity identifiers are always obtained by concatenating the name space prefix of the data source with the primary entity identifier (using and underscore):

nameSpacePrefix_mainEntityID

Although one may consider an *infrastructure scope strategy*, where the assumption is that all primary entities identifers are persistent identifiers, therefore unique across several data sources, in OpenAIRE this is not generally the case, hence we adopt a common and safe strategy of identifier generation.

Sub-entity identifiers Assigning identifiers to sub-entities can be performed following different strategies, some more "optimistic" and some more "pessimistic" about the ability of inferring unambiguous "unique information" for sub-entities from their properties in the information package. For example, one may assume that:

- *Infrastructure scope strategy*: sub-entities with the same "unique information" are collapsed in the same entity across different data sources (infrastructure scope). Entity splitting will identify and solve possible entity "overloads" in a second stage.

  uniqueInformation

- *Data source scope strategy*: Sub-entities with the same "unique information" are collapsed in the same entity but only within the same data source scope. Entity splitting will identify and solve possible entity "overloads" in a second stage.

  nameSpacePrefix_uniqueInformation

- *Primary entity scope strategy*: Sub-entities with the same "unique information" are collapsed in the same entity but only within the same primary entity scope (see Figure 9). De-duplication of entities will solve redundancy in a second stage.

  nameSpacePrefix_mainEntityID_uniqueInformation

Assigning identifiers to sub-entities follows different strategies depending on the specific workflow, hence the relative information packages structure.

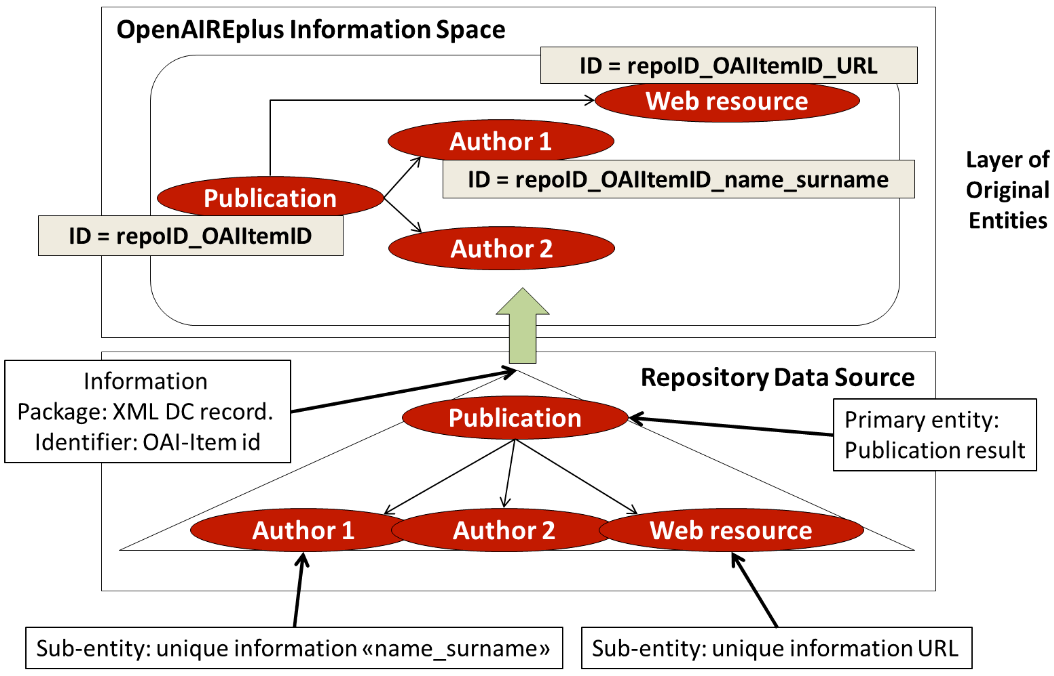
OpenAIREplus Information Space
ID = repoID_OAIItemID_URL
Web resource
Author 1
ID = repoID_OAIItemID_name_surname
Publication
ID = repoID_OAIItemID
Author 2
Layer of Original Entities
Information Package: XML DC record. Identifier: OAI-Item id
Repository Data Source
Publication
Primary entity: Publication result
Author 1
Author 2
Web resource
Sub-entity: unique information «name_surname»
Sub-entity: unique information URL

Figure 9 – Assigning unique identifiers to sub-entities: primary entity scope

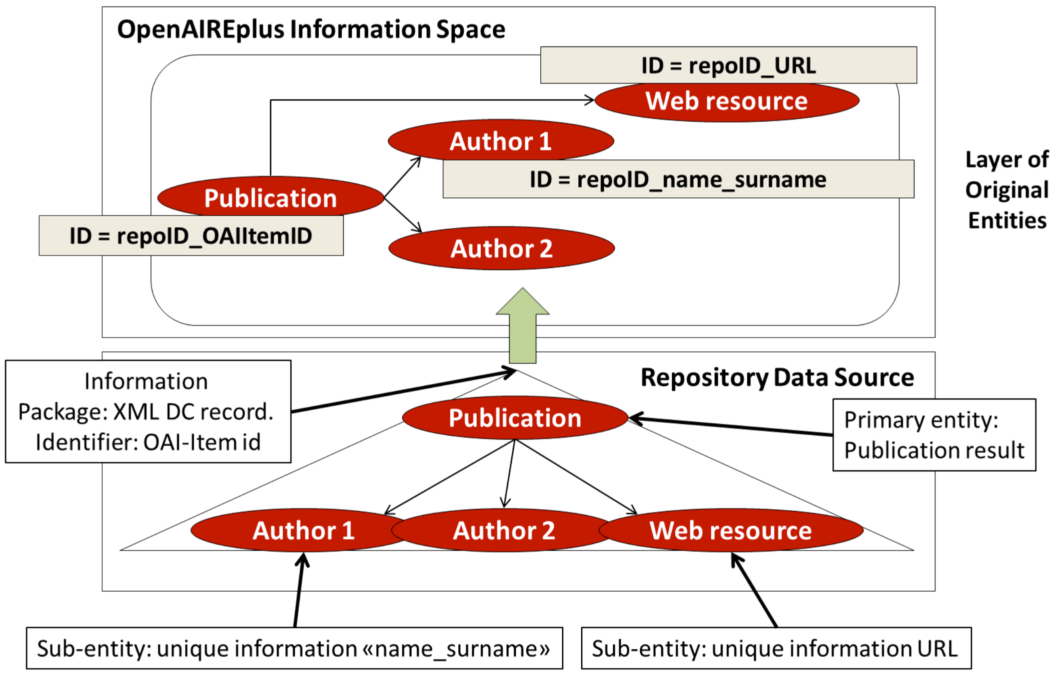
OpenAIREplus Information Space
ID = repoID_URL
Web resource
Author 1
ID = repoID_name_surname
Publication
ID = repoID_OAIItemID
Author 2
Layer of Original Entities
Information Package: XML DC record. Identifier: OAI-Item id
Repository Data Source
Publication
Primary entity: Publication result
Author 1
Author 2
Web resource
Sub-entity: unique information «name_surname»
Sub-entity: unique information URL

Figure 10 – Assigning unique identifiers to sub-entities: data source scope

4 Data inference

As highlighted in Section 1, the process of mapping **information packages** onto **original entities** is not enough to ensure a high quality information space. Indeed, the autonomy of the data sources brings in three main issues:

- A certain degree of information duplication: information about the same real-world entities will be collected from several data sources;
- A certain degree of entity disconnection: data sources deliver entities and relationships between them, which are generally limited to the boundaries of the data source; cross-data source relationships are generally missing;
- A certain degree of inaccuracy: data sources may deliver only a portion of the entity information required by the OpenAIREplus information space.

To tackle such issues, original entities serve as input to a data inference process, which yields a layer of **inferred entities** resulting from resolving entity inaccuracy and disconnection thorugh a mix of human intervention and information inference services. Inferred entities logically "override" original entities in the information space to provide an enhanced view of the information space, namely the layer of **visible entities**, which will be effectively accessible to end-users and applications. The data inference process, which leads from original entities to the new set of inferred entities, consists of four main phases, at the end of which the set of inferred entities is actually transferred into the information space and used to give life to a new set of visible entities. The phases are executed over a clone of the OpenAIRE information space, containing a copy of the current set of original entities. The phases are:

1. *De-duplication of original entities*: the phase returns a first set of inferred entities which delivers a set of disambiguated visible entities;
2. *Data inference actions over the set of visible entities*: the phase returns a second set of inferred entities, which enriches the first set and therefore further updates the set of visible entities; this second set enriches the information space of new entities but may introduce further duplicates;
3. *De-duplication of the set of visible entities*: the phase returns a third and final set of inferred entities which delivers a set of disambiguated visible entities;
4. *Transfer the final set of inferred entities into the information space*: the set of inferred entities is moved from the information space clone into the production informarion space in order to re-calculate the set of visible entities.

The whole data inference process is always executed over original entities and can be re-executed anytime to generate a new visible entity layer. As such, there is no need to keep in the information space a history of inference actions, since both original entity and inferred entity layers are intended to be always "refreshed" by executing a data population operation or a data inference operation. Still, the information space data model needs to be extended to enable the distinction between original and inferred entities and the identification of visible entities as a logical overlap of the former two.

In this section we shall first introduce the data inference actions which can be executed in OpenAIRE and comment on how such actions are to be encoded in the OpenAIRE data model.

4.1 Inference actions

OpenAIREplus original entities can be subject to the following *data inference* actions, performed by data mining and inference services and by humans:

- *Merging of entities;*
- *Adding new inferred entities (relationship between main entities);*
- *Updating of properties of an original entity*;
- *Removing original entities.*

Such actions may be caused by different data inference workflows, which may or may not involve humans. Examples in the first OpenAIRE service settings are:

- *End-user feedbacks*, when approved by data curators;
- *Data curator* edit activities;
- *Automatic inference algorithms*: (i) similarity relationships between result entities, (ii) information extraction from full-text of publications, which may infer title, authors, organizations, and emails of publication entities, (iii) citation management, which may lead to the introduction of publication results not available to the information space, (iv) subject classification, which may lead to the introduction of new entities, etc.
- *End-user claim/vaidation actions* through the OpenAIRE portal, e.g., project coordinators validating publication-project relationships.
- *Deduplication services* identifying a set of entities of the same type which all correspond to the one and the same real-world entity.

In the following we shall focus on how to modify the OpenAIRE data model in order to represent the outcome of data inference actions on the information space and to resolve data inference conflicts that may arise between original and inferred entities. For each of these aspects, we shall provide and explanation of the problem, an extension of the data model to handle the problem, and, where necessary, a solution to the problem using the updated model.

4.1.1 Trust and Inference Provenance of Entities

Inference actions, hence the entities they bring onto the information space, may have different **levels of trust**, depending on their service or agent which generated them, namely **entity inference provenance**. Main entities and linked entities in the model are therefore enriched with three properties:

- *addedByInference*: a flag which is set to TRUE when the entity is created as a consequence of an inference action, including de-duplication. Set to FALSE for original entities.
- *deletedByInference*: a flag which is set to TRUE when some inference process establishes that the entity should be removed from the visible entities.
- *Trust*: a 0 to 1 value that establishes the level of credibility of the information. By definition, original entities are assigned a OE level of trust, while entities from the expert-valided entity pool a value EVE > OE. Inferred entities are assigned a value IE < OE which depends on the quality and reliability of the automatic inference mechanisms or of the humans bringing the information.

- *Inference Provenance*: a value from a controlled dictionary which encodes the algorithm/process/service which brought the inferred entity in the information space, e.g., "de-duplication", or deleted, i.e., made invisible, an original entity from the information space. In the case of original entities the property is initially set to NULL.

4.1.2 Merging of entities

The de-duplication service operates over the set of original entities of the same main type (i.e., Person, Organization, Result, DataSource, Project) and identifies groups of entities which are redundantly describing the same entity. For each group, the service identifies a *representative entity*, which by default is the one with the higher level of trust among the duplicates (if more entities have the same level of trust, the one with the shorter identifier is selected). The remaining duplicate entities (*merged entities*) will not be visible to end-users and will point to the representative entity. However, in the case merged entities bear values for properties which are unavailable (empty values) for the representative entity, such values will be used for indexing and visualization purposes ("shadow entity" strategy). Hence, the ordering of the entities plays an important role in establishing their usage within the information space.

The second de-duplication phase occurs after a number of inference actions which may introduce further duplication. In this context, if a group of duplicates includes a representative entity, then all entities merged into the latter are included in the new group and a new representative entity is elected.

Merging introduces further issues, which regard the strategy to be adopted to cope with linked entities, static entities, and structural entities associated with entities that were merged and should therefore be excluded from the set of visible entities. In other words, in order to offer a coherent view of the space, all entities "sourrounding" merged entities must be altered in order not to be associated with with the representative entity. For example, merging two Publication p1 and p2 into p1 implies that Project, Authors, Instance, and Subject entities of p2 will be linked to p1 (avoiding duplicated associations in the case p1 already points to the same entities). Note that Title and Date structural entities are considere part of a single Result and will not be linked to the representative entity.

To this aim the data model needs to represent the possibility of adding a new relationship with a status *Inferred*, which encodes such re-linking, withouth losing the original relationship (in order to "clean" the inferred information and return to the original layer of entities). As shown in Figure 11, this requires a major change, since explicit relationships between entities, such as *collectedFrom*, become **explicit relationship entities** (in green in the Figure) which include properties capable of capturing such status. Specifically, the model will include:

for each main entity type:

- A relationship *mergedWith*, which points from one entity to the group-representative entity;

for all explicit relationships (not modelled through linked entities):

- Properties *Inferred, deletedByInference, Trust,* and *Inference Provenance*

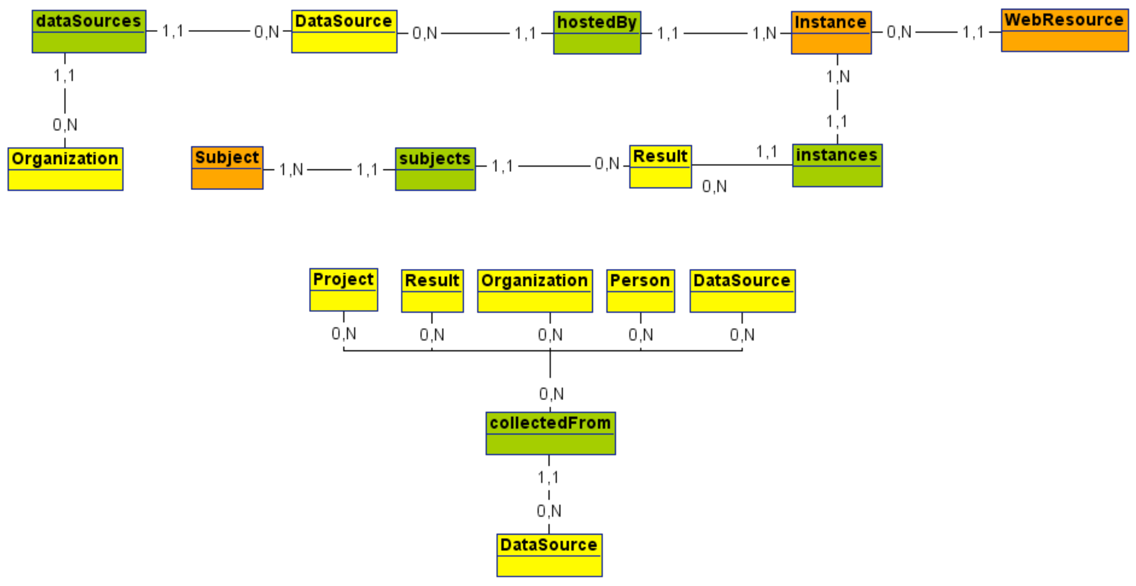

Figure 11 – Extension to data model to cope with inferred entities: explicit relationships become entities

When merging a set of entities $e_1,\dots,e_K$ into a representative entity e_j two main strategies are adopted, which regard explicit relationship entities and linked entities. As an example we shall consider Result entities.

Explicit relationship entities Result entities are surrounded by the following explicit relationship entities: *Instances, CollectedFrom*, and *Subjects*.

Instances entities associate Results with a set of Instance entities (and in turn a set of WebResource entities). In this case, all Instance entities of merged entities are to be connected to the representative entity, so that all files of all Results are visible. To this aim, for each Result in $r_2,\dots,r_K$ all *instances* relationships are set the flag *deletedByInference* to TRUE. All such relationships are cloned, i.e., the same values for all properties, but with the flag *inferred* set to TRUE, *deletedByInference* set to FALSE, *InferenceProvenance* set to "de-duplication", *Trust* set to a given value, and as target Result entity the representative entity.

The same strategy is adopted for the other explicit realtionship entitites surrounding Result entities, namely *CollectedFrom* and *Subjects*. Existing explicit relationship entities for merged entities are made logically invisible (set the flag *deletedByInference* to TRUE) and clone relationship entities are created which point to the representative entity and have the flag *inferred* set to TRUE. In the case the source entity of the relationships to be cloned is already linked with the representative entity, the relationship is not created (e.g., merged entities and representative entity share a subset of Subject entities).

Linked entities Result entities are surrounded by the following linked entities:

- Relationship *projects* to Result_Project linked entities
- Relationship *creators* to Person_Result linked entities
- (If *Publication* entity) relationship *pub1 and pub2* to Publication_Publication linked entities
- (If *Publication* entity) relationship *publication*$^{-1}$ Publication_Dataset linked entities
- (If *Dataset* entity) relationship *dataset1 and dataset2* to Dataset _ Dataset linked entities

- (If *Dataset* entity) relationship *dataset* $^{-1}$ Publication_Dataset linked entities

For each linked entity involving merged entities, the same two steps highlighted above take place. Existing linked entities for merged entities are made logically invisible (set the flag *deletedByInference* to TRUE) and clone linked entities are created which point to the representative entity and have the flag *inferred* set to TRUE. In the case the source entity of the linked entity to be cloned is already linked with the representative entity, the relationship is not created (e.g., merged entities and representative entity share a subset of Project entities).

Note: as an optimization of the merging process, which leads to cloning of entities hence to a possible increase of the entities in the information space, we could keep track of the changes inferred by merge actions within linked entities and explicit relationship entities. For each linked entity type and explicit relationship entity type connecting the entities A and B, the model would feature two associations *original_<entityA>* and *original_<entityB>* as an addition to the associations *<entityA>* and *<entityB>* which such entities contain. The former pair keeps memory of the original pointers *<entityA>* and *<entityB>* to the entities in A and B before the merge took place; the inferred pointers will substitute the original ones and be kept in *<entityA>* and *<entityB>*.

4.1.3 Adding an inferred entity

Some inference processes may lead to the introduction of new main entities together with linked entities or explicit relationship entities. Consider for example the citation inference process, which returns a set of Publication_Publication entities with semantics of type "citation" and may add new Publication entities relative to the citations that are not in the information space. In this case, all such entities are added with the *inferred* flag set to TRUE, *deletedByInference* set to FALSE, *InferenceProvenance* set to "citation_inference", *Trust* set to a given value.

4.1.4 Updating an original entity

Updating an entity consists in selecting one main entity and updating its property values or its relationships with other entities. Updates are modeled as additions of entities obtained from cloning the updated entity. The new entity is tagged as *inferred* and has a level of trust UE, to be used in the second de-duplication phase. Depending on the value, the entity may become the representative entity of a group which includes its original clone or be used as "shadow" entity in such a group.

4.1.1 Removing an entity

An entity can be removed from the information space by setting its flag *inferredAsDeleted* to TRUE. As any other inference action, the action is accompanied by its inference provenance information and level of trust.

5 General data management issues

5.1 Administrative entity properties

All entities feature the following administrative properties:

- *Timestamp of creation* (linked entities have a start date and an end date inherited by the CERIF semantic layer);

All entities but Structural and Static entities:

- *Timestamp of inference action, i.e., deleteByInference, mergedWith*
- A flag *visible*, set to TRUE if the entity is a visible entity, to FALSE otherwise
- A flag *ForMining*, set to TRUE if the entity should not be indexed but only delivered to the Information Inference Service (WP7)

Data Source entities:

- Name Space Prefix property, which indicates a meaningful (intelligible) string to be used to prefix local identifiers of entities and form a unique OpenAIRE identifier.

5.2 Users management

A set of Users which may have a number of Roles and, after having registered, may "create" a number of Results in the database, by fetching them from different kinds of sources or by manually ingesting them. Figure 12, shows the relationships between the tables whose properties are listed in Table 3. The "email" was chosen as primary key to follow the indication of D-NET user profiles, with which the table will be synchronized.

Table 3 - Database extension: User management in OpenAIRE, tables and properties

User • email: PK String • firstName: String (optional) • secondName: String (optional) • institution: String (optional)	**Creation** • creation: PK FK Results(resultID) • creator: PK FK Users(email) • dateOfCreation: Date • lastUpdateDate: Date	**Role** • name: PK String
User_Role • role: PK FK Roles(name) • user_email: PK FK Users(email)		

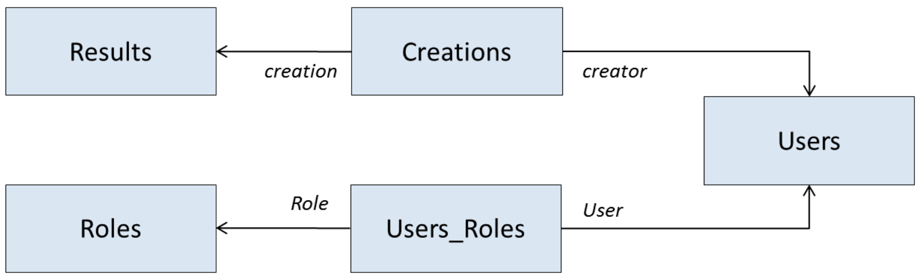

Figure 12 – Database extension: User management in OpenAIRE, tables and relationships

5.3 Visualization issues

Usage of vocabularies and classification schemes through the semantic layer The property *Name* of Class and Scheme is replicated into linked entities, main entities and structural entities (when used to model vocabularies) to enable visualization of relationship instances.

Usage of hierarchies of Funding entities In order to ease the visualization of a hierarchy of Funding entities, each entity is assigned an encoding of its position in the tree. The property *position* is based on the assumption that the *Funding* "nodes" of a *Funding* classification trees are numbered starting by 0 (where the root is virtually the fundingScheme), from left to right and down to lower levels (see Figure 13). The identifier of a node is then obtained as the numbering path leading from the root to the node. **Note**: node additions, movements, or deletions cause the *Funding* tree to be re-numbered, hence *position* properties to be updated.

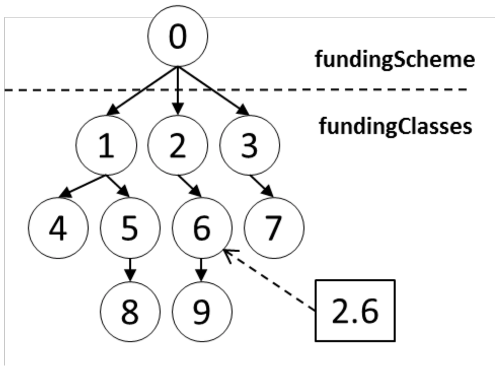

Figure 13 – Tree numbering for node positioning

6 OpenAIRE Data Sources

6.1 Initial instantiation of the database

The database as described above will initially be instantiated to include:

- *FP7 projects* as Funding static entities;
- *Licensing vocabulary* as Class and Scheme entities: to be provided by OpenAIREplus, currently: Embargo, Copyrighted, OpenAccess
- *Subject vocabulary* as Class and Scheme entities;
- *ResourceType* for Datasets as Class and Scheme entities: DataCite v2.2 vocabulary;
- *Title* types as Class and Scheme entities: DataCite v2.2 vocabulary;
- *Date* types as Class and Scheme entities: DataCite v2.2 vocabulary;
- *Nationality* types as Class and Scheme entities: ISO standard used in OpenAIRE;
- *Language* types as Class and Scheme entities: ISO standard used in OpenAIRE;
- *Linked entities: at least one vocabulary of Class and Scheme entity for each linked entity*: CERIF data model.

6.1.1 CORDA data source: FP7 EC Projects

The cooperation with CORDA (ref. Veronique Riegler) consisted in identifying a mapping between the CORDA wharehouse tables and the tables related with project data defined in the OpenAIRE data model. In the following, for each OpenAIRE table we provide the field-to-fiel correspondence with the data that will be provided by CORDA, together with the operations that will be need to be implemented to refine the mapping, such as format conversion or primary key generation.

Projects

OpenAIRE Field name	OpenAIRE Format	OpenAIRE Notes	CORDA Field name	CORDA format	CORDA Notes
ProjectID	Integer	**Primary Key**	Project ID (grant_agreement_number)	Integer	**Primary Key**
Web_site	String		Project Internet Address	String	
EC_project_website	String		Project Call Webpage URL	String	
Call_identifier	String		Project Call Identifier	String	
Acronym	String		Project Acronym	String	
Title	String		Project Title	String	
start_date	Date	**Check for format convertion**	Project Start Date	Date	
fundedHow	String	**Foreign key**	Project Funding Scheme	String	
end_date	Date		Project End Date	Date	
SC39	Boolean		Special Clause Model=sc_fp7_39	String	
fundedBy	String	**Foreign Key**	Project Program	String	

Persons

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
personID	PK	**Primary Key**: hash-generated ID	**Not provided**		
projectID	Integer	(only for persons from CORDA, used to generate personID)	Project ID	Integer	**Primary Key**
participantOrder	Integer	(only for persons from CORDA, used to generate personID)	Participant Order	Integer	**Primary Key**
projectRole	String	(only for persons from CORDA, used to generate personID)[1]	Person Role	String	
Name	String		Person Last Name	String	
Surname	String		Person First Name	String	
Nationality	String		**Not provided**		
Email	String		Contact email	String	
Function (role in project, currently implicitly is "coordinator")	String		Contact function	String	
Fax	String		Contact fax	String	
Phone	String		Contact phone	String	

Organizations

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
OrganizationID	**String**	**Primary Key**	Organisation PIC	String	**Primary Key**
Legal_short_name	String		Organisation Short Name	String	
legal_name	String		Organisation Legal Name	String	
web_site_url	String		Organisation Web Page	String	
logo_url	String	Optional	**Not provided**	N/A	
country_of_origin: countryID	String	**Foreign Key** (ISO code: it should match the one used by CORDA - check)	Organisation Country	String (ISO code)	
legal_body	String		Public Body (flag Y/N)	String	
legalPerson	Boolean		Legal Person (flag Y/N)	String	
nonProfit	Boolean		Non Profit (flag Y/N)	String	
researchOrganization	Boolean		Research Organisation (flag Y/N)	String	
higherEducation	Boolean		Higher Education (flag Y/N)	String	

[1] This value will always be "Coordinator" (at least the string used by CORDA to describe the coordinator, we should check for the real value when the data will come). This is because we only have access to Persons when they are coordinator a project. By determining Person ID based on the person role we leave ourselves the possibility to host Persons with different roles in the future. In this case we would require to add a relationships Persons-Participants N:M to the ER schema, where we describe the kind of relationships, rather than relying on a 0:N relationship CoordinatingPerson as we are doing now (see table Participants).

internationalOrgEurInterest	Boolean		International Org Eur Interest (flag Y/N)	String	
internationalOrganizaztion	Boolean		International Organisation (flag Y/N)	String	
Enterprise	Boolean		Enterprise (flag Y/N)	String	
SMEvalidated	Boolean		SME Validated (flag Y/N)	String	

Participants

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
ParticipantID	PK	**Primary Key (Generated from)**	**Not provided**		
Project	FK	**Foreign Key to Projects**	Project ID	Integer	**Primary Key; Foreign Key to Projects**
ParticipantNumber	Integer		Participant Order	Integer	**Primary Key**
RespOrganization	String	**Foreign Key to Organizations (transform Organization PIC in our current format: ORG-…)**	Organisation PIC	String	**Foreign Key**
CoordinatingPerson	FK	**Foreign Key to Persons** (hash-generated ID, based on Project, ParticipantNumber and string “Coordinator”)[1]	**Not provided**	Not provided	
Coordinator	Boolean	TRUE if beneficiary_number = 1 false otherwise	**Not provided**	Not provided	

Frameworks

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
FrameworkID	String	**Primary Key**	Project framework	String	**Primary Key**
Name	String	Use Project Framework values	**Not provided**		
Acronym	String	Use Project Framework values	**Not provided**		

SpecificProgrammes

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
specificProgrammeID	String	**Primary Key**	Project specific program	String	**Primary Key**

[1]The string used by CORDA to describe a coordinator should be used, which we assemued here to be "Coordinator". We should check when the data will arrive if this is the case.

Name	String	Use Project Framework values	**Not provided**		
Acronym	String	Use Project Framework values	**Not provided**		
Description	String		Project specific program description	String	
FrameworkID	String	**Foreign Key**	Project framework	String	

Programmes

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
programmeID	String	**Primary Key**	Project Program	String	**Primary Key**
Name	String		Project Program Description	String	
Acronym	String		**Not provided**		
specificProgrammeID	String	**Foreign Key**	Project Specific Program	String	

FundingSchemes

OpenAIRE field name	OpenAIRE Format	OpenAIRE Notes	CORDA field name	CORDA format	CORDA Notes
fundingSchemeID	String	**Primary Key**	Project Funding Scheme	String	**Primary Key**
Name	String		Project Funding Scheme		
Description	String		Project Funding Scheme Description	String	,

7 Managing the Information Space using HBase

7.1 Mapping information space entities into HBase data model.

The Information Space will be stored in three different storage services:

- HBase NOSQL Service: main entities and linked entities will be stored into such service according to the Column Family model described below.
-
- Relational Database Service: static entities, such as funding schemes, subjects, Schemes and Classes, will be stored into a relational database. The same will hold for the relationships between Results and End-users, when latter execute claim or deposition (in the Orphan repository) actions.

7.1.1 Storing Main entities and Linked Entities

In the infrastructure, the data collection flow can be summarized as follows (see Figure 14):

1. Collection of information packages from external data sources onto MDStores and relational database Service;
2. Storage of information packages into MDStores and relational database.
3. Unpackaging of information packages into entities and storage of such entities into the HBase Service in the Column Family "Original entities".

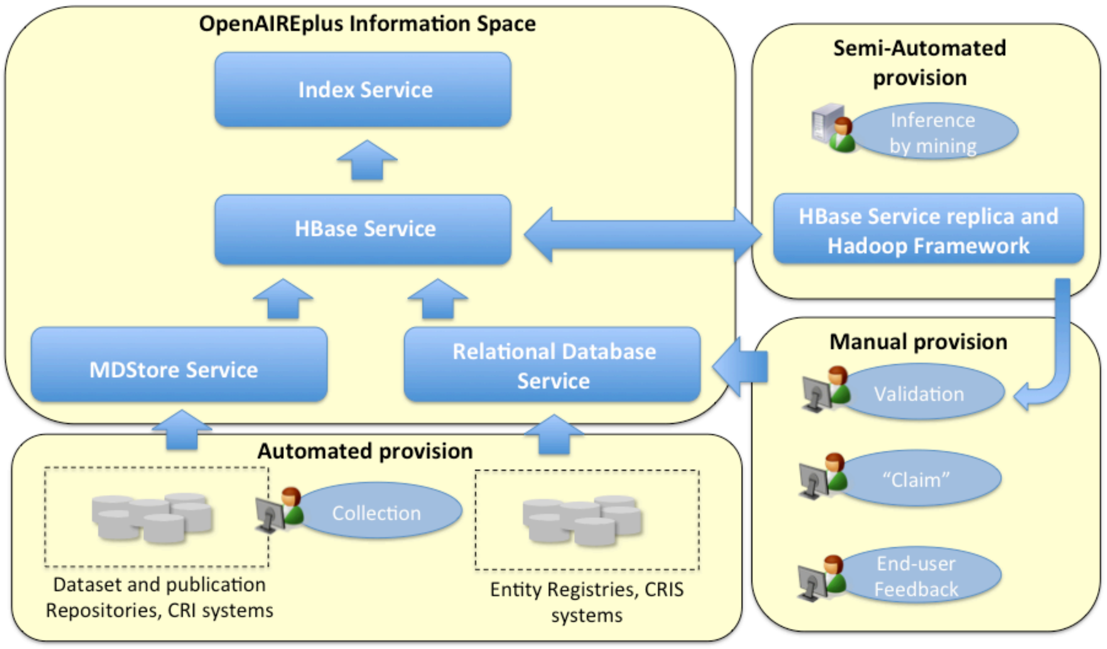
OpenAIREplus Information Space
Index Service
HBase Service
MDStore Service
Relational Database Service
Semi-Automated provision
Inference by mining
HBase Service replica and Hadoop Framework
Manual provision
Validation
"Claim"
End-user Feedback
Automated provision
Collection
Dataset and publication Repositories, CRI systems
Entity Registries, CRIS systems

Figure 14 - Data Flow and services

The HBase service will store in each row a main entity and a linked entity, organized in the following columns:

The following columns characterize rows:

- The identifier of the row;

- The entity type of the object;
- The original object (protocol buffer encoding), as provided by the MDStore (XML record) or the Relational Database (XML encoding of relational record);
- A list of relationships to other objects in the information space: the column cell may contain the properties of such relationships, if any, e.g. class, scheme (protocol buffer encoding);
- A property stating if the object has been collected from data sources or added by inference services;
- A property stating if the object has been deleted by inference services;
- An XML encoding of the object to be used for full-text indexing (see D6.2 for details).

Consolidation A second Hadoop processing will complete the XML encoding to contain all fields and relationships required for visualization in the full-text Index. The process will collect for each entity (e.g., a Result), the values relative to properties of other entities related with the former (e.g., project names) to be used for visualization and search purposes. The process will store the result in another Column Family, named "Complete entities", in order to leave untouched the Column Family relative to the original data as collected from the data sources, and possibly repeat the process again without regenerating such content.

Deduplication Subsequent Hadoop processing will de-duplicate the information space, starting from Results and then completing with Persons and Organizations. As a result of such processing, the cell of the column Body of the represenatative entities (see entity merge actions above) will be modified to include the information relative to provenance of the merged entities. For example, in the case of Publicaitons, the Instances, the DataSources (collectedFrom), and the Projects relative to all merged entities will be embodied in the representative entity. Similarly the Body cell of the merged entities will be included in the entity XML visualization template. In the same action, "shadow fields" may be added to the record, if necessary (when a field value is missing in the representative entity).

7.2 Full-text Indexing Strategy

In OpenAIRE the portal will enable the following access functionalities:

- Full-text search of entities of the main entity types;
- Faceted browsing over properties of the main entity types;
- Navigation between main entities connected by linked entities.

The D-NET index service is based on Solr Apache full-text index. To address the requirements above, the index will be organized to contain records relative to entities of the main types. In particular, such records typically contain:

- A property *entityType* which classifies the record w.r.t. its entity type;
- Properties of the structural entities and static entities connected to the main entity;
- Properties named after the relationship (Scheme-Class pairs) between the entity and other entities whose values are significant properties of the target entity record; for example, the relationship "author" between a result entity A and a person entity B will be materialized into the index as a property *author* with value

the name of B; note that records can include realtionships at multiple levels of depth, e.g., data sources of results;

- A property *record* which contains the XML representation of the record for user interface usage; the record will for example contain elements (or attributes) which contain the pointer to the person B.

Materializing relationships between entities equates to hardcode “joins” between main entities into the index. Hence, the decision of which relationships are to be materialized is to be taken based on the search and navigation logic to be offered to end-users.

An index reflects the current status of authoritative entities and has therefore to be kept synchronized with the last changes to original entities and the enrichments of inference activities. Due to the size of the database, synchronization is performed incrementally, so as to include only changes occurred after the last synchronization operation. New or updated authoritative entities can be identified based on the time-stamp, while deleted entities are logically removed from the space and can therefore be identified.

Synchronization may involve millions of entities (e.g., refresh of UKPUBMED repository) and lead to database queries which may take hours, which slow down the database and the index. To cope with such numbers, synchronization is performed in different record tranches, where database is searched for entities which changed since the last commit, resulting entities are clustered based on a hash function which guarantees that no more than M index update actions are executed for each cluster, and each hash entity class is then applied to the index.