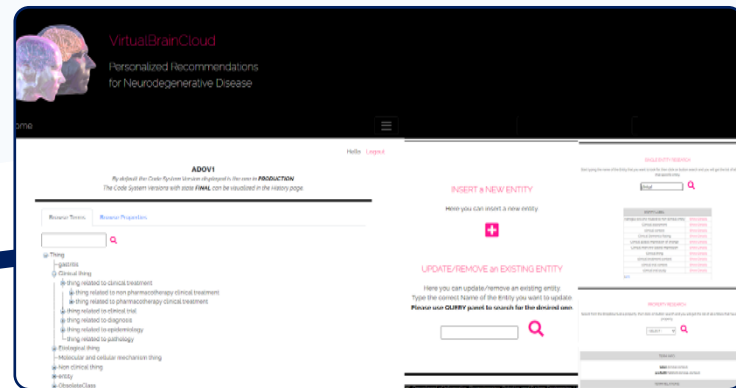




VirtualBrainCloud

Personalized Recommendations for Neurodegenerative Disease



www.VirtualBrainCloud-2020.eu

Public deliverable report

D3.15: CTS2 standard compliant interface will be released to project partners

Date April 2023
Authors Università di Genova
© VirtualBrainCloud consortium

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.	Introduction	3
1.1.	CTS2 Terminology Resources Sub-subtitle	3
1.2.	Functional Profiles	3
1.3.	Proposed Architecture	4
2.	Partners involved	4
3.	Description of work performed	5
3.1.	OWL to CTS2 transformation chain	5
3.2.	CTS2 list of necessary operations to start with	5
4.	Results	13
4.1.	Public Section:	15
4.2.	Private Section:	15
4.3.	Functional Profile Read & CTS2 Terminology Resource EntityDescription	17
4.3.1.	TreeView visualization:	17
4.3.2.	Entity details	18
4.3.3.	Technical implementation	19
4.4.	Functional Profile Query & CTS2 Terminology Resource EntityDescription	21
4.4.1.	Single Entity Research	21
4.4.2.	Property Research	22
4.5.	Functional Profile Maintenance & CTS2 Terminology Resource EntityDescription	22
4.5.1.	Insert a new entity	23
4.5.2.	Update an existing entity	24
4.5.3.	Delete an entity	27
4.6.	CTS2 Terminology Resource Map	27
4.6.1.	Single Mapping Read	28
4.6.2.	Technical implementation	28
4.7.	Functional Profile Maintenance & CTS2 Terminology Resource CodeSystemVersion	31
4.7.1.	Update an existing CodeSystemVersion	31
4.7.2.	Close an existing Code System Version	32
4.7.3.	Create a new CodeSystemVersion	32
5.	Conclusion, next steps	33



1. Introduction

The CTS2 is a standard defined by HL7 and 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 (using 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.

1.1. 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.

1.2. 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.

1.3. Proposed Architecture

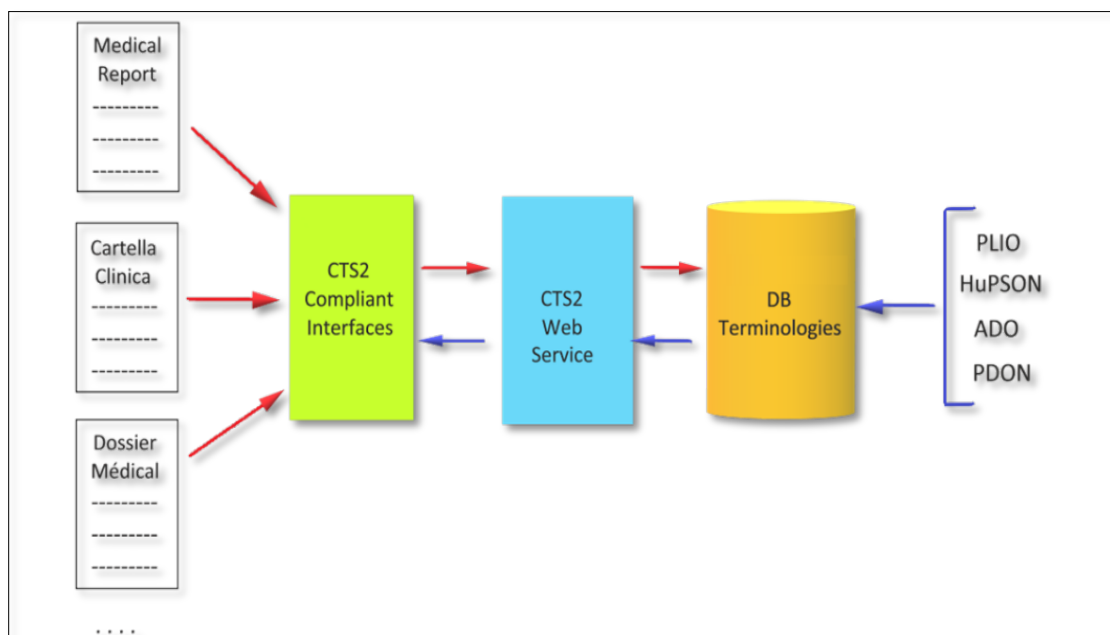


Figure 1: Proposed Architecture

According to the Consortium Agreement the interface is open source, and it can be used by everyone in the consortium. 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.

2. Partners involved

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 CTS2 approach from University of Genova (UNIGE).

The two deliverables share the same premise, the main aim is to explore the potential of two different theoretical approaches that devised two distinct standards. With these two deliverables two demos of the websites are made available at the following addresses:



- 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: Read, Query, Maintenance, and Map profiles based on HL7-CTS2. The 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 is the result of Milestone MS9.

3. Description of work performed

3.1. OWL to CTS2 transformation chain

We divided the process of transferring information from the ontologies to the CTS2 database into two subsystems.

First, we developed a console application that is able to open the ontology in rdf format, read all the classes and store them as records of a database table together with all their properties. Before starting the process, the user needs to specify the path where the ontology is and create an empty version of the database, which must contain one table with two attributes: “Id” and “Name”. Then the user needs to change the connection string to the database in order to store information in the newly created database.

Then we developed a second system that reads records stored into the table in the supporting database and then store them through the available standard functions into the CTS2 database.

3.2. CTS2 list of necessary operations to start with

In order to correctly insert the information of each specific ontology into the database, the following functions exposed by the CTS2 compliant web services by Healthropy srl need to be used in order.

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.

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 needs to be created one each for the ontologies whose entities map to one or more entities of the specific ontologies involved in the TVB Cloud project.

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 maintenance service can update 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 for.
- **hasOntologyLanguage:** The formal language used to express the code system or the ontology.
- **usedOntologyEngineeringTool:** The name or URI of an ontology engineering tool.

A single Code System may exist in different versions, released over the years or periodically. So, to each Code System will correspond one or more Code System Versions.

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.

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:

- defaultLanguage: The name or URI of the default language used in this version.
- documentURI: The URI of the specific document/format where this resource was retrieved.
- 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):



- **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.
- **sourceAndNotation:** A description of where the source of the version may be found, what format and language it is available in.
- **predecessor:** The name or URI of the resource version that immediately preceded this version on an evolutionary path.
- **officialResourceVersionId:** An official label or identifier that was assigned to this version by its publisher.
- **officialReleaseDate:** Information about the source, format, release date, version identifier, etc. of a specific version of an abstract resource.
- **officialActivationDate:** The date that this version of the resource is stated by its publishers to go into effect.

The single Code System Version will contain several entities, which could be classes or properties.

Create EntityDescription: 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.

Figure 2 shows an extract of the entities that is necessary to create in order to correctly manage the ADO ontology.

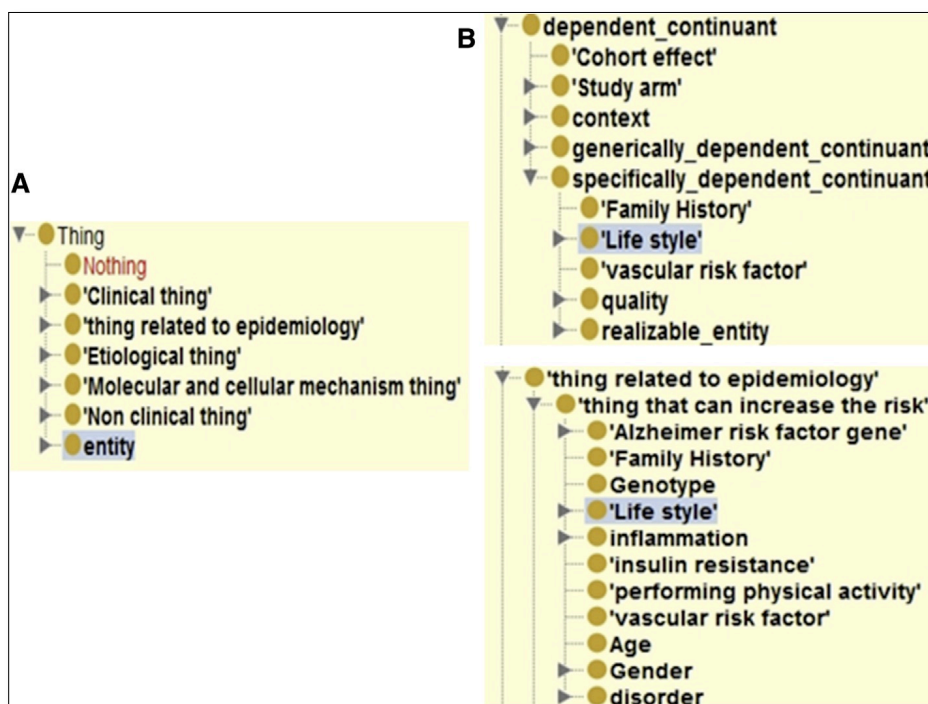


Figure 2: ADO Classes



Update EntityDescription: 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.

Specifically, the first attribute that should be addressed is *EntityType* as ontologies contain multiple types of entities. Examples from the ADO ontology are:

a. **Annotation Property:**



```

<!-- http://sc.ai.fraunhofer.de/NDDUO#Synonym -->
<AnnotationProperty rdf:about="http://sc.ai.fraunhofer.de/NDDUO#Synonym"/>

```

Figure 3: Example of Object Property from ADO.

b. Object Property:

```

<!-- http://sc.ai.fraunhofer.de/NDDUO#adjacent_to -->
<ObjectProperty rdf:about="http://sc.ai.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 4: Example of Object Property from ADO.

c. Class:

```

<!-- http://sc.ai.fraunhofer.de/AlzheimerOntology#Abeta_42 -->
<Class rdf:about="http://sc.ai.fraunhofer.de/AlzheimerOntology#Abeta_42">
  <rdfs:subClassOf rdf:resource="http://sc.ai.fraunhofer.de/AlzheimerOntology#amyloid_beta_protein"/>
  <rdfs:subClassOf>
    <Restriction>
      <onProperty rdf:resource="http://sc.ai.fraunhofer.de/NDDUO#has_role"/>
      <someValuesFrom rdf:resource="http://sc.ai.fraunhofer.de/AlzheimerOntology#Blood_Biomarker"/>
    </Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <Restriction>
      <onProperty rdf:resource="http://sc.ai.fraunhofer.de/NDDUO#has_role"/>
      <someValuesFrom rdf:resource="http://sc.ai.fraunhofer.de/AlzheimerOntology#Cerebrospinal_fluid_Biomarker"/>
    </Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <Restriction>
      <onProperty rdf:resource="http://sc.ai.fraunhofer.de/NDDUO#is_entity_used_in"/>
      <someValuesFrom rdf:resource="http://sc.ai.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 5: Example of Class from ADO.

According to the type of element of the ontology the following CTS2 elements have been used in the order listed below:

- Create Annotation Property Description*: EntityDescription.EntityType has count = 2, one is owl:AnnotationProperty and the other is rdf:Property.
- Create Object Property Description*: EntityDescription.EntityType has count = 2, one is owl:ObjectProperty and the other is rdf:Property.
- Create Named Entity Description*: EntityDescription.EntityType = owl:Named.

Another attribute that should be deeply analyzed is *Property*, it could belong both to a Class or an Object Property. It is composed by two elements:

- Predicate**: The name or URI of the property predicate. This element is of type Predicate Reference and it be derived from:
 - Literal Predicate*: Predicate Reference is a string that contains the name of the property.



- *Entity Predicate*: Predicate Reference is the URI of an existing entity which should belong to the group of Annotation Properties or Object Properties.
- b. **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 which exists of three types:

- *Literal Target*: when the statement type is LITERAL. It can be used for example with the Annotation Properties, e.g., label or Synonym.
- *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, e.g., when the property is an Object Property.
- *Resource Target*: when the statement type is RESOURCE.

An entity may have more than one value for the same predicate. In this case it is necessary to create a *List of StatementTarget* containing all items and then assign that list to Property.Value, while Property.Predicate remains unchanged.

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

Literal Target:

1. Single Literal Target:

```
<!-- http://scai.fraunhofer.de/NDDUO#preceded_by -->
<ObjectProperty rdf:about="http://scai.fraunhofer.de/NDDUO#preceded_by">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  <rdfs:domain rdf:resource="http://www.ifomis.org/bfo/1.1/span#ProcessualEntity"/>
  <rdfs:range rdf:resource="http://www.ifomis.org/bfo/1.1/span#ProcessualEntity"/>
  <dc:source>OBO_REL:preceded_by</dc:source>
  <rdfs:comment>An example is: translation preceded_by transcription; aging preceded_by development
  links classes of processes. Clearly, however, these two relations are not independent of each
  instance of C1 in a given lineage is preceded_by cellular processes involving an instance of C
  about what happened earlier, given what we know about what happened later. Thus it does not pr
  each is such as to be succeeded by some instance of P. Note that an assertion to the effect th
  in virtue of which the preceded_by relation obtains. Typically we will be interested in strong
  with a condition to the effect that the corresponding instances of P and P1 share participants
  a treatment of causality) that the one process in some way affects (for example, initiates or
  <rdfs:isDefinedBy>P preceded_by P&apos; if and only if: given any process p that instantiates P at
  is earlier than t.</rdfs:isDefinedBy>
</ObjectProperty>
```

Figure 6: Example of a Single Literal Target from ADO whose predicate is literal.



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'-sulfonilydianiline</NDDUO:Synonym>
  <NDDUO:Synonym>Dapsone</NDDUO:Synonym>
  <NDDUO:mapping>http://purl.obolibrary.org/obo/CHEBI_4325</NDDUO:mapping>
</owl:Class>

```

Figure 7: Example of more than 1 Literal Target from ADO whose predicate is an entity (Annotation Property).

Entity Reference Target:

1. Single Entity Reference Target:

```

<!-- http://scai.fraunhofer.de/AlzheimerOntology#ACE -->
<Class rdf:about="http://scai.fraunhofer.de/AlzheimerOntology#ACE">
  <rdfs:subClassOf rdf:resource="http://scai.fraunhofer.de/AlzheimerOntology#Alzheimer_risk_factor_gene"/>
  <rdfs:subClassOf>
    <Restriction>
      <onProperty rdf:resource="http://scai.fraunhofer.de/NDDUO#bearer_of"/>
      <someValuesFrom rdf:resource="http://scai.fraunhofer.de/AlzheimerOntology#Genetics"/>
    </Restriction>
  </rdfs:subClassOf>
  <metadata:prefixIRI rdf:datatype="http://www.w3.org/2001/XMLSchema#string">AlzheimerOntology:ACE</metadata:prefixIRI>
  <rdfs:label>ACE</rdfs:label>
</Class>

```

Figure 8: Example of a single Entity Reference Target whose predicate is an entity.

2. Literal Predicate with 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 9: Example of more than one Entity Reference Target whose predicate is literal.

The input parameters of the function Update Entity Description are:



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

Below some choices on the mapping between ontology element and CTS2 resources are reported.

- a. Attributes that cannot be modified and must be chosen during the **creation phase**:
 - Entity.EntityID.Name: it could be a string or a number corresponding to the index of the specific entity in the supporting database. 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, it depends on the type of function used to create the Entity Description and therefore it can't be changed once the entity is created.
- b. Attributes that can be changed through the Update Entity Description Request:
 - Entity.Definition = “isDefinedBy”
 - Entity.Example = “example”

(For each of them some metadata can be specified, e.g., the language).

 - Entity.Parent = “subClassOf”

Entity.Property = Any other property that could not fit into other attributes, e.g., “Synonym”, “label”, “disjointWith”, etc.**Create MapCatalogEntry:** The functional profile involved is the Map Catalog Maintenance Service. The input parameters are:

- **changeSet**
- **mapName:** The name by which the new catalog entry will be known within the context of the CTS2 service instance.
- **fromCodeSystem:** The name or URI of the code system from which the “from” entity references are derived (Type: NameOrURI).
- **fromValueSet:** The name or URI of the value set from which the “from” entity references are derived (Type: NameOrURI).
- **toCodeSystem:** The name or URI of the code system from which the “to” entity references are derived (Type: NameOrURI).
- **toValueSet:** The name or URI of the value set from which the “to” entity references are derived (Type: NameOrURI).

Return type is a MapCatalogEntry.

Create MapVersion: The functional profile involved is the Map Version Maintenance Service. The input parameters are:

- **changeSet**
- **createParameters:** The creation parameters (Type: CreateMapVersionRequest).

Return type is MapVersion.

The CreateMapVersionRequest function includes the following attributes:

- **mapVersionURI:** 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.



- **mapVersionName:** An identifier that uniquely names the map version within the context of the service.
- **fromCodeSystemVersion:** The name or URI of the “from” code system version.
- **fromValueSetDefinition:** The name or URI of the “from” value set definition.
- **toCodeSystemVersion:** The name or URI of the “to” code system version.
- **toValueSetDefinition:** The name or URI of the “to” value set definition.
- **useCodeSystemVersions:** The names or URIs of the code system versions to be used in the resolution of the value sets.

Create MapEntry: The functional profile involved is the Map Entry Maintenance Service. The input parameters are:

- **changeSet**
- **assertedBy:** The name or URI of the MapVersion that contains this assertion (Type: NameOrURI).
- **mapFrom:** The name or URI of the map entry to map from (Type: EntityNameOrURI).
- **processingRule:** An indicator that states whether all maps sets will be evaluated or processing will stop on the first hit.

Return type is MapEntry.

AddMapSet: The functional profile involved is the Map Entry Maintenance Service. The function adds the described MapSet to the input entry and return the result of the addition. The input parameters are:

- **ChangeSet**
- **mapVersion:** The name or URI of the MapVersion that contains this entry (Type: NameOrURI).
- **mapFrom:** The name or URI that identifies the specific entry (Type: EntityNameOrURI).
- **entryOrder:** The relative order of the entry within the existing list of set (Type: NaturalNumber).
- **processingRule:** An indicator that states how the target will be processed.

Return type is MapEntry.

AddMapTarget: The functional profile involved is the Map Entry Maintenance Service. The function adds the supplied map target in one-based target slot. The input parameters are:

- **changeSet**
- **entry:** The existing MapEntry to add to the target to (Type: MapEntry).
- **addToSet:** The one-based identifier of the particular MapSet to add to the target to (Type: NaturalNumber).
- **targetSlot:** The one-based slot number to add the new target to (Type: NaturalNumber).
- **target:** The parameters of the target to add (Type: MapTargetRequest).

Return type is MapEntry.

Among others, the MapTargetRequest function includes the following attributes:

- **mapRule:** The map rule for the target. If omitted, the rule always evaluates to TRUE.
- **mapTo:** the specific entity to include in the map on the ‘TO’ side (Type: URIAndEntityName).
- **externalEntryId:** an identifier assigned by an external body

4. Results



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



4.1. Public Section:

The homepage summarizes some useful information about the CTS2 standard.

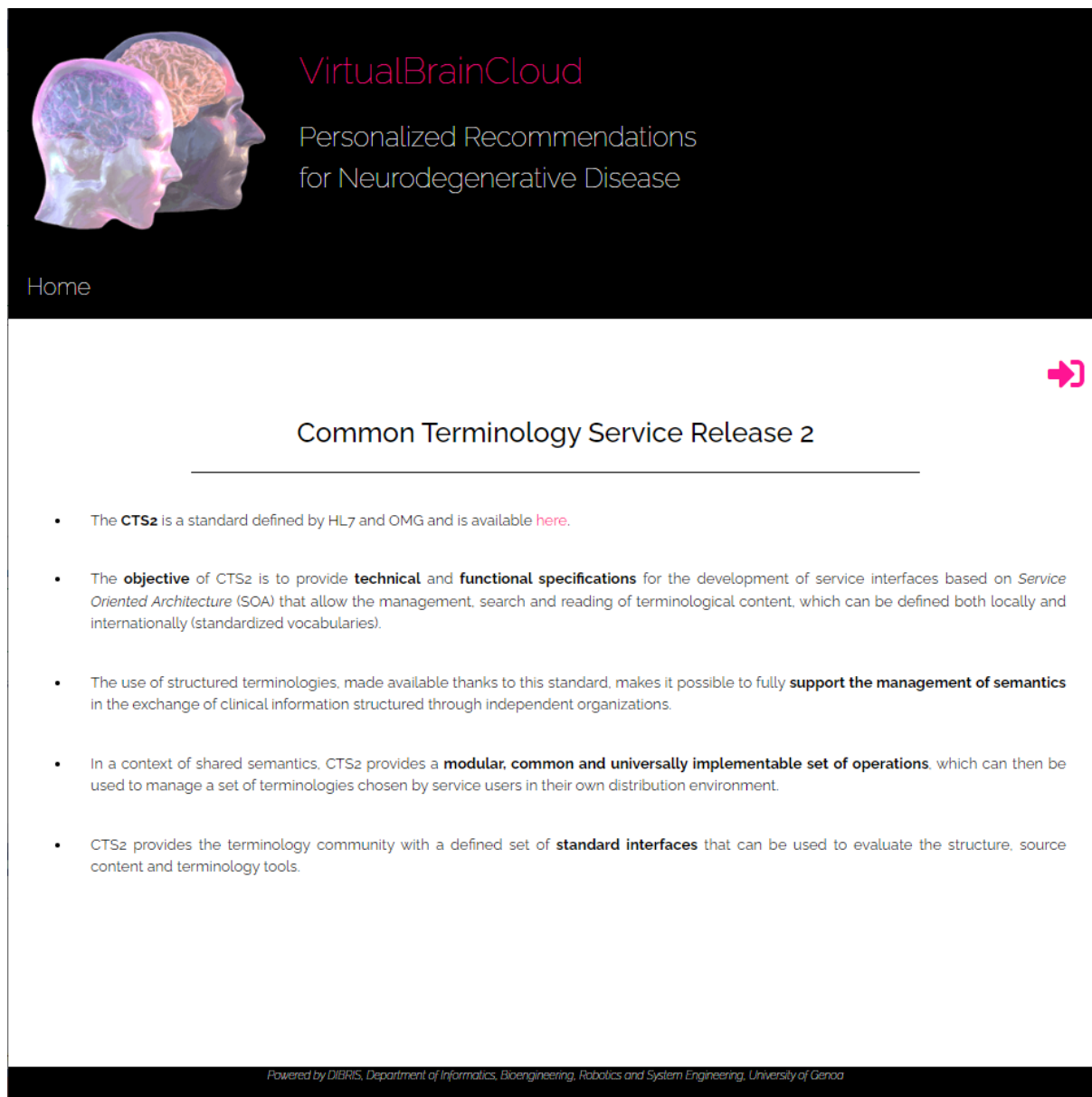


Figure 10: Home Page of the CTS2 compliant interface

4.2. Private Section:

Only authorized people that participate in the TVB Cloud project can access the pages related to ontologies. The access to this part of the platform is regulated through a **login page**:

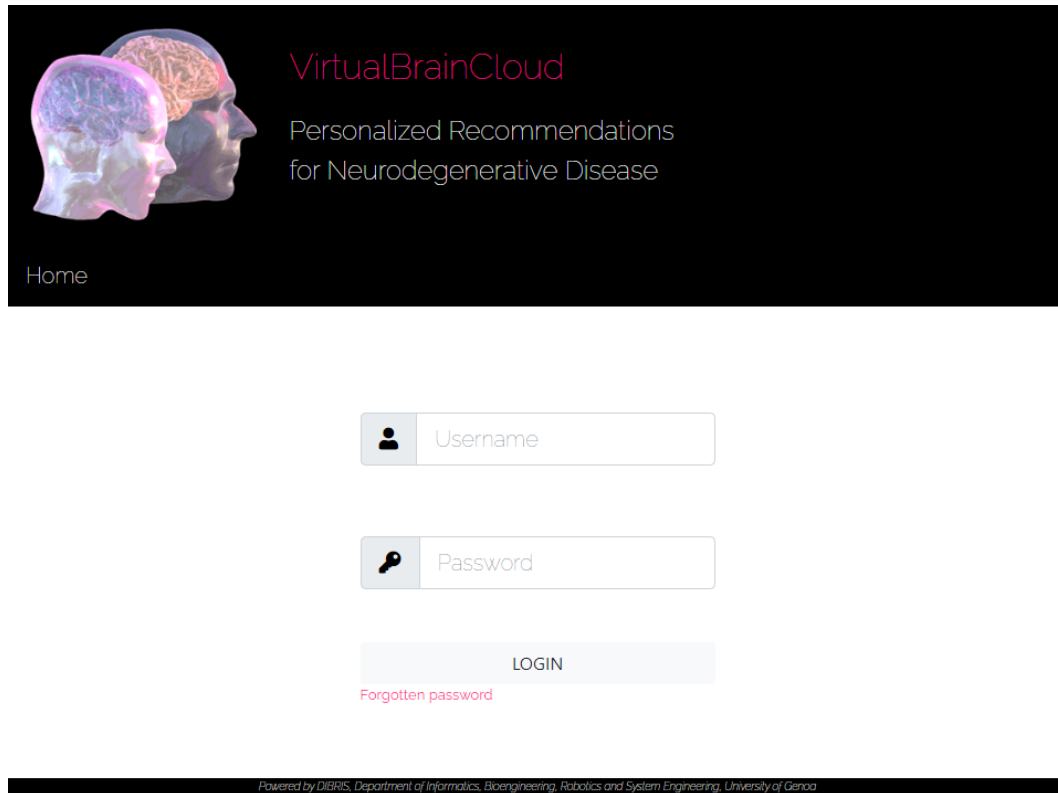


Figure 11: Login page of the CTS2 compliant interface

A page to restore password is also available. The user can click on “**Forgotten password**” and he/she will be redirected the following page:

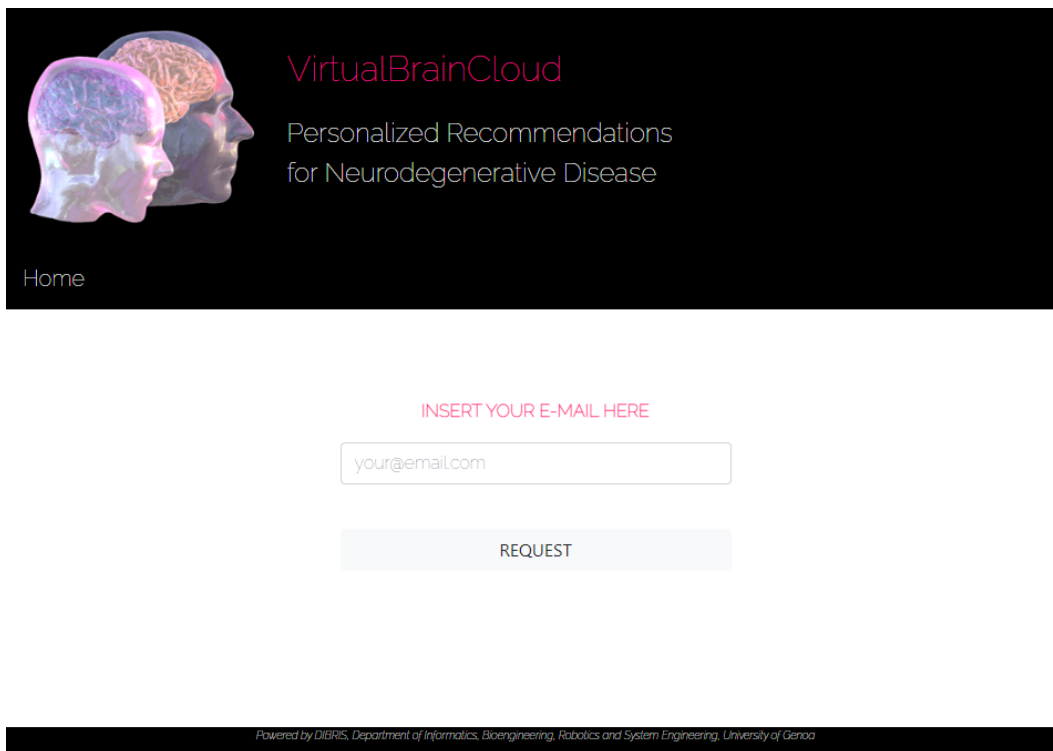


Figure 12: Page where the user can ask for the reset of credentials. The email address that he should use is the one he/she used for registration.



4.3. Functional Profile Read & CTS2 Terminology Resource EntityDescription

4.3.1. TreeView visualization:

In the first part of the page the user can browse the entities (classes and properties) of a specific CodeSystemVersion.

The screenshot displays the VirtualBrainCloud interface. At the top, there is a header with the logo and the text "VirtualBrainCloud Personalized Recommendations for Neurodegenerative Disease". Below the header, there is a navigation bar with "Home" and a menu icon. The main content area shows the "ADOV1" section, with a note: "By default the Code System Version displayed is the one in **PRODUCTION**. The Code System Versions with state **FINAL** can be visualized in the History page." Below this, there are two tabs: "Browse Terms" (selected) and "Browse Properties". A search bar is present. The tree visualization shows a hierarchy of classes under "Thing":

- Thing
 - gastritis
 - Clinical thing
 - thing related to clinical treatment
 - thing related to non pharmacotherapy clinical treatment
 - thing related to pharmacotherapy clinical treatment
 - thing related to clinical trial
 - thing related to diagnosis
 - thing related to epidemiology
 - thing related to pathology
 - Etiological thing
 - Molecular and cellular mechanism thing
 - Non clinical thing
 - entity
 - ObsoleteClass

Figure 13: Example of the tree visualization of ADO Classes



The screenshot shows the VirtualBrainCloud homepage with the title "Personalized Recommendations for Neurodegenerative Disease". Below the header, there is a section for "ADOV1" with a note about code system versions. The main content area features two tabs: "Browse Terms" and "Browse Properties". The "Browse Properties" tab is active, displaying a tree view of properties. The tree structure is as follows:

- adjacent_to
- bearer_of
- comment
- contributor
- creator
- date
- derives_from
- format
- has_agent
 - ...has_participant
- has_role
- identifier
- inheres_in
- is_entity_used_in
- label
- language
- located_in
- mapping
- part_of
- preceded_by

Figure 14: Example of the tree visualization of ADO Properties

4.3.2. Entity details

If the user clicks on one element of the TreeView both entity information and relations will appear in a grid.

The screenshot displays the "Entity Details Visualization" in a grid format. It is divided into two main sections: "TERM INFO:" and "TERM RELATIONS:". The "TERM INFO:" section contains the following details:

- Synonym** non pharmacotherapy treatment;
non pharmacotherapy treatments;
- label** thing related to non pharmacotherapy clinical treatment;
- prefixIRI** NDDUO:Non_pharmacotherapy;

The "TERM RELATIONS:" section contains the following details:

- is_entity_used_in** clinical treatment context;
- subClassOf** thing related to clinical treatment;

Figure 15: Entity Details Visualization



4.3.3. Technical implementation

It is necessary to remember that if the state of the CodeSystemVersion is FINAL then it is closed it cannot 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: CodeSystemVersion with state “open” (it will be only one)
- Current: last CodeSystemVersion with state “final”
- Previous: 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 previous versions, he/she will be able to do that in the history page.

Get CodeSystemVersion for CodeSystem: 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.

Considering CodeSystemVersion with tag “Production”, 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 CodeSystemVersion: 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 from 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 CodeSystem. 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. 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:** the URI of the specific CodeSystemVersion, in example the ADOV1.
- **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”.

To find the entity whose name is the same of a string specified by the user the “equals” is required.

- **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). 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.



4.4. Functional Profile Query & CTS2 Terminology Resource EntityDescription

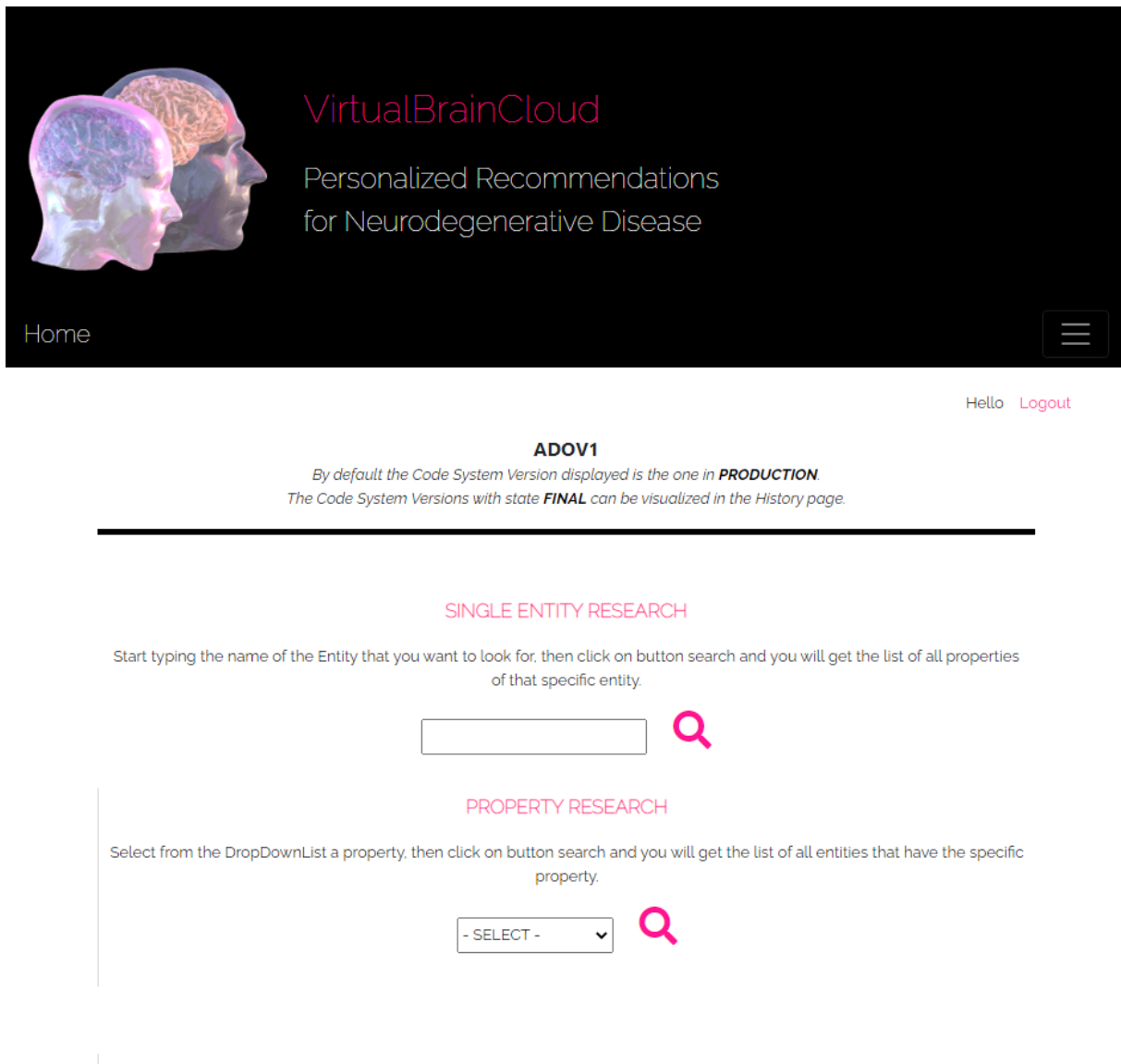


Figure 16: Functional Profile Query Interface

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.

4.4.1. 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.



SINGLE ENTITY RESEARCH

Start typing the name of the Entity that you want to look for; then click on button search and you will get the list of all properties of that specific entity.

clinical

🔍

PROPERTY RESEARCH

Select from the DropDownList a property; then click on button search and you will get the list of all entities that have the specific property.

- SELECT -

🔍

ENTITY LABEL	
Astroglia cell line related to non clinical entity	Show Details
Clinical assesment	Show Details
clinical context	Show Details
Clinical Dementia Rating	Show Details
Clinical global impression of change	Show Details
Clinical interview based impression	Show Details
Clinical thing	Show Details
clinical treatment context	Show Details
clinical trial context	Show Details
clinical trial study	Show Details

TERM INFO:

Label clinical context;
prefixIRI NDDUO:clinical_context.

TERM RELATIONS:

subClassOf context.

Figure 17: Example of Single Entity Research

If the user wants to see all the details of that specific entity, he/she can click on “*Show Details*” on the corresponding row of the grid and details will be visualized in right part of the page.

4.4.2. Property Research

This second operation gives the user the possibility to find all the entities that have a specific Annotation or Object property, Figure 21 shows the example searching for all entities that have a synonym.

SINGLE ENTITY RESEARCH

Start typing the name of the Entity that you want to look for; then click on button search and you will get the list of all properties of that specific entity.

🔍

PROPERTY RESEARCH

Select from the DropDownList a property; then click on button search and you will get the list of all entities that have the specific property.

Synonym

🔍

ENTITY LABEL	
_Presence of alzheimer disease associated Protein	Show Details
_Presence of Docosahexenoic acid	Show Details
Abeta 1-40 test	Show Details
Abeta 1-42 test	Show Details
Abeta 42	Show Details
Abeta-RAGE interaction	Show Details
ACE inhibitor	Show Details
Acetylation	Show Details
acetylcholinesterase inhibitor	Show Details
Action potential	Show Details

TERM INFO:

Synonym ACE inhibitors;
angiotensin-converting enzyme inhibitor;
mapping http://purl.obolibrary.org/obo/CHEBI_35457;
Label ACE inhibitor;
prefixIRI AlzheimerOntology:ACE_inhibitor.

TERM RELATIONS:

subClassOf inhibitor.

Figure 18: Example of Property Research

Even in this case it is possible to click on “*Show Details*” and the grid will show term information and relations.

4.5. Functional Profile Maintenance & CTS2 Terminology Resource EntityDescription

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

- a. Insert a new entity.
- b. Update an existing entity.
- c. Delete an existing entity.

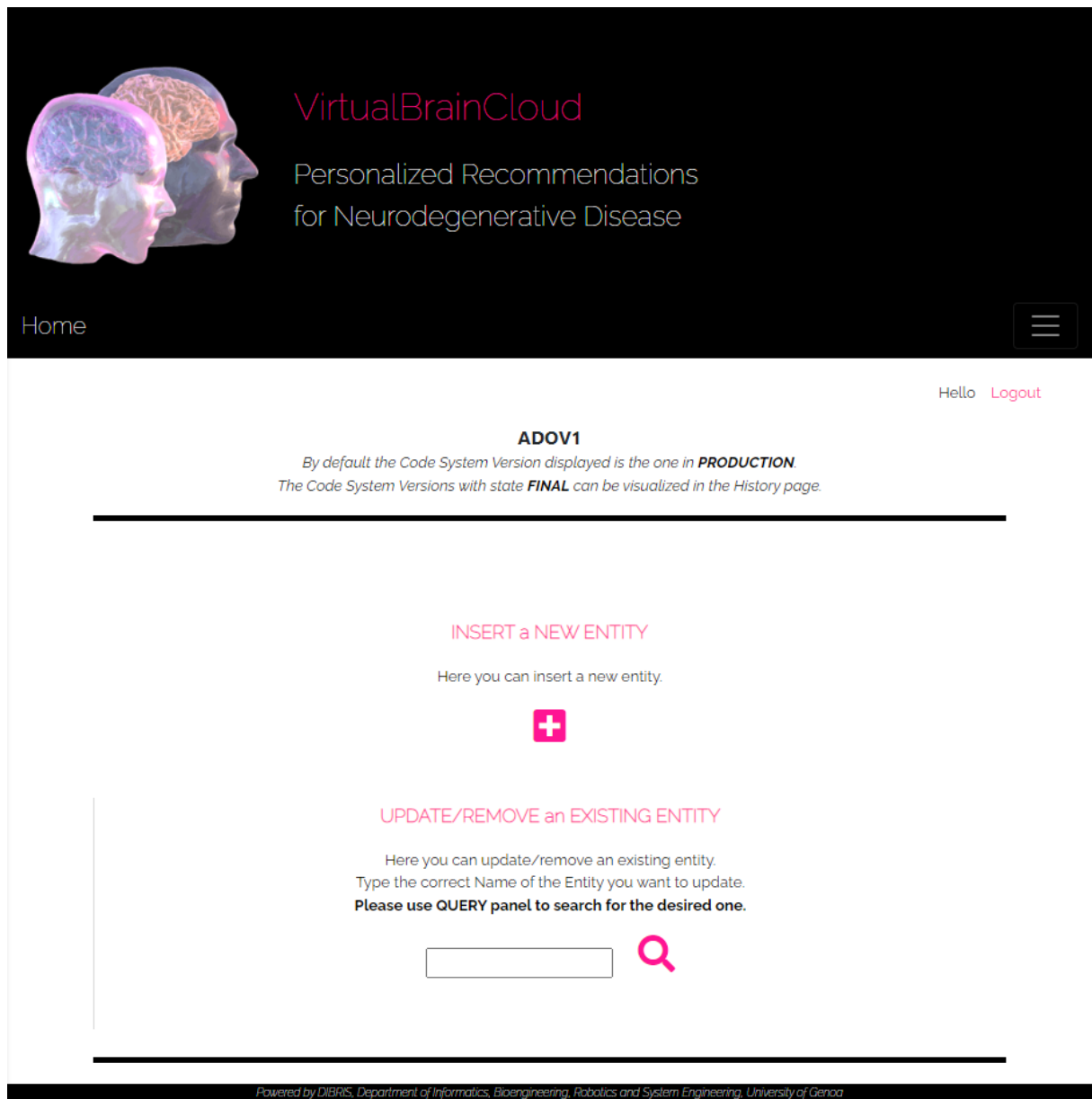


Figure 19: Functional profile Maintenance Interface

4.5.1. Insert a new entity

The complete set of technical operations that should be executed to insert a new entity have been described in detail previously (Section 7, **Create Entity Description**). To insert a new entity, it is necessary to determine the CodeSystem and CodeSystemVersion that it belongs to, its type, and its name, which could be saved as the Annotation Property “label” or just stored as display name to be used to visualize the entity. In the case the entity is a Class, it will not immediately appear in the TreeView, because a parent needs to be specified. User should press on the icon plus to open the specific panel. It is necessary to insert ALL the information that are required (*), 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 accessed and the second one is the version in PRODUCTION. The type (AnnotationProperty, ObjectProperty or Class) is necessary because the entity should be created with the corresponding function. Once ALL the mandatory information (*) is set the user can press button SAVE.



INSERT a NEW ENTITY

Here you can insert a new entity.

NEW ENTITY

CODE SYSTEM':

CODE SYSTEM VERSION':

ENTITY TYPE':

ENTITY NAME':

Label

Figure 20: Insert new entity

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

4.5.2. Update an existing entity


The complete set of operations that should be executed to update an existing entity have been described in detail in previously (Section 7, **Update EntityDescription**).



In order to reach the update section, the user has to choose the entity he wants to update typing the exact name of the entity in the textbox, if the user is not sure of the exact name, he can search using Profile Read or Query. Once the entity details grid is ready two icons appear: **EDIT** and **REMOVE**.



UPDATE/REMOVE an EXISTING ENTITY

Here you can update/remove an existing entity.
Type the correct Name of the Entity you want to update.
Please use QUERY panel to search for the desired one.

ObsoleteClass 

TERM INFO:



prefixIRI AlzheimerOntology:ObsoleteClass;

TERM RELATIONS:

displayName ObsoleteClass;
subClassOf Thing;

Figure 1: Update / Remove entity

If the user clicks on the edit icon two other buttons will appear under the grid, one to edit term information and the other one to edit term relations.

TERM INFO:

prefixIRI AlzheimerOntology:ObsoleteClass;

TERM RELATIONS:

displayName ObsoleteClass;
subClassOf Thing;

Figure 22: Buttons to update term information and relations

If the user clicks on button “Update Term Info” then the following attributes of the entity will be load with existing values and they will be ready to be edit:

- Definition
- Example
- Note
- Parent: the exact name of the parent should be written
- Annotation properties (n = 4): All the ontology Annotation properties will be loaded in a dropdownlist.



TERM INFORMATION:

Each of these element can be filled with more than one value, please use ; to divide them, otherwise they will be considered only one.

Definition:

Entity Type:

Example:

Note:

Parent:

ANNOTATION PROPERTIES:

Figure 23: Update Term Information panel

The user can insert more than one value for each attribute, but in order to correctly store them they should be separated by a semicolon symbol “;”. To insert a new Annotation Property the “+” button should be pressed, the maximum number of properties that could be added is 4, but this limit could be extended in the future, if necessary. The EntityType cannot be changed as a consequence of the process of entity creation.

If the user clicks on button “Update Term Relations” then the following attributes of the entity will be load with existing values and they will be ready to be edit:

- Object Properties (n = 4): All the ontology Object properties will be loaded in a dropdownlist.
- Other Properties: properties that do not belong to Annotation nor to Object properties.



OBJECT PROPERTIES:

bearer_of

OTHER ONTOLOGY PROPERTIES:

Add other ontology property.

Figure 24: Update Term Relations panel

4.5.3. Delete an entity

The user may want to remove an entity from the list of entities that can be used of a specific CodeSystemVersion because it has been added wrongly. From the technical implementation point of view, the function 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.

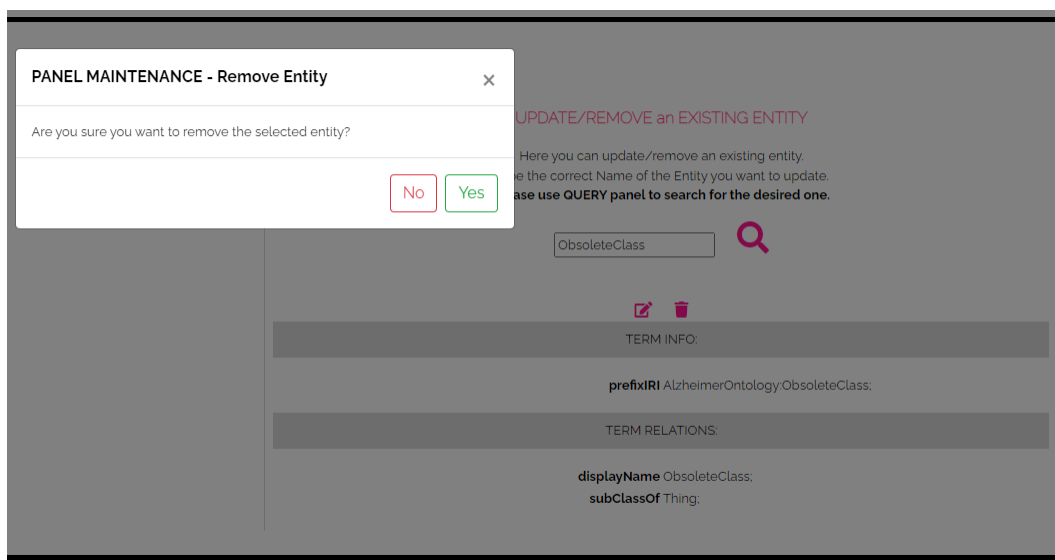
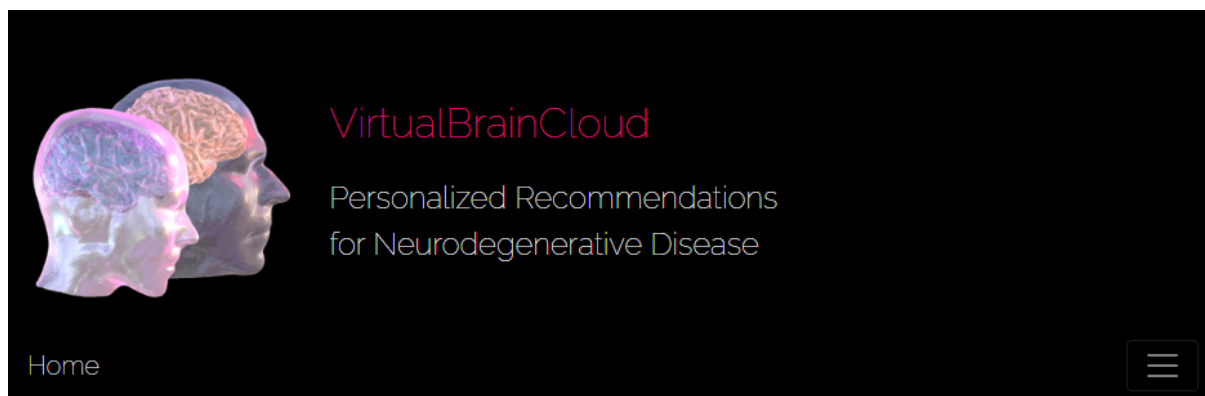


Figure 25: Modal delete entity

4.6. CTS2 Terminology Resource Map

This section contains the functions that allow the user to visualize all mappings between the selected ontology and the other ontologies with at list one entity mapped.



Hello [Logout](#)

ADOV1

*By default the Code System Version displayed is the one in **PRODUCTION**
The Code System Versions with state **FINAL** can be visualized in the History page.*

LIST OF MAPPINGS

Here you can visualize the list of all ontologies that have **at least one entity** that maps to the considered ontology.

ONTOLOGIES	
A formal representation of the knowledge within th	Show Details
AdaLab ontology	Show Details

FROM ONTOLOGY:

TO ONTOLOGY:

Powered by DIBRIS, Department of Informatics, Biengineering, Robotics and System Engineering, University of Genoa

Figure 26: Visualization of list of mappings

4.6.1. Single Mapping Read

The user can visualize the complete list of all MapEntries belonging to the single Map by clicking on “Show Details” on the line of the desired ontology.

4.6.2. Technical implementation

Get all Maps: Function that returns the DirectoryURI. It contains the reference to all the map catalog entries available.

Restrict by CodeSystem: Function which restricts the list of maps to those belonging to the CodeSystem identified as ‘FROM’ (i.e., the CodeSystem which the user wants to know the maps of). The attributes are:

- **directory:** The DirectoryURI containing the references to the map catalog entries (Type: MapCatalogEntryDirectoryURI). It is the output of the previous function.
- **codeSystems:** The identifiers used locally or the URIs of the CodeSystems the user wants to know the maps of (Type: NameOrURIList).
- **mapRole:** It could be ‘FROM’ or ‘TO’, so it indicates the side of the map (Type: MapRole).



Returns a MapCatalogEntryDirectoryURI.

Resolve as List: Returns a list of MapCatalogEntries corresponding to the 'TO' CodeSystems that map to the specified 'FROM' CodeSystem.

For each of these MapCatalogsEntries, it is necessary to extract the specific matching at two nested levels:

- a. **CodeSystemVersions:** First, it is necessary to identify the each couple of specific CodeSystemVersions involved in the map.

Exists MapVersion For Map: Function that determines if a MapVersion exists between two MapCatalogs. The input attributes are:

- **map:** The identifier of a map, which can be a name or a URI (Type: NameOrURI). Within this project each map between CodeSystems is identified by the name of the two CodeSystems joint with an “_”.
- **tag (optional):** An element of type VersionTagReference which denotes the version tag of the CodeSystemVersion whose map is requested. It can be: production, current, obsolete or previous.
- **context (optional):** Set of parameters that allow to filter the search, e.g., language or date (Type: ReadContext).

Returns a Boolean. If True is returned:

Read MapVersion For Map: Function which retrieves the information about a map given in input the same attributes of the previous function **Exists MapVersion For Map**.

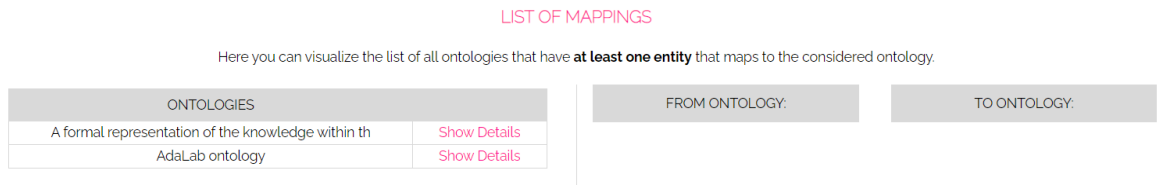


Figure 27: Example of Ontologies that map to ADO.

To reach the second level of mappings , the user can click on the button “*Show Details*”:

- b. **Entities:** It is necessary to identify which entities belonging to the 'FROM' CodeSystem map to the entities belonging to the 'TO' CodeSystem for the specific CodeSystemVersion. Each mapVersion contains a list of mapEntries. Each of them, as mentioned above, maps TO ZERO or more target Entities that belong to the TO CodeSystemVersion. This information is contained into the attribute MapEntry.mapTarget (Type: MapTargetList). Finally, each mapTarget correspond to one entity of the 'TO' CodeSystemVersion. So, it is possible to read all the information about the entity using the sequence of functions mentioned in Section **8.4.1 Single Entity Research**. Mappings will appear in the right side of the screen in a grid.



ADOV1

By default the Code System Version displayed is the one in **PRODUCTION**
The Code System Versions with state **FINAL** can be visualized in the History page.

LIST OF MAPPINGS

Here you can visualize the list of all ontologies that have **at least one entity** that maps to the considered ontology.

ONTOLOGIES		ADOV1	ISO-FOOD1
A formal representation of the knowledge within th	Show Details	List of Entities	List of Entities Details
AdaLab ontology	Show Details	phytic acid	phytic acid
		arachidonic acid	arachidonic acid
		ratio	ratio UO:0010006
		ratio	ratio UO:0000190
		Person	Person
		copper	copper
		aluminium	aluminium
		catechin	catechin
		lithium	lithium
		zinc	zinc
		protein	Protein
		serotonin	serotonin
		biomarker	Biomarker
		inflammation	Inflammation

Figure 28: List of all mapEntries belonging to the mapCatalog named identified by the two CodeSystems ADO and ISO-FOOD1.




4.7. Functional Profile Maintenance & CTS2 Terminology Resource CodeSystemVersion

UPDATE CodeSystemVersion

Here you can update the features of the Code System Version in PRODUCTION.

CODE SYSTEM:	<input type="text" value="ADO"/>
CODE SYSTEM VERSION ID:	<input type="text" value="ADOV1"/>
LANGUAGE:	<input type="text" value="EN"/>
SOURCE and NOTATION:	<input type="text" value="Source: https://bioportal.bioontology.org/ontologies/ADO"/>
RELEASE DATE:	<input type="text" value="23/07/2013"/>
ACTIVATION DATE:	<input type="text" value="23/07/2013"/>
RESOURCE VERSION ID:	<input type="text" value="111"/>



CLOSE CodeSystemVersion

If you are sure that the CURRENT Code System Version is completed you can close it.
Once it is closed it can't be edit anymore.




Figure 28: Section of the user interface which allows to Update the CodeSystemVersion.

4.7.1. Update an existing CodeSystemVersion

The complete set of operation that should be executed have been described in detail in the previous paragraph. By clicking the icon in the corresponding section, the user will visualize the attributes that are available for editing and loaded with the existing values:

- Language
- Source and Notation
- Release Date
- Activation Date

The button 'Back' allows to go back without updating the resource, while the button 'Save' will update the existing information stored into the database for that specific resource.

N.B: The complete set of information is visible only when current data is after the Release and Activation dates.



UPDATE CodeSystemVersion

Here you can update the features of the Code System Version in PRODUCTION.

CODE SYSTEM:	<input type="text" value="ADO"/>
CODE SYSTEM VERSION ID:	<input type="text" value="ADOv1"/>
LANGUAGE:	<input type="text" value="EN"/>
SOURCE and NOTATION:	<input type="text" value="Source: https://bioportal.bioontology.org/ontologies/ADO"/>
RELEASE DATE:	<input type="text" value="23/07/2013"/>
ACTIVATION DATE:	<input type="text" value="23/07/2013"/>
RESOURCE VERSION ID:	<input type="text" value="111"/>

Figure 29: Panel Update CodeSystemVersion

4.7.2. 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. The user can close the CodeSystemVersion in production using the button with a closed locker.

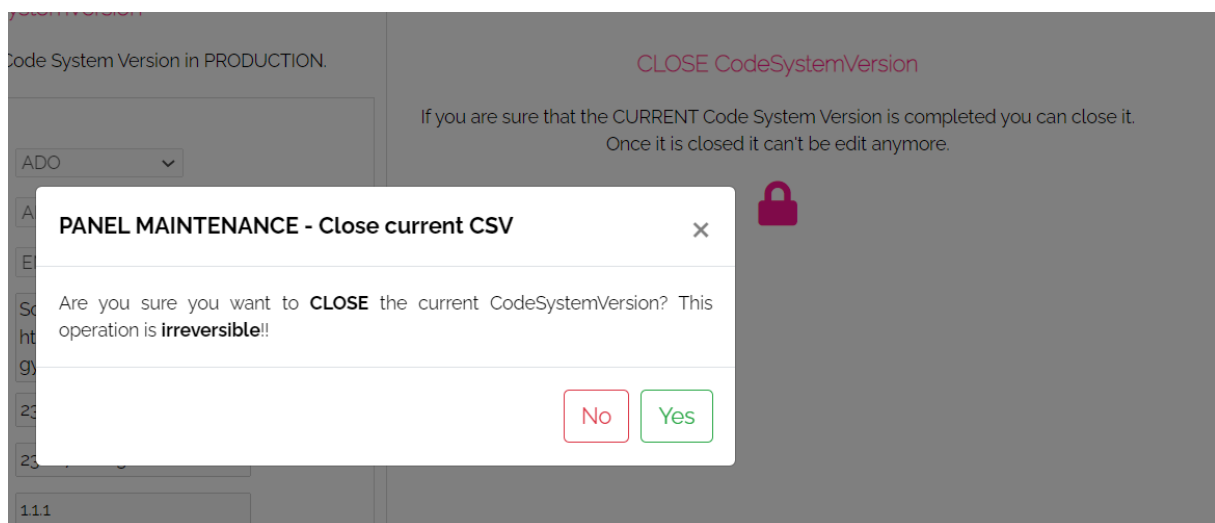


Figure 30: Modal close CodeSystemVersion

4.7.3. Create a new CodeSystemVersion

The complete set of operations used to reach this goal have been described in detail in the previously (Section 7, **Create CodeSystemVersion**). To create a new CodeSystemVersion, it is necessary to set to FINAL the state of the one in PRODUCTION.

So, only when there is no CodeSystemVersion open, this operation becomes available in the functional profile MAINTENANCE.

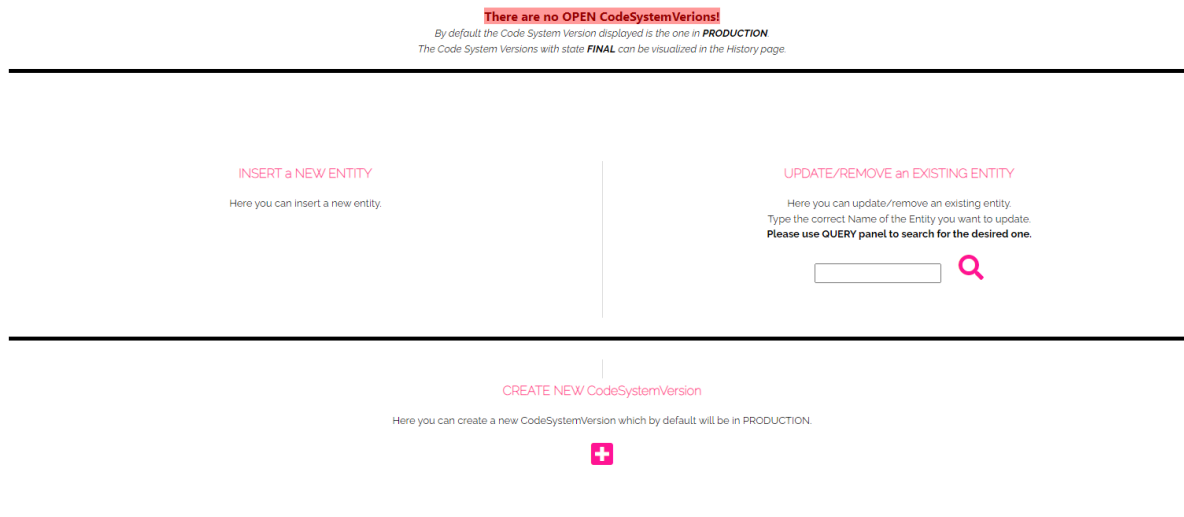


Figure 31: Functional profile Maintenance with no CodeSystemVersion in PRODUCTION

By clicking on the plus icon, a panel appears. Here the user can insert the name of the new CodeSystemVersion.

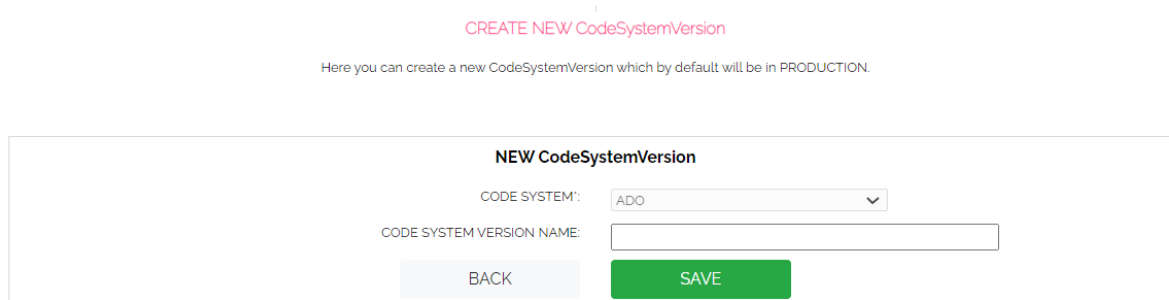


Figure 32: Panel for creating a new CodeSystemVersion

All the other information could be updated within the panel Update CodeSystemVersion.

5. Conclusion, next steps

This document clearly presents the application of the CTS2, which is a complete and solid standard for terminology management in clinical field.

However, in order to guarantee the correct exploit of the standard, these two characteristics require to take into account a great number of complex items. For this reason, within this project, we devised a user interface which allows a friendly usage of the standard potentiality without the need to deeply know its technical aspects. The availability of an almost complete implementation of the standard, realized by a spin-off of the University of Genoa before the beginning of this project, allowed us to concentrate uniquely on the development of the user interface. The interface is web based and exchange information through SOA based interface defined by the standard. These interfaces are owned by the university spin-off, Healthropy s.r.l., and are maintained independently from the project. The access will be ensured to all the authorized partners also after the project conclusion for a reasonable amount of time.



The state of the art of terminological services evolved since the proposal of this project. Specifically, REST interfaces able to support the terminological service have been defined and can easily interact with FHIR resources.

A possible future development could be to integrate the developed architecture in hospital environments that natively use FHIR through the involvement of these new interfaces.