VirtualBrainCloud

Personalized Recommendations for Neurodegenerative Disease

www.VirtualBrainCloud-2020.eu

Public Deliverable Report

D3.14: Integrated, open source terminology versioning and management tool CTS2 compliant application shared with project partners
MS9: Integrated, open source terminology versioning and management tool CTS2 compliant application shared with partners

Date	May 2020
Authors	Mauro Giacomini, Sara Mora (UNIVERSITA DI GENOVA) Stephan Gebel, Sumit Madan (FRAUNHOFER) Petra Ritter (CHARITE)
Dissemination level	**public**
Website	www.VirtualBrainCloud-2020.eu

This project has received funding from the **European Union´s Horizon 2020** research and innovation programme under **grant agreement No 826421**

Table of content

1. Preface

A key objective of Task 3.4 "Workflows for clinical data curation and processing" and Task 3.5 "A metadata framework for unified metadata annotations and Data Catalogues" within the wider range of the Work Package 3: "Data Processing, Standardization and Data-FAIRification" is to deliver a semantic framework for neurodegenerative disease, that serves as a central resource for controlled vocabularies and shared ontologies within different TVB-Cloud work packages and includes relevant services.
Within TVB-Cloud two systems will be implemented and made available for the consortium members. The Ontology Lookup Service (OLS) from Fraunhofer Institute for Algorithms and Scientific Computing (SCAI) and the Common Terminology Service CTS2 approach from University of Genova (UNIGE).

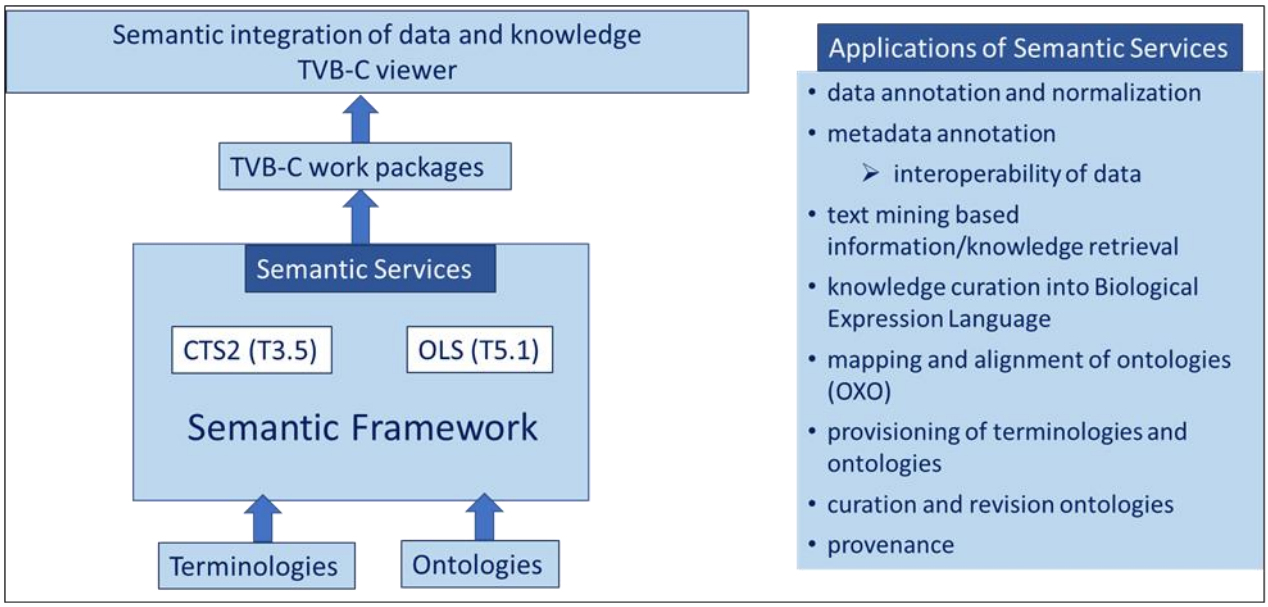

Figure 1: Schema of the Semantic Framework in TVB-Cloud

The two tools are both being implemented in the TVB-Cloud solution, one was mainly developed by Fraunhofer SCAI in the framework of the Human Brain Project (HBP) and the other - CTS2 engine - by the University of Genoa. The CTS2 engine has been implemented previously by Healthropy srl (spin-off of the University of Genoa). Within Virtual Brain Cloud project the University of Genoa developed an open source compliant interface that allows the use of CTS2 engine.

For these two tools, two demos are made available at the following URLs:

- OLS, the ontology lookup service software is being used to host the Referential Ontology Hub for Applications within Neurosciences (ROHAN), which is accessible at https://rohan.scai.fraunhofer.de/ols/index. (for more information see delivery 3.5.1: "Complete, updated semantic framework for neurodegeneration research documented").
- CTS2 compliant interface: a prototype of Read, Query and Maintenance HL7-CTS2 based tool is open to everyone in the TVB-Cloud consortium, controlled by a login system. At present, this access can be given to any member that wants to make a trial, by writing an email to (mauro.giacomini@dibris.unige.it). The URL address of this interface prototype is: http://www.medinfo.dibris.unige.it/VBC_CTS2. This application corresponds to the achieved Milestone MS9.

This project has received funding from the **European Union's Horizon 2020** research and innovation programme under **grant agreement No 826421**

2. Partners involved

CHARITE
AMU
CODEBOX
CODEMART
EODYNE
Fraunhofer
FZJ
UNIGE

3. Introduction

The CTS2 is a standard defined by Health Level 7 (HL7) and Object Management Group (OMG) and it is available at http://www.omg.org/spec/CTS2/1.2/. The objective of CTS2 is to provide technical and functional specifications for the development of service interfaces based on Service Oriented Architecture (SOA). CTS2 based interfaces allow the management, searching and reading of terminological content, which can be defined both locally and internationally (standardized vocabularies). The use of structured terminologies, made available thanks to this standard, makes it possible to fully support the management of semantics in the exchange of clinical information through independent organizations. In a context of shared semantics, CTS2 provides a modular, common and universally implementable set of operations (put, read, update and remove), which can then be used to manage a set of terminologies chosen by service users in their own distribution environment. CTS2 provides the terminology community with a defined set of standard interfaces that can be used to evaluate the structure, source content and terminology tools.

1. Provides a consistent specification to develop service interfaces to manage, search and access terminology content, either locally, or across a federation of terminology service nodes, independent of the terminology content and underlying technological stack.
2. Structured terminologies supply the basis for information interoperability by improving the effectiveness of information exchange within a specific domain.
3. Specifically, the structured terminologies provide a tool to organize information and to define the information semantics using consistent and computable mechanisms.
4. In a shared semantics environment, the CTS2 provides a modular and common set of behaviors which can be used to deal with a set of terminologies chosen by the clients.
5. The service contributes to interoperability by supporting an easy access to the foundational elements of shared semantics.

CTS2 is a collection of:

- Models: formal descriptions of what things are and how they are invoked.
- XML Schemas: sensible and accessible descriptions of the various resources.
- Rules for constructing URL's (or methods for SOAP/Object environment) to access this content.

4. CTS2 Terminology Resources

CodeSystem: a classification system or a code system or an ontology or a thesaurus, etc. with information about its publisher, release cycles, purpose, etc.

CodeSystemVersion: a version of a CodeSystem with information about the release date, release format, contact information, etc.

EntityDescription: a description about a class or a role or an individual in a specific CodeSystemVersion.

Map: a collection of rules for transforming entities of a CodeSystem into entities represented in a second one with information about the creators, intended use, CodeSystem involved, etc.

MapVersion: a version of a Map that carries the FROM and the TO CodeSystemVersion.

MapEntry: a definition of a set of rules that identifies how a single Entity that belongs to the FROM CodeSystemVersion maps TO ZERO or more target Entities that belong to the TO CodeSystemVersion.

5. Functional Profiles

Maintenance: set of capabilities to create, update, delete a terminology resource through sets of changes (ChangeSets), with specific properties to indicate the creation date, the user that changes the terminology content, the change information, the effective date, etc.

Read: set of capabilities to read a terminology resource in a specific context (e.g. language, time, changeset).

Query: set of capabilities to search the terminology resources with specific features in a certain context (e.g. language, time, changeset)

Import/Export: how to load and/or retrieve content in external formats.

History: set of capabilities to get the list of all the changes applied to a specific terminology resource and the list of all the ChangeSet applied to the whole terminology content.

6. Proposed Architecture

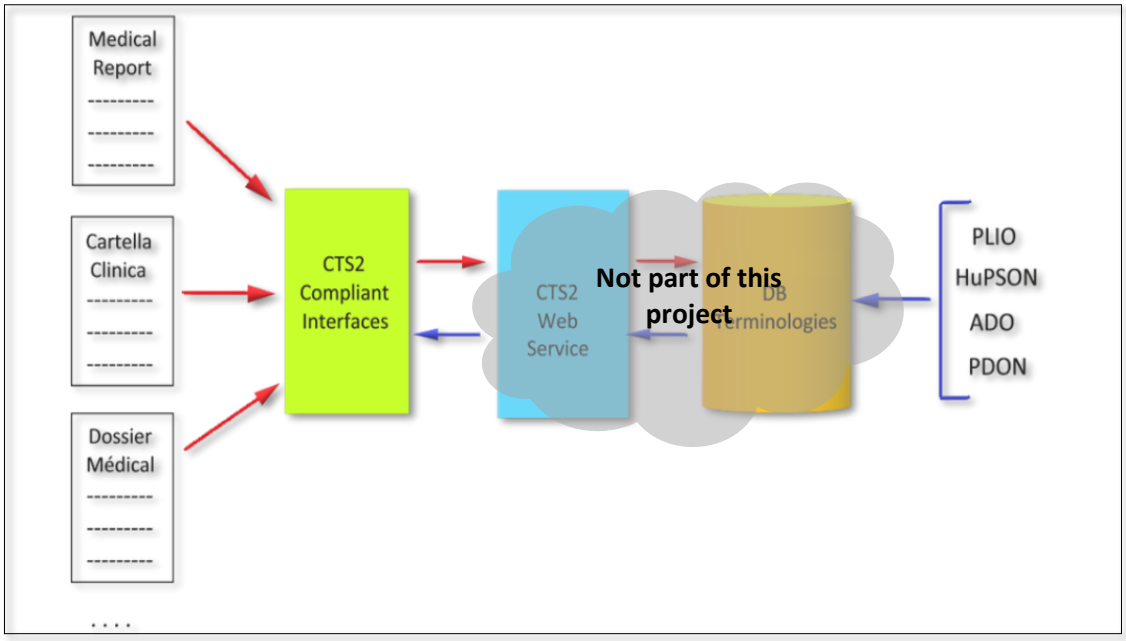

Figure 2: Proposed Architecture

According to the Grant Agreement (GA) the interface is open source and it can be used by everyone. The CTS2 web service and the database are not part of this project, but they're granted on loan for free use by Healthropy srl to the consortium members.

7. CTS2 Compliant Interface

The CTS2 compliant interface is realized as a Web ASP.NET Application (.NET Framework) of Visual Studio. This interface is connected to the HTS (Health Technology Service) powered by Healthropy srl and it uses in a standardized way his exposed functions to read, insert and update terminological contents into the database. The CTS2 compliant interface is composed by:

Home Page: Everyone can access this page, here some useful information about the CTS2 standard are given.

Figure 3: Home Page of the CTS2 compliant interface

5 Pages each specific for 1 ontology. The access to these sections is regulated through a login page, so that only authorized people that is working in that specific ontology can access the page. At present all interested TVB-Cloud members can gain access. During the development of this project we will insert this interface in the Virtual Brain Cloud Platform and we will define access policies.

7.1. Preliminary CTS2 operation used

In order to correctly insert the information of each specific ontology into the database the following CTS2 operation need to be used.

Create CodeSystem: The functional profile involved is the CodeSystem Catalog Maintenance Service. The input parameters are:

- → **ChangeSet**: The URI of an OPEN change set to record the changes (Type: ChangeSetURI).
- → **URI**: The about URI of the catalog entry to be created (Type: ExternalURI).
- → **Name**: The local identifier of the catalog entry in the context of the service (Type: CodeSystemName).

Return type is CodeSystemCatalogEntry, which has the properties listed in figure below.

It is necessary to create a CodeSystem for each of the ontologies involved in the project so: Alzheimer's Disease Ontology (ADO), Human Physiology Simulation Ontology (HuPSON), Protein-Ligand Interactions Ontology (PLIO) and Parkinson's Disease Ontology (PDON). Other CodeSystems will be created for each of the ontologies that are mapped to that specific ontologies of the project.

Figure 4: CodeSystemCatalogEntry Class

ChangeSet is a Unique Identifier (UI) of a set of change instructions that can potentially transform the contents of CTS2 service instance from one state to another. It is one of the Core Elements of CTS2 so it may be created by each of the functional profiles. In order to make a change through a maintenance service it is required the following sequence of operations:

a. Create a new Change Set
b. Make one or more changes to one or more resources, providing the URI of the created change set.
c. Update any additional provenance information on the change set
d. Commit change set to modify the database or Rollback change set in case of errors. Rollback deletes both the modifications of the specific change set and the change set itself.

Update CodeSystem: Update the contents of an existing catalog entry. Return type is a CodeSystemCatalogEntry and attributes that can be updated in an existing CodeSystemCatalogEntry are:

→ CodeSystemCategory: The category or type of resource that the code system represents.
→ OntologyDomain: The subject domain of the code system or ontology.
→ OntologyType: The nature of the content of the code system or ontology.
→ DesignedForOntologyTask: The purpose for which the code system or ontology was originally designed.
→ HasOntologyLanguage: The formal language used to express the code system or the ontology.
→ UsedOntologyEngineeringTool: The name or URI of an ontology engineering tool.

Create CodeSystemVersion: The functional profile involved is the CodeSystem Version Catalog Entry. The input parameters are:

→ ChangeSet
→ URI: The URI that uniquely identifies the new CodeSystemVersion (Type: DocumentURI)
→ Name: A name that uniquely identifies this version in the context of the implementation service (Type: CodeSystemVersionName).
→ SourceAndNotation: The source and notation used in this CodeSystemVersion (Type: SourceAndNotation).
→ VersionOf: The name or URI of the code system that this is a version of (Type: NameOrURI).

Return type is CodeSystemVersionCatalogEntry, which has the properties listed in Figure 3.

Figure 5: CodeSystemVersionCatalogEntry Class

Update CodeSystemVersion: Update an existing CodeSystemVersionCatalogEntry.

The input parameters are:

→ **ChangeSet**

→ **CodeSystemVersion**: The name or URI of the code system version to be updated (Type: NameOrURI).

→ **Request**: The parameters to be updated (Type: UpdateCodeSystemVersionRequest).

Return type is a CodeSystemVersionCatalogEntry and attributes that can be updated in an existing CodeSystemVersionCatalogEntry are:

a. **DefaultLanguage**: The name or URI of the default language used in this version.
b. **DocumentURI**: The URI of the specific document/format where this resource was retrieved.
c. **Imports**: The name or URI of a code system version whose content is imported and "asserted by" this CodeSystemVersion.

The UpdateCodeSystemVersionRequest function includes also attributes contained in the class Update Resource Version Description (one of the common model elements of the class Resource Maintenance):

d. **State**: The state of the resource version. Resource versions can change state from OPEN to FINAL, but not the other direction. Once a resource version description is finalized and committed, it becomes immutable.
e. **SourceAndNotation**: A description of where the source of the version may be found, what format and language it is available in.
f. **Predecessor**: The name or URI of the resource version that immediately preceded this version on an evolutionary path.
g. **OfficialResourceVersionId**: An official label or identifier that was assigned to this version by its publisher.
h. **OfficialReleaseDate**: Information about the source, format, release date, version identifier, etc. of a specific version of an abstract resource.
i. **OfficialActivationDate**: The date that this version of the resource is stated by its publishers to go into effect.

Create Named Entity Description / Create Predicate Description / Create Object Property Description: The functional profile involved is the Entity Description Maintenance Service. The input parameters are:

→ **ChangeSet**

→ **EntityID** (optional): The entity code and/or namespace identifier of the entity to be created (Type: ScopedEntityName).

→ **DescribingCodeSystemVersion**: The URI or local identifier of the CodeSystemVersion that this entity description is both described in and described by (Type: NameOrURI).

Return type is NamedEntityDescription.
In Figure 4 is shown an extract of the entities that is necessary to create in order to correctly manage the ADO ontology.

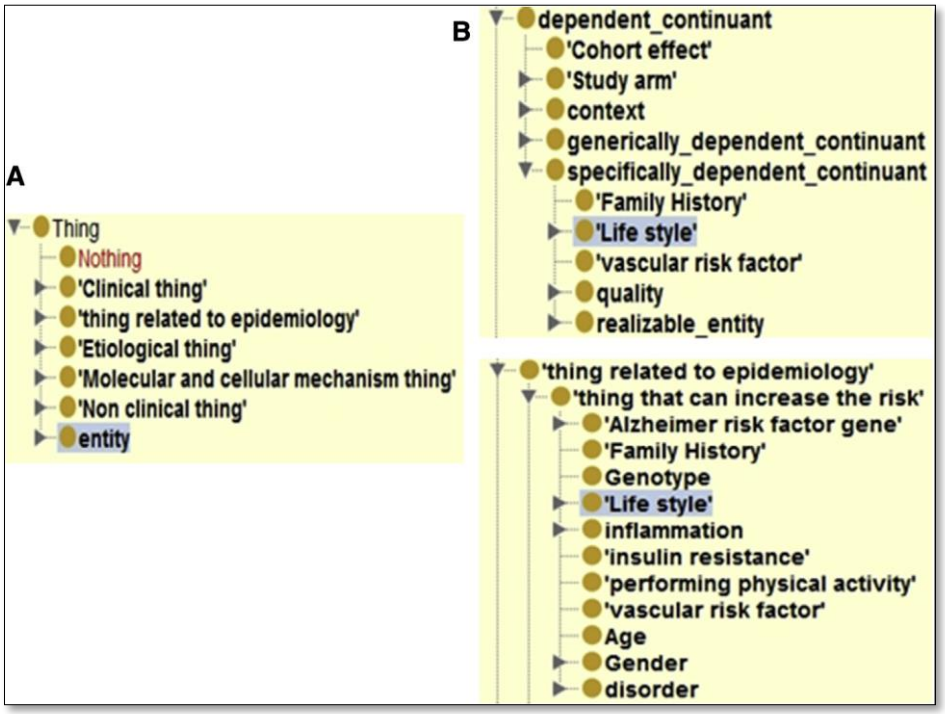

Figure 6: ADO Classes

Update Entity Description: Once Entities have been created, they can be updated to add attributes. The most important that will be used to add specific properties to each element in the ontologies are:

→ **AlternateEntityID**: Alternative unique identifiers that reference the about entity in the context of describing CodeSystemVersion. The "namespace" names the context from which the name (or code) is derived.
→ **Definition**: An explanation of the intended meaning of a concept. An EntityDescription may have multiple definitions, each derived from a different source, represented in a different language or having a different purpose.
→ **Example**: An example of an instance or instances of the referenced entity, typically written for human consumption.
→ **Note**: A note or comment about the history, status, use, or other description about the EntityDescription.
→ **Property**: Additional "non-semantic" (annotation) assertions about the entity being described that do not fit into the other categories.
→ **SourceStatements**: A DirectoryURI that resolves to the list of statements that were used in the assembly of this description.
→ **Parent**: The set of direct "parents" asserted by DescribingCodeSystemVersion. It is the responsibility of the service to determine what predicate(s) represent "parent/child" relationships. Typically "parent" is associated with "rdfs:subClassOf" in the OWL/RDF world.

- → **Parents**: A DirectoryURI that resolves to the list of direct "parents" asserted by DescribingCodeSystemVersion. This returns the parent elements above.
- → **Ancestors**: A DirectoryURI that resolves to the transitive closure of the "parents" relationship(s). The primary purpose for this attribute is to provide a handle for subsumption queries. As an example, to determine whether Class X was a subclass of Class Y, one would query whether the EntityReference to Y was a member of X.ancestors.
- → **Children**: A DirectoryURI that resolves to the list of direct "children" asserted by DescribingCodeSystemVersion. As with parent, it is the responsibility of the service to determine what predicate(s) represent "parent/child" relationships.
- → **Descendants**: A DirectoryURI that resolves to the transitive closure of the "children" relationship(s). The primary purpose for this attribute is to provide a handle for subsumption queries. As an example, a second way to determine whether Class X was a subclass of Class Y, one would query whether the EntityReference to X was a member of Y.descendants.
- → **EntityType**: The set of type(s) which the EntityReference is an instance of. Because this is a terminology service, EntityType must include one of owl:Class, owl:Individual, rdf:Property, or skos:Concept, although it may carry many other types as well.
- → **Instances**: A DirectoryURI that resolves to the list of entities that asserted to be instances of the type represented by about URI. This element is present only if resolution will return a non-empty set.
- → **EquivalentEntity**: An entity that has been determined to be equivalent to the about entity in the context of the assertions made by DescribingCodeSystemVersion.

The first attribute that should be considered is the **EntityType**. An example can be found in the ADO ontology which has both:

a. **Object Property**:

```
<!-- http://scai.fraunhofer.de/NDDUO#adjacent_to -->

<ObjectProperty rdf:about="http://scai.fraunhofer.de/NDDUO#adjacent_to">
    <rdfs:domain rdf:resource="http://www.ifomis.org/bfo/1.1/snap#Continuant"/>
    <rdfs:range rdf:resource="http://www.ifomis.org/bfo/1.1/snap#Continuant"/>
    <dc:source>OBO_REL:adjacent_to</dc:source>
    <rdfs:comment>Note that adjacent_to as thus defined is not a symmetric relation,
    in contrast to its instance-level counterpart.
    For it can be the case that Cs are in general such as to be adjacent to instances
    of C1 while no analogous statement holds for C1s in general in relation to instances of C.
    Examples are: nuclear membrane adjacent_to cytoplasm; seminal vesicle adjacent_to urinary bladder;
    ovary adjacent_to parietal pelvic peritoneum</rdfs:comment>
    <rdfs:isDefinedBy>C adjacent to C&apos;
    if and only if: given any instance c that instantiates C at a time t,
    there is some c&apos; such that: c&apos; instantiates C&apos;
    at time t and c and c&apos; are in spatial proximity</rdfs:isDefinedBy>
</ObjectProperty>
```

Figure 7: Example of Object Property from ADO

b. **Class**:

```
<!-- http://scai.fraunhofer.de/AlzheimerOntology#Abeta_42 -->

<Class rdf:about="http://scai.fraunhofer.de/AlzheimerOntology#Abeta_42">
    <rdfs:subClassOf rdf:resource="http://scai.fraunhofer.de/AlzheimerOntology#amyloid_beta_protein"/>
    <rdfs:subClassOf>
        <Restriction>
            <onProperty rdf:resource="http://scai.fraunhofer.de/NDDUO#has_role"/>
            <someValuesFrom rdf:resource="http://scai.fraunhofer.de/AlzheimerOntology#Blood_Biomarker"/>
        </Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <Restriction>
            <onProperty rdf:resource="http://scai.fraunhofer.de/NDDUO#has_role"/>
            <someValuesFrom rdf:resource="http://scai.fraunhofer.de/AlzheimerOntology#Cerebospinal_fluid_Biomarker"/>
        </Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <Restriction>
            <onProperty rdf:resource="http://scai.fraunhofer.de/NDDUO#is_entity_used_in"/>
            <someValuesFrom rdf:resource="http://scai.fraunhofer.de/NDDUO#procedures_using_biomarkers_context"/>
        </Restriction>
    </rdfs:subClassOf>
    <metadata:prefixIRI rdf:datatype="http://www.w3.org/2001/XMLSchema#string">AlzheimerOntology:Abeta_42</metadata:prefixIRI>
    <NDDUO:Synonym>A42</NDDUO:Synonym>
    <NDDUO:Synonym>abeta</NDDUO:Synonym>
    <NDDUO:Synonym>amyloid beta 42</NDDUO:Synonym>
    <NDDUO:Synonym>amyloid-(42)</NDDUO:Synonym>
    <rdfs:comment>AD109</rdfs:comment>
    <rdfs:comment>AD113</rdfs:comment>
    <rdfs:label>Abeta 42</rdfs:label>
</Class>
```

Figure 8: Example of Class from ADO

As we can see in the examples both types of entities have properties. The difference between the NamedEntityDescription created with the three before mentioned operation is mainly the EntityType.

- Create Named Entity Description.EntityType = owl:Named.
- Create Predicate Description.EntityType = rdf:Property. This type can be used for example to identify the Annotation Properties.
- Create Object Property Description.EntityType has count =2 in this case: owl:ObjectProperty and rdf:Property. This type can be used for example to identify the Object Properties.

The other attributes that should be analyzed more in detail are the **Class** ones:

→ **Predicate**: The name or URI of the property predicate.

→ **Value**: The target(s) of the property. Note that this can only represent the literal format of the property. The details about the original property will be found in the CorrespondingStatement if the CTS2 implementation supports the statement profile.
So, the attribute value of a property is of Class Statement Target and it can be of three types:

- *Literal Target*: when the statement type is LITERAL. It can be used for that properties like the ID of the entity.
- *Entity Reference Target*: the URI and optional namespace/name when the target type is ENTITY. It can be used when a property refers to another entity, for example of type owl:ObjectProperty.
- *Resource Target*: when the statement type is RESOURCE.

An entity may have more than 1 value for the same predicate, in this case it is necessary to create a List of StatementTarget containing both items and then assign it to property.value, while property.predicate remains unchanged.

→ **CorrespondingStatement**: a link to the original statement from which this Property is derived. Will only be present in CTS2 implementations profile.

→ **PropertyQualifier**: An assertion whose subject is the assertion in the property instead of the property subject.

Below are reported some practical examples of the use of properties:

Literal Target:

1. Single Literal Target:

```
<!-- http://scai.fraunhofer.de/AlzheimerOntology#Dalman -->

<owl:Class rdf:about="&AlzheimerOntology;Dalman">
    <rdfs:label>Dalman</rdfs:label>
    <rdfs:subClassOf rdf:resource="&AlzheimerOntology;drug_used_in_treatment"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&NDDUO;has_role"/>
            <owl:someValuesFrom rdf:resource="&AlzheimerOntology;Antipsychotic_drugs"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment>AD007</rdfs:comment>
    <NDDUO:Synonym>Flurazepam</NDDUO:Synonym>
</owl:Class>
```

Figure 9: Example of a Single Literal Target from ADO

2. More than 1 Literal Target:

```
<!-- http://scai.fraunhofer.de/AlzheimerOntology#Dapsone -->

<owl:Class rdf:about="&AlzheimerOntology;Dapsone">
    <rdfs:label>Dapsone</rdfs:label>
    <rdfs:subClassOf rdf:resource="&AlzheimerOntology;drug_used_in_treatment"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&NDDUO;has_role"/>
            <owl:someValuesFrom rdf:resource="&AlzheimerOntology;Nonsteroidal_Antiinflamatory_drugs"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <NDDUO:Synonym>DAPSONE</NDDUO:Synonym>
    <rdfs:comment>AD048</rdfs:comment>
    <NDDUO:Synonym>4,4&apos;-sulfonyldianiline</NDDUO:Synonym>
    <NDDUO:Synonym>Dapsone</NDDUO:Synonym>
    <NDDUO:mapping>http://purl.obolibrary.org/obo/CHEBI_4325</NDDUO:mapping>
</owl:Class>
```

Figure 10: Example of more than 1 Literal Target from ADO

Entity Reference Target:

1. Single Entity Reference Target:

```
<!-- http://scai.fraunhofer.de/NDDUO#bearer_of -->

<owl:ObjectProperty rdf:about="&NDDUO;bearer_of">
    <rdfs:isDefinedBy>A relation between an entity and a dependent continuant; the reciprocal relation
        of inheres_in</rdfs:isDefinedBy>
    <rdfs:comment>comment: Examples: red eye bearer_of redness
        ! range: snap:DependentContinuant</rdfs:comment>
    <dc:source>OBO_REL:bearer_of</dc:source>
    <rdfs:comment>This relation links a bearer to its qualities, functions, dispositions and other
        dependent continuants.</rdfs:comment>
    <rdfs:range rdf:resource="&snap;DependentContinuant"/>
</owl:ObjectProperty>
```

Figure 11: Example of a Single Entity Reference Target

2. More than one Entity Reference Target:

```
<!-- http://www.ifomis.org/bfo/1.1/snap#DependentContinuant -->

<owl:Class rdf:about="&snap;DependentContinuant">
    <rdfs:label rdf:datatype="&xsd;string">dependent_continuant</rdfs:label>
    <owl:equivalentClass>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&snap;GenericallyDependentContinuant"/>
                <rdf:Description rdf:about="&snap;SpecificallyDependentContinuant"/>
            </owl:unionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&snap;Continuant"/>
    <owl:disjointWith rdf:resource="&snap;IndependentContinuant"/>
    <owl:disjointWith rdf:resource="&snap;SpatialRegion"/>
    <rdfs:comment rdf:datatype="&xsd;string">Definition: A continuant [snap:Continuant] that is
        either dependent on one or other independent continuant [snap:IndependentContinuant] bearers
        or inheres in or is borne by other entities.</rdfs:comment>
</owl:Class>
```

Figure 12: Example of more than one Entity Reference Target

The input parameters of the function Update Entity Description are:

→ ChangeSet
→ Request: The set of fields to be changed (Type: UpdateEntityDescriptionRequest).
→ EntityID
→ DescribingCodeSystemVersion

Below are detailed some choices examples of CTS2 attributes that have been made to insert the ontology information in the database.

Attributes that cannot be modified and must be chosen during the creation phase:

- Entity.EntityID.Name = source name : class name. For example NDDUO:Clinical. It was not possible to choose only the class name because it was not unique within the different sources, an example of that is Outcome.
- Entity.EntityType = “owl:Named”. It can’t be modified because it is a consequence of the function that generates the NamedEntityDescription.

Attributes that can be changed through the Update Entity Description Request:

- Entity.Definition = “isDefinedBy”
- Entity.Note = “comment”
- Entity.Example = “example”

  (For each of them can be specified the language for example.)

- Entity.Parent = “subClassOf”
- Entity.Property = Any other property of the class, for example “Synonym”, “label”, “disjoinWith”, etc.

7.2. Functional Profile Read

Classes:

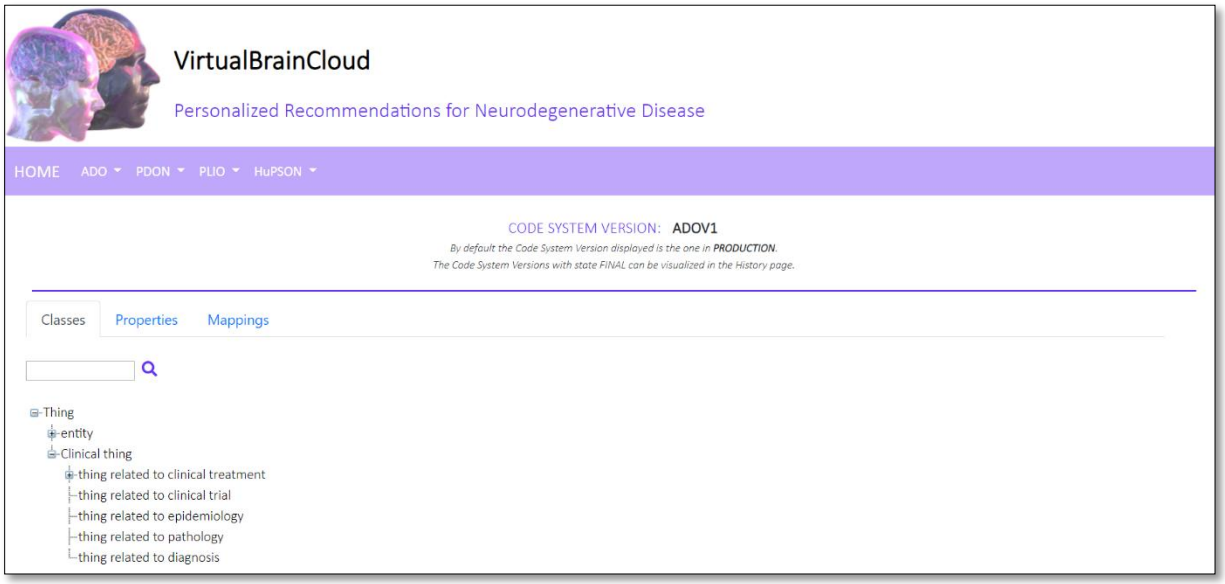

Figure 13: Example of the tree visualization of ADO Classes

Properties:

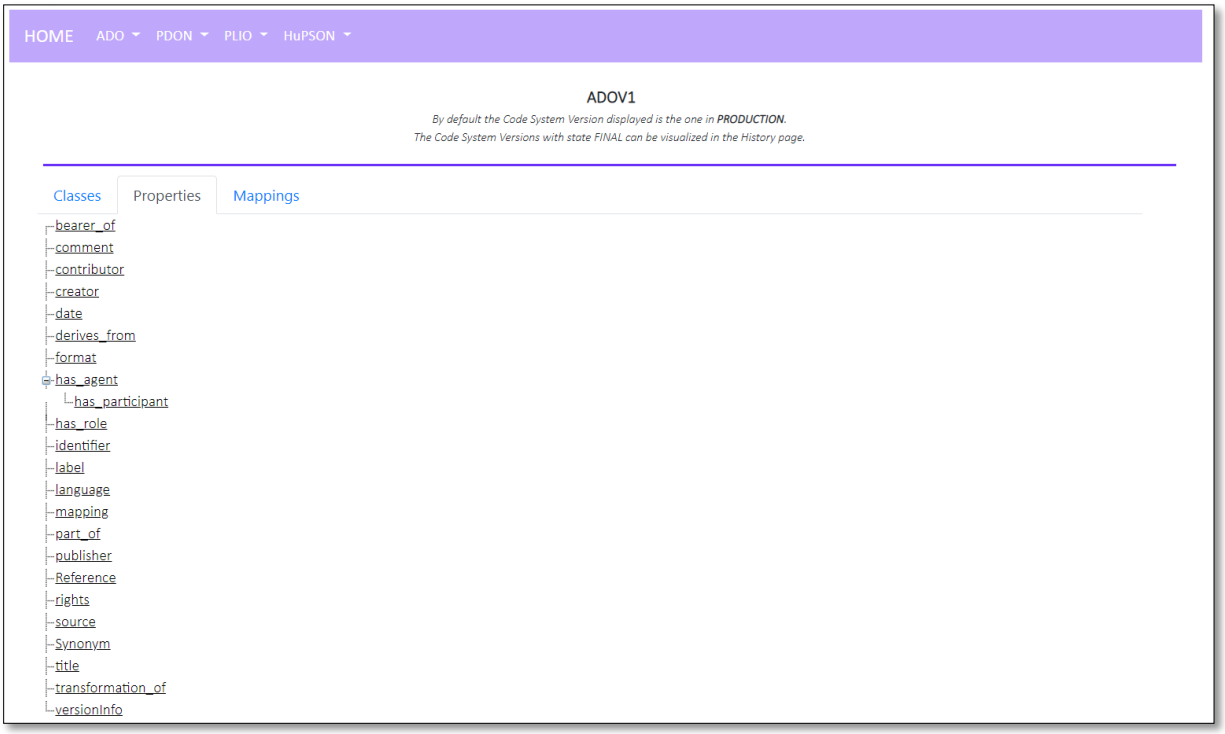

Figure 14: Example of the tree visualization of ADO Properties

The first part of this page of the interface let the user visualize the entities of a specific CodeSystemVersion. The CTS2 operations that are needed to reach this task are different depending on the state of the CodeSystemVersion. It is necessary to remember that if the state is FINAL the version is closed it can't be modified anymore, otherwise if it is OPEN then it can be modified.

The versions of a specific CodeSystem may also be identified by these Version Tags:

- Production: the CodeSystemVersion with state "open" (it will be only one)
- Current: the last CodeSystemVersion with state "final"
- Previous: the CodeSystemVersion with state "final" immediately before the current
- Obsolete: all the other CodeSystemVersions with state "final" before the previous

The list of version tags can be obtained using the function:

Get Supported Tag: Function that returns the set of version tags known to the service. All services must recognize the CURRENT tag, although the interpretation is service specific.

By default, the interface will show the CodeSystemVersion with version tag "production", so the one with state "open". Then if the user wants to have a look to the previous versions, he will be able to do that in the history page. In order to get the last version, the following operation should be used:

Get Code System Version for Code System: Retrieve the specific CodeSystemVersion details for the specified CodeSystem from the service. The input parameters are:

→ **CodeSystem**
→ **Tag:** The version tag of the resource. Defaults to the URI for CURRENT if not supplied (Type: NameOrURI).

→ QueryControl
→ Context

Return type is CodeSystemVersionCatalogEntry.

Otherwise, if the user wants to get the list of the previous versions of that specific CodeSystem it can be done thanks to the DirectoryURI that references the known versions of the specific CodeSystem and it can be found in the attribute “versions” of the specific CodeSystem.

For each CodeSystemVersion, a TreeView containing the entities is shown. Entities are grouped in levels of deepness so the first that is presented is the parent of all the others, then the user can visualize children of each parent. To get the list of all entities of a specific CodeSystemVersion the following operations have to be used in series:

Get All Entities: Function that returns an URI that ends with EntityDescription?Id[ ]=All

Restrict to Code System Version: Function used to restrict the research to the entities of a specific CodeSystemVersion. The input parameters are:

→ Directory: A DirectoryURI that resolves to a set of EntityReferences (Type: EntityDirectoryURI). In this case it needs to be valued with the URI obtained as a result of the function “get all entities”.
→ CodeSystemVersion: The name or URI of a code system version used to restrict the set of entities in directory (Type NameOrURI).

Return type is EntityDirectoryURI.

Resolve as List: Returns an Entity List containing the set of EntityDescriptions identified by the EntityDirectoryURI obtained at the previews step. The input parameters are:

→ Directory: A directoryURI that resolves a set of EntityReferences (Type EntityDirectoryURI).
→ ResolveOnlyCodeSystem: The name or URI of a Code System. If absent all the elements and attributes will be returned.
→ QueryControl: It is a parameter that controls the ordering, timeout and query behavior.
→ Context: It is a parameter that controls the language, date, time and other contextual variables.
→ ReturnFilter: A list of zero or more component references. The returned list of entities will contain only the required elements and attributes plus any elements or attributes named in the filter.

To visualize the details of specific entity it is necessary to click on the specific TreeNode. In order to fill the details panel with all the information of the specific Entity it is necessary to use the following operation:

Restrict: This function returns a DirectoryURI that references the set of all elements represented by directory that match the criteria specified in filter. The input parameters are:

→ Directory
→ MatchAlgorithm (optional): The name or URI of the match algorithm to use when selecting values. The default value if the parameter is not supplied is “CONTAINS” - the supplied match value appears anywhere in the target. (Type: NameOrURI). The list of supported matchalgorithms can be obtained by using the function:

Get Supported Match Algorithm: A Match Algorithm is a predicate that determines whether an entity resource quality for membership in a set based on supplied matching criteria. Example of match algorithms are “equals” and “contains”.

→ **MatchValue** (optional)**:** The value to be used in comparison. The structure and format of matchValue depends on the specific matchAlgorithm. As an example, a “startsWith” algorithm would be plain text, a “regularExpression” algorithm would have a regular expression, while an “exists” algorithm would have nothing in the matchValue argument. (Type: String)

→ **FilterComponent** (optional)**:** The name or URI of a property or model element to be filtered. If omitted, all properties are searched. (Type: EntityNameOrURIList)

The restrict function in this context is used to find the entity whose name is the same of the string specified by the user. So, the following choices should be done:

- Directory: the URI of the specific CodeSystemVersion, in example the ADOV1
- MatchAlgorthms: in this case the “equals” is required
- MatchValue: the string specified by the user
- FilterComponent: it should be built by using the:

  Get Supported Model Attribute: The set of model attributes that can be referenced in filter instances for the given service implementation. In this case it is necessary a model attribute that limit the research on the entityIDName.

In the left-top part of the page there is a textbox and a search button, this way the user is given the possibility to search for a SPECIFIC ENTITY in the TreeView which will automatically expand to show that item.

In figure below is represented a sample of the class details visualization.

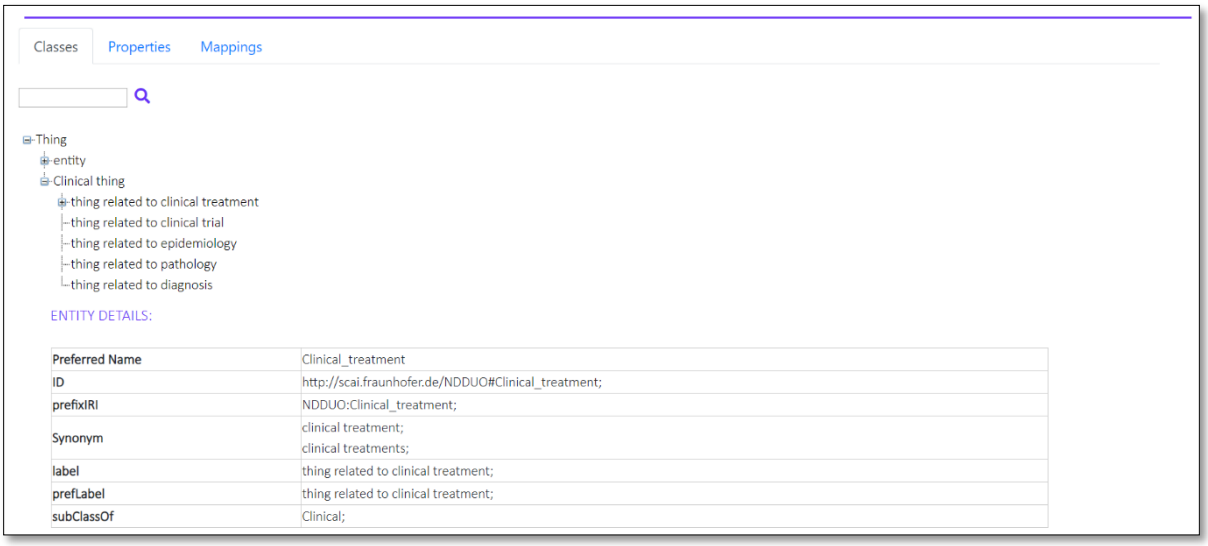

Figure 15: Class Details Visualization

7.3. Functional Profile Query

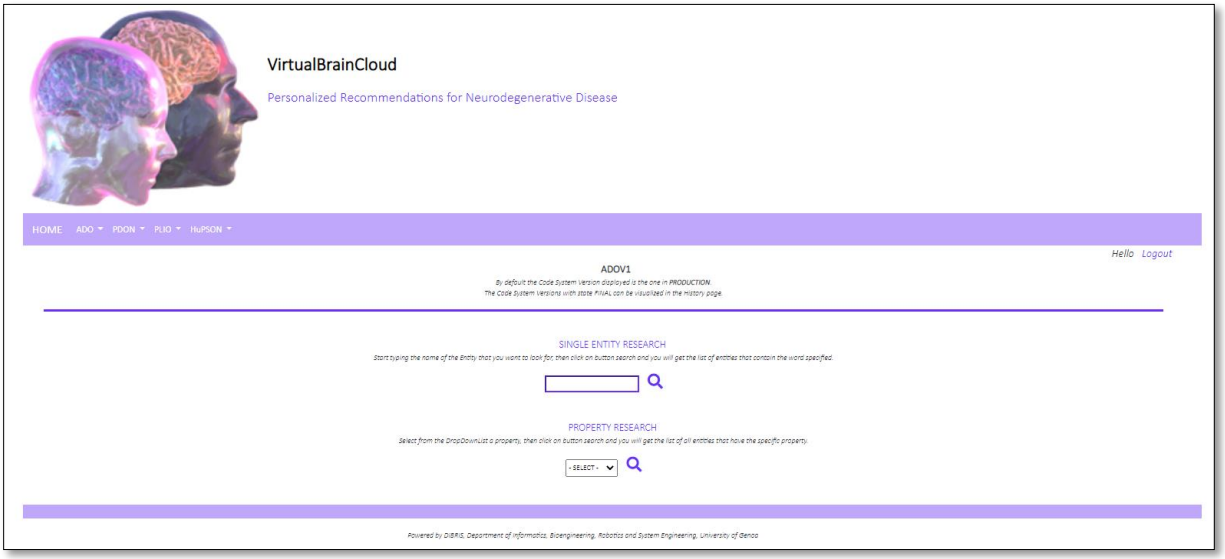

Figure 16: Functional Profile Query Interface

This page gives the user the possibility to select a subset of all the EntityDescriptions of a specific CodeSystemVersion via various restriction criteria, the intersection, union and difference of these sets and the ability to render the result of these queries in different formats. It also provides tools to determine the presence or absence of EntityReferences in these sets.

7.3.1. Single Entity Research

This first operation gives the user the possibility to get the list of all the classes that contains a certain word. To reach this goal the functions that need to be used are the same of the read process, but the very important change is in the input parameters of the **Restrict** function.

In this case the needed match algorithm is the “contains” one.

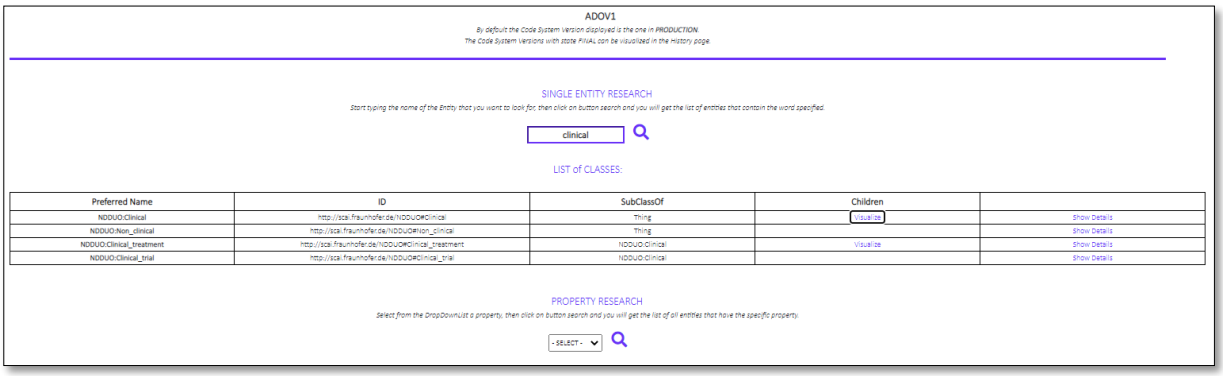

Figure 17: Example of Single Entity Research

If the user wants to see all the details of that specific entity, he can click on the buttonfield “Show Details” of the gridview, this way a panel with all the information will appear at the top of the page:

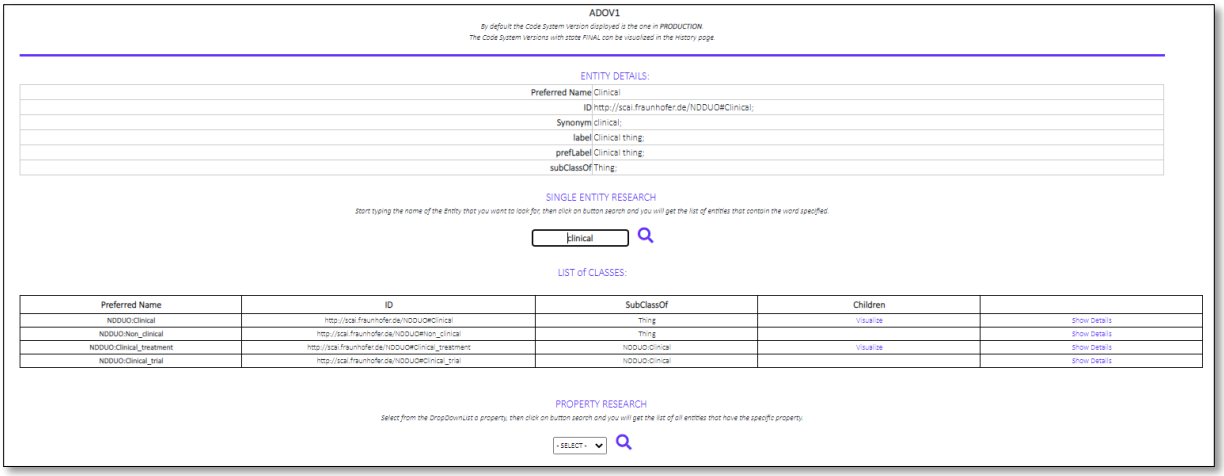

Figure 18: "Show Details" panel

Otherwise if the user wants to see all the children of that specific entity (if present) he can click on the button “Visualize” in the column “Children”.

7.3.2. Property Research

This second operation gives the user the possibility to find all the entities that have a specific property, in example a definition, a comment, an example.

Figure 19: Property Research example

Even in this case it is possible to click on “Show Details”.

7.4. Functional Profile Maintenance

This section contains the functions that allow the user to modify the CodeSystem. The list of functions is:

1. Insert a new entity
2. Update an existing entity
3. Set entity status to INACTIVE or delete an entity
4. Create a new Code System Version
5. Update an existing Code System Version
6. Close an existing Code System Version

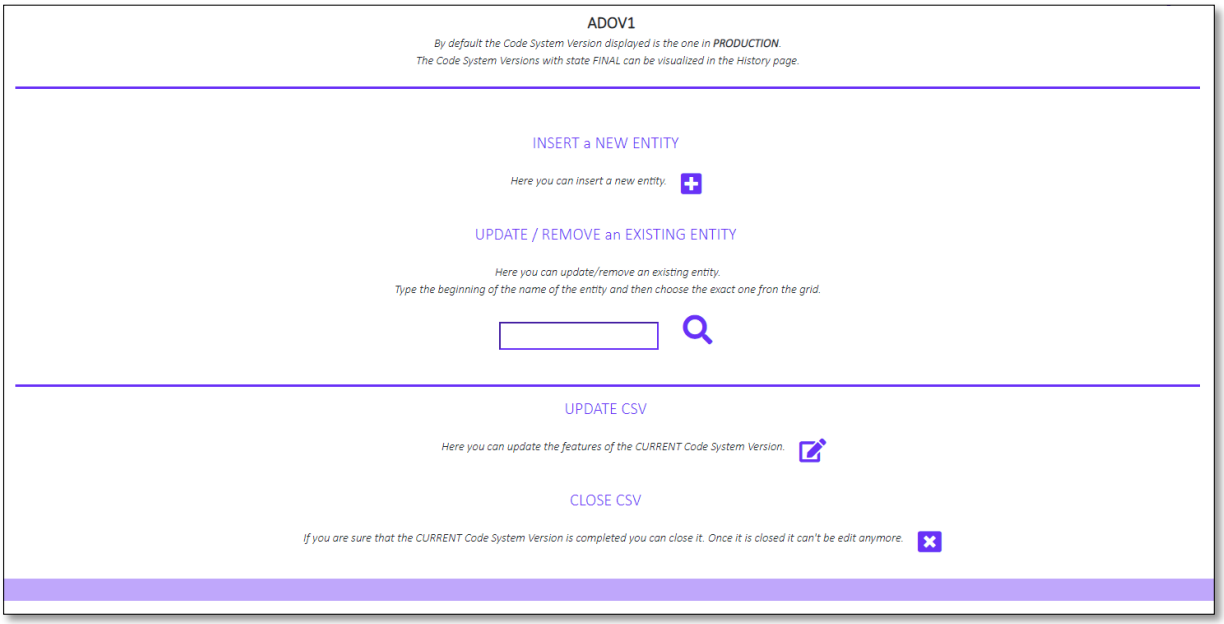

Figure 20: Functional Profile Maintenance

7.4.1. Insert a new entity

The complete set of operations that should be executed to insert a new entity have been described in detail in the previous paragraphs. To insert a new entity, it is necessary to determine the CodeSystem and CodeSystemVersion that it belongs to, its source, its name and its parent (if it has one, so this argument is optional). This way the entity will immediately appear in the TreeView.

User should press on the icon plus to open the specific panel. It is necessary to insert ALL the information that are required (*) in the panel, otherwise an error will appear, and the button save will be disabled. CodeSystem and CodeSystemVersion are determined by the system, the first one as a consequence of the ontology page where the user is accessed and the second one is the CURRENT version. The source is necessary because the entity name may not be unique in the entire ontology, but it is unique in its own source. Once ALL the mandatory information (*) is set the user can press button SAVE.

NB: These are the attributes that could not be changed in the future (except for the parent one), so if the user inserts something wrong, he must delete the entity and insert it again, that because of the rules of the CTS2 standard.

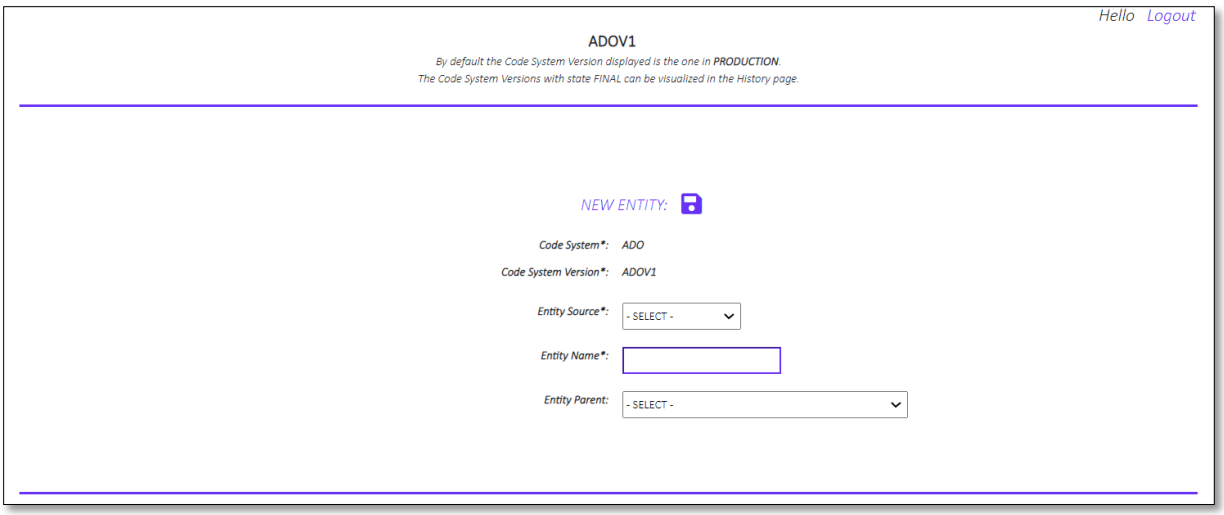

Figure 21: Insert new entity

7.4.2. Update an existing entity

The complete set of operations that should be executed to update an existing entity have been descripted in detail in the previous paragraphs. In order to reach the update section, the user has to choose the entity he wants to update typing the beginning of the name in the textbox, find the desired entity in the gridview and then click on "Show Details" in the corresponding gridview row. Once the

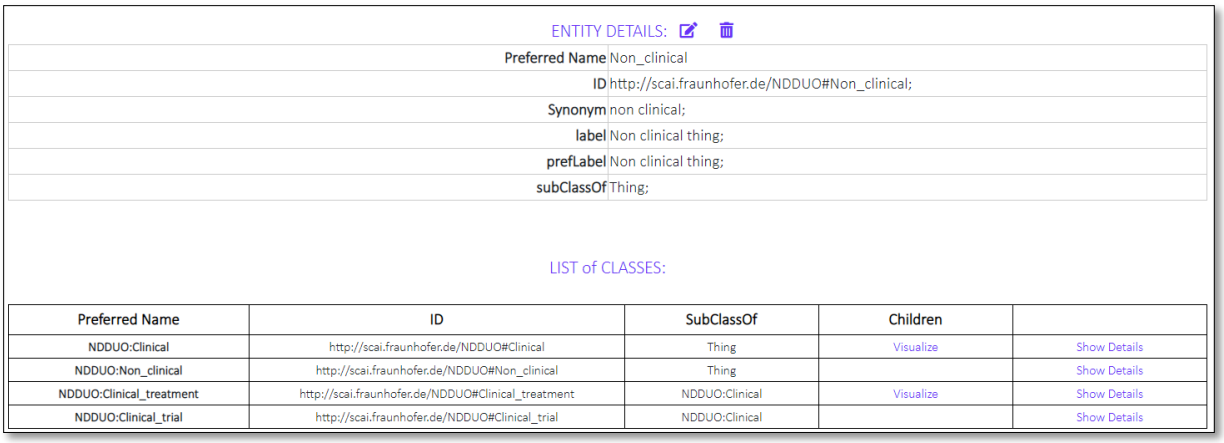

Figure 22: Update / Remove entity

entity details panel is open two icons appear: **edit** and **remove**.

If the user clicks on the edit icon the specific panel will appear. As we can see in figure below, in the first part the user can update the attributes and the literal properties of the entity while in the second part he can update the object properties. At the moment a maximum of three values can be assigned for each property but in the future, it can be extended, if necessary.

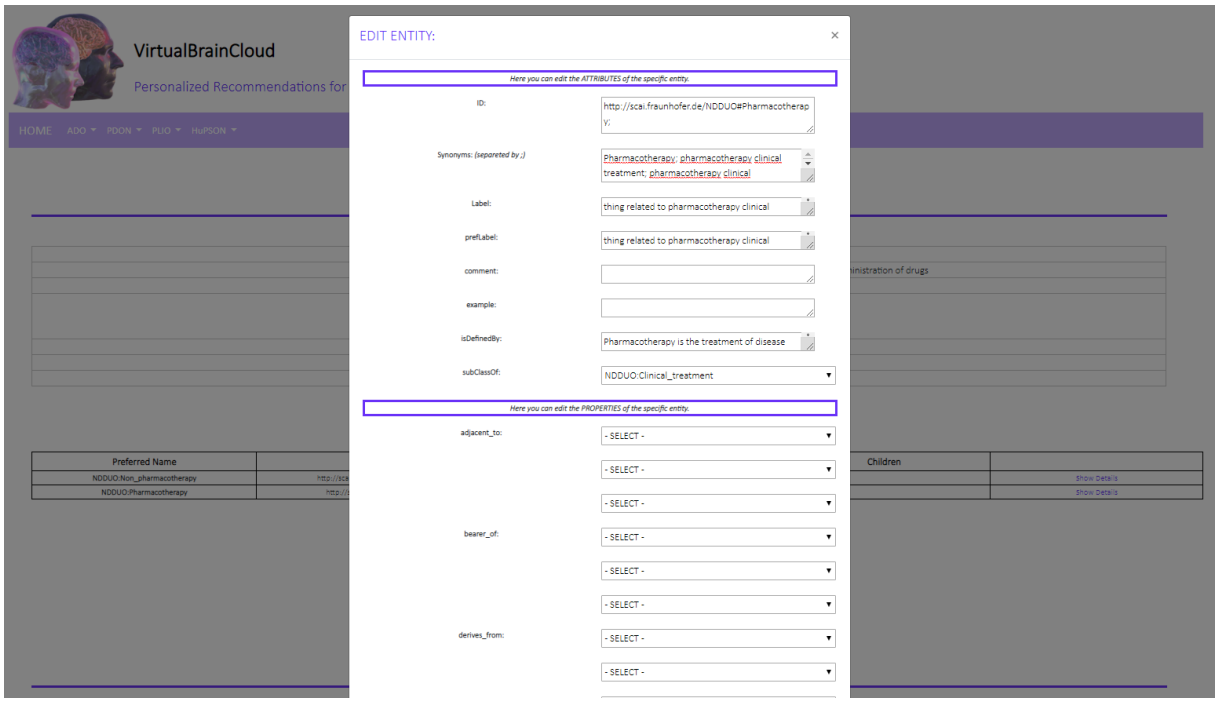

Figure 23: Modal edit single entity

7.4.3. Set entity status to INACTIVE or delete an entity

If the user wants to remove an entity from the list of entities that can be used of a specific CodeSystemVersion two options are proposed: the first one that should be used if an entity is no more valid for example from one version to the next one; the second if the entity doesn't belong to the specific CodeSystemVersion because it has been added wrongly.

1. Each entity has the property "EntryState" that can be valued with ACTIVE or INACTIVE. To change the status from ACTIVE to INACTIVE the following function should be used:

   Update Changeable Metadata: Update the metadata for a change set and return an image of the updated resource. This operation may alter the state of the service. The input parameters of this function are:

   → **ChangeSet**
   → **EntryID:** The entryID of the resource whose state is to be updated. (Type: PersistentURI)
   → **Status** (optional)**:** The name or URI of valid workflow status. (Type: NameOrURI)
   → **EntryState** (optional)**:** The new state of the resource (ACTIVE or INACTIVE). (Type: EntryState)
   → **EffectiveDate**: The new effective date of the resource. (Type: DateAndTime)
   → **ChangeNotes**: The new change notes of the resource. (Type: OpaqueData)
   → **ChangeSource**: The new change source of the resource. (Type: SourceReference)

   All the input parameters Status, EntryState, EffectiveDate, ChangeNotes, ChangeSource constitute an element of type **UpdateChangeableMetadataRequest**.

   Return Type of the function is: Changeable.

2. The operation which is used to physically delete an entity is:

Delete Changeable: Remove the changeable resource from the service. The input parameters are:

→ **ChangeSet**

→ **ChangeableResource:** The image of the resource to remove. (Type: Changeable)
In this case the resource that needs to be removed is a NamedEntityDescription.

There is no return type in this case.

The specific entity can be deleted by clicking on the icon of trash and this operation is irreversible.

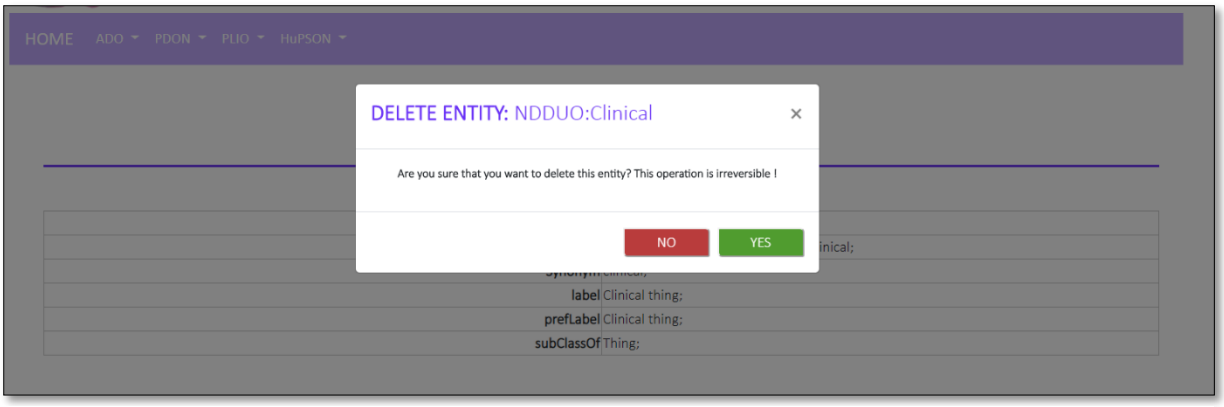

Figure 24: Modal delete entity

The status can be modified trough the update panel that can be opened by clicking on the edit icon.

7.4.4. Create a new CodeSystemVersion

The complete set of operations used to reach this goal have been described in detail in the previous paragraphs. If the user wants to create a new CodeSystemVersion, it is necessary to set to FINAL the state of CURRENT one. This operation is available only when there is no CodeSystemVersion open.

7.4.5. Update an existing CodeSystemVersion

The complete set of operation that should be executed and the attributes that can be updated have been descripted in detail in the previous paragraph.

7.4.6. Close an existing Code System Version

This operation is a subclass of the Update because to close a CodeSystemVersion it is necessary to Update it and set the attribute “state” to FINAL. It is a simplification to allow the user to perform the operation more easily.

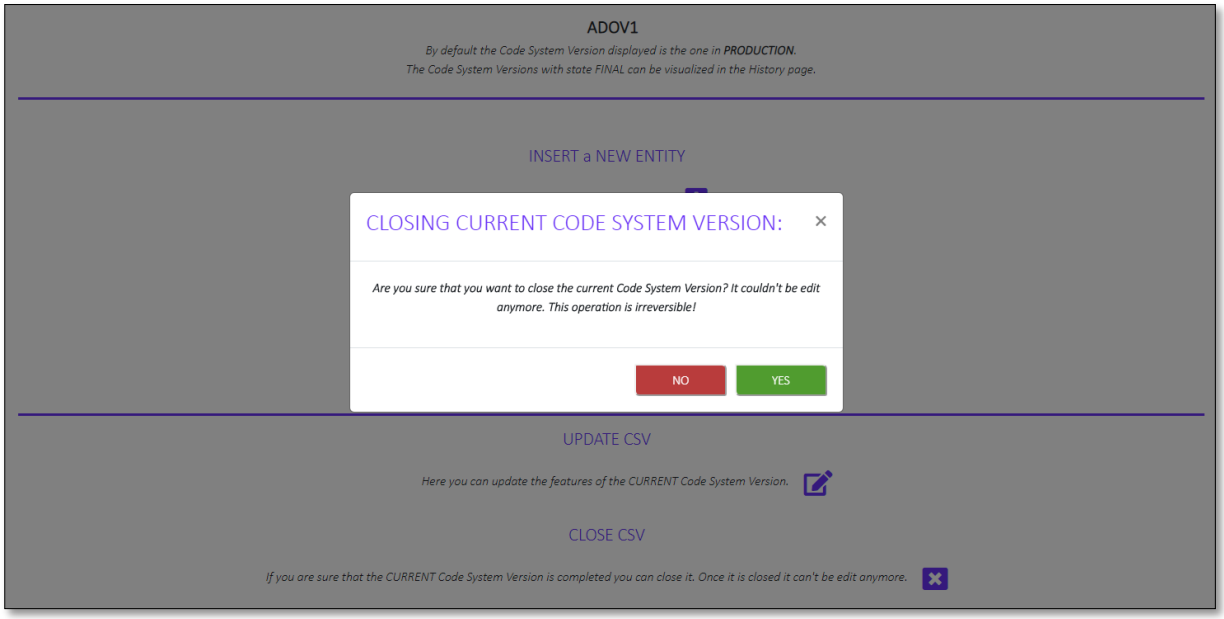

Figure 25: Modal close CSV

8. Glossar

ADO - Alzheimer's Disease Ontology

ASP.NET - Active Server Pages terminology using .NET languages

CSV – Code System Version

CTS2 – Common Terminology Service Release 2

DB – Database

GA - Grant Agreement

HBP – Human Brain Project

HL7 - Health Level 7

HTS - Health Technology Service

HuPSON – Human Physiology Simulation Ontology

OLS - The Ontology Lookup Service

OMG – Object Management Group

OWL – Web Ontology Language

PLIO – Protein–Ligand Interaction ontology

PDON - Parkinson's Disease Ontology

RDF – Resource Description Framework

ROHAN - Referential Ontology Hub for Applications within Neurosciences

SCAI - Fraunhofer Institute for Algorithms and Scientific Computing

SOA - Service Oriented Architecture

SOAP – Simple Object Access Protocol

UI - Unique Identifier

UNIGE - University of Genoa

URI - Uniform Resource Identifier

URL – Uniform Resource Locator

XML – Extensible Markup Language