NISTIR 7561

OntoSTEP: OWL-DL Ontology for STEP

Sylvere Krima
Raphael Barbau
Xenia Fiorentini
Rachuri Sudarsan
Ram D. Sriram

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7561

OntoSTEP: OWL-DL Ontology for STEP

Sylvere Krima
Raphael Barbau
Xenia Fiorentini
Rachuri Sudarsan
Ram D. Sriram
Manufacturing Systems Integration Division
Manufacturing Engineering Laboratory

May 2009

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Deputy Director

Abstract

The Standard for the Exchange of Product model data (STEP) [1] contains product information mainly related to geometry. The modeling language used to develop this standard, EXPRESS, does not have logical formalism that will enable rigorous semantics. In this paper we present an OWL-DL (Web Ontology Language - Description Logic) [2] version of STEP (OntoSTEP) that will allow logic reasoning and inference mechanisms and thus enhancing semantic interoperability. The development of OntoSTEP requires the conversion of EXPRESS schema to OWL-DL, and the classification of EXPRESS instances to OWL individuals. Currently we have considered AP203 [3] - the most widely used Application Protocol (AP) for the exchange of Computer-Aided Design (CAD) files - and STEP Part 21 [4] CAD files - CAD files conformant to the data exchange format defined in Part 21 - for schema level conversion and instance level classification respectively. We will describe a web application to demonstrate OntoSTEP. We are currently extending OntoSTEP to include information such as function, behavior, and assembly requirements.

Table of Contents

Table of Figures

Index of Tables

1 Introduction

Manufacturing organizations spend a considerable amount of resources to understand and apply the Product Lifecycle Management (PLM) approach. The PLM approach enables organizations to manage, in an integrated fashion, the product portfolio from conception to disposal [5]. Representation and management of product information is the key for a successful implementation of PLM.

To enable the exchange of product data through a product lifecycle, the International Organization for Standardization (ISO) has developed the Standard for Exchange of Product model data (STEP) [1] (ISO 10303), which is still evolving to meet the needs of modern Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) systems. The implementable data specification of STEP is represented by the Application Protocols (APs). Examples of the most widely used APs are the AP203 [3] and AP214 [6] for the exchange of CAD files, and AP239 [7] for the Product Life Cycle Support (PLCS). These APs mainly focus on product management data and geometry information. Unfortunately the representation of function and behavior is outside of their scope. We call concepts such as function and behavior as "beyond geometry information" since they are most often related to the geometry of the product.

The STEP APs are defined using the EXPRESS language. EXPRESS (ISO 10303-11) [8] is a data modeling language designed by ISO to model STEP entities. EXPRESS was developed to enhance product modeling and provides support to describe "the information required for designing, building, and maintaining products." Data models (or schemas) are represented in EXPRESS as a network of concepts. Concepts are called entities and relationships between concepts are called attributes. Entities and attributes are therefore the basic constructs of EXPRESS. STEP Part 21 [4] defines the syntax for representing data according to a given EXPRESS schema.

Many tools have been developed to check the syntax of the EXPRESS information models and the validity of the instances against the information models. Unfortunately, these tools are specifically developed for STEP implementers and consumers so their usage is restricted to the field of product modeling in EXPRESS. Moreover, since EXPRESS is not based on formal semantics, it is difficult to check the quality of these tools.

Our goal is to overcome these issues by translating STEP in OWL-DL [2] (Web Ontology Language - Description Logic), which allows the application of mechanisms to check models and data validity, to check the consistency of the instances, and to infer new knowledge. These mechanisms are performed by software tools called reasoners. Measuring model quality is easier since they are based on a formal semantics. We refer to the translated STEP as OntoSTEP in the remainder of the paper. OntoSTEP could be used to express and semantically enrich product information available in STEP files. In our use-case we translate in OWL the STEP AP 203 data model and Part 21 CAD files. The methodology followed for the use-case is fully applicable to any other STEP AP and any Part 21 file.

Semantic interoperability between two CAD systems necessitates a framework shown in Figure 1. OntoSTEP is the first major step towards developing such a framework. A semantic model supports the representation not only of product geometry concepts but also of "beyond geometry" concepts. The top of Figure 1 shows our previous work, while the bottom represents our current focus and how OntoSTEP fits in.

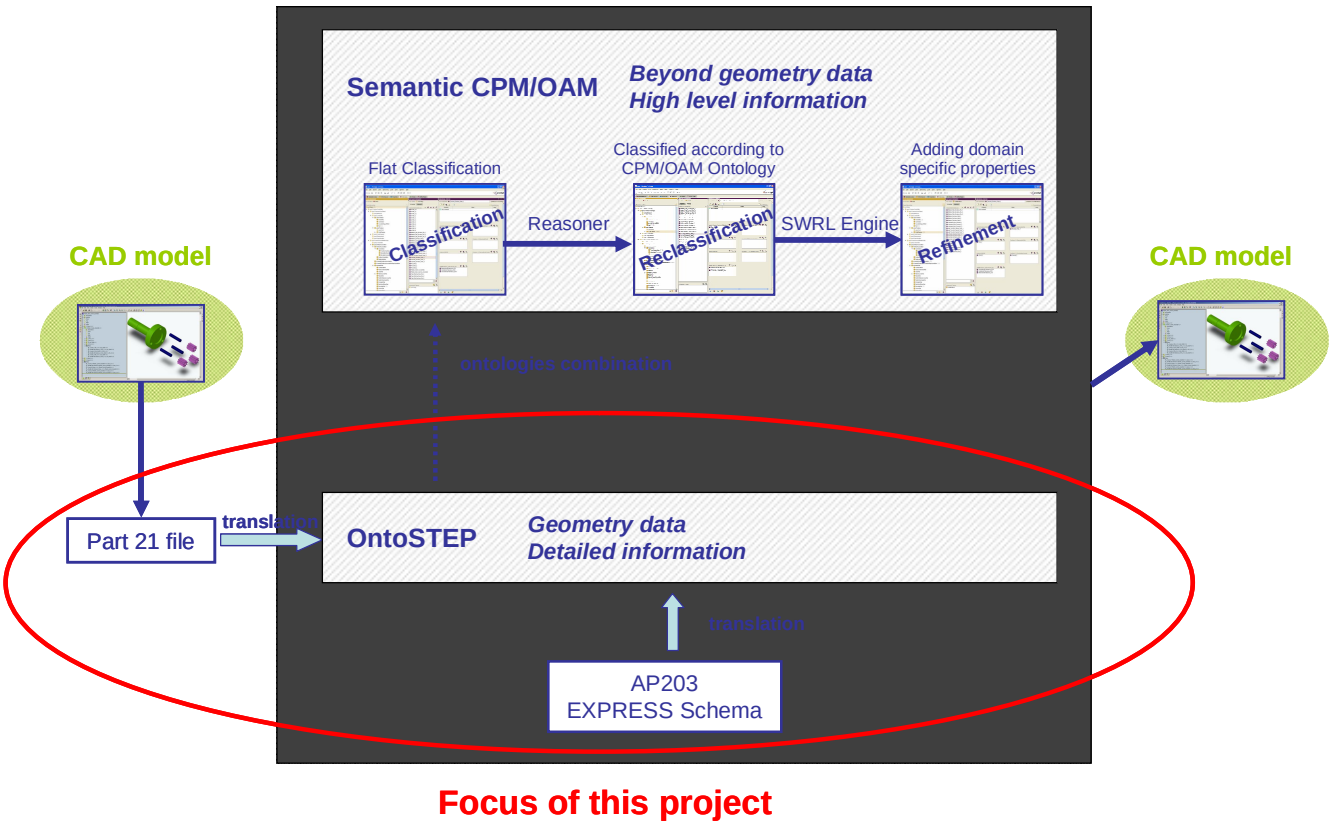

Figure 1. OntoSTEP project overview

In our previous work [9], we created a semantic model including beyond geometry concepts from the NIST (National Institute of Standards and Technology) Core Product Model (CPM) [10] and Open Assembly Model (OAM) [11] (semantic CPM/OAM in Figure 1.) This model was developed in OWL-DL 1.0 [2] and enriched with Semantic Web Rule Language (SWRL) rules [12]. A third party reasoner allows reclassifying the input instances and the SWRL rules allow refinement and improve the model.

In the future, we plan to combine OntoSTEP with the semantic model of CPM and OAM: the instantiation of such a combined model could be, in part, automatically performed from a Part 21 CAD output file. A plug-in for CAD application will enable the beyond geometry information of the designed product to be included in the CAD models. Beyond geometry information and geometry information would then be represented in a unique consistent model.

The paper is organized as follows. We review some works related to OntoSTEP in Section 2. Then we introduce the OntoSTEP mapping rules in Section 3. We present the details of our implementation and the tools used to realize it in Section 4. We discuss a use case for OntoSTEP in Section 5 and, finally, we present our conclusions and future plans in Section 6.

2 Related work

This section discusses two related efforts that aim to develop a translation from EXPRESS to OWL. The approaches taken by these authors and their main contributions are explained below.

Intelligent Self-describing Technical and Environmental Networks (S-TEN) [13] is a project funded by the European Community. One of its objectives is to “exploit the Semantic Web for scientific and engineering applications.” This project describes a bi-directional translation between

EXPRESS and OWL.

To understand the approach adopted in the S-TEN project, we briefly trace the development of STEP APs. Different APs may require common parts. When the first APs were created, these parts were copied and adapted according to the goal of the AP. Currently APs are being migrated from their old versions to modular ones. The modularity ensures that the information available from an AP can be shared among the other APs. S-TEN focuses on translating modules, so the translated parts are used within several APs. Hence in the S-TEN project no AP is covered in full. The STEP modules are also modified, either to take advantage of the use of OWL, or as an improvement. For instance, some entities used to express relationships are directly translated to relationships. Moreover new capabilities are added, such as a better management of the product identifiers. A manual check is performed after the translation of the EXPRESS schemas to ensure that the meaning of the data models is the same in EXPRESS and in OWL. The final ontology is stored in a database.

The deliverables of the S-TEN project include a description of the bi-directional translation from EXPRESS to OWL and a list of the EXPRESS features that have not been translated yet (e.g., optional/mandatory attributes, aggregate bounds, lists, and arrays). As parts of the S-TEN project, tools have been developed to manage and access web-based databases, to convert EXPRESS schemas and instances to OWL, and to convert OWL data to EXPRESS [14].

Zhao and Liu, from the Zhongshan University in China, proposed a methodology to represent EXPRESS models in OWL and SWRL [15], a rule-based language for OWL. Their report describes the approach in two parts. In the first part, the mapping between EXPRESS schemas and OWL and SWRL is discussed. SWRL adds more capabilities to OWL by permitting rule-based inference over the ontology. In the second part, the combination of the OWL ontology and the SWRL rules is integrated into Jess [16], a Rule Engine for Java. As OWL, SWRL and the Jess rules can be described in EXtensible Markup Language (XML) [17], an EXtensible Stylesheet Language (XSL) [18] transformation is performed to achieve this integration. The different steps involved in the process are described. The authors also present a set of tools that can be used to query and reason over the ontology.

Zhao and Liu also translated procedural code contained in the EXPRESS schemas. The procedural code specifies algorithms which can be used to compute derived attributes or to check the validity of data. Since OWL is not a procedural language, the authors chose to use Jess rules to represent EXPRESS procedures and functions. However it is not clear whether this mapping between procedures and Jess rules could work for all the procedures, especially those from AP 203. Moreover, some aspects of the EXPRESS language are not properly dealt with. For instance, the translation of ordered lists in EXPRESS was not proposed. Automated tools doing the entire translation are planned, but we are not aware of any software released.

3 OntoSTEP

The goal of our work is improving interoperability of product data by defining the semantics of the STEP models in a formal logic. In this paper, we translate the EXPRESS models in OWL 2 [19]. OWL-DL, a sublanguage of OWL based on Description Logic, provides several features we need to add semantics:

- *consistency checking*: this mechanism ensures that no contradictions are present within the model
- *inference*: this capability allows to extract new knowledge through logic reasoning
- *decidability*: this characteristic ensures that the reasoning is performed in finite time.

In this section, we present the rules for translating EXPRESS to OWL. The translation of the instance data resulting from the EXPRESS schemas is also introduced.

For our examples, we use the STEP AP 203 data model as it is by far the most common EXPRESS schema used by Computer-aided design (CAD) systems to exchange product geometric information. The methodology followed for this AP is fully applicable to any other STEP APs.

3.1 Mapping the main concepts

We illustrate our translation of the main concepts in EXPRESS through the following example from AP203.

```
ENTITY product_category;
name        : label;
description : OPTIONAL text;
END_ENTITY; -- product_category

ENTITY product_related_product_category
SUBTYPE OF (product_category);
products : SET [1:?] OF product;
END_ENTITY; -- product_related_product_category

ENTITY product;
id                  : identifier;
name                : label;
description         : text;
…
END_ENTITY; -- product
```

In this example three entities are described: `product`, `product_category`, and `product_related_product_category`. A `product` has an `identifier`, a `name` and a `description`. A `product_category` contains a `name` and may have a `description`. A `product_related_product_category` is a `product_category` that identifies the products that satisfy the type identified by the category. In order to simplify this example, some parts of the actual AP203 entity definitions have been removed.

The concept of entity in EXPRESS is similar to the concept of a class in object-oriented modeling: entities can be seen as abstractions of real-world objects (instances) and can be organized in hierarchies. These hierarchies conform to the following inheritance principle: sub-entities (`product_related_product_category` in our example) inherit the attributes of their super-entities (`product_category` in our example) and the instances of the former are also instances of the latter. Attributes specify relationships between entities or between entities and data. An attribute consists of a name and a type: in our example, the first attribute of the entity `product` is called `id`, and its type is `identifier`. An attribute may be optional, as in the case of description in `product_category`, and its type can be a collection of data, as in the case of products in `product_related_product_category.`

In our translation, EXPRESS entities and instances map respectively to OWL classes and individuals. Attributes correspond to OWL properties -ObjectProperties link classes together, while DataProperties link classes to data types. The domain of a property defines which classes can have this property. Without restrictions, properties in OWL are aggregations, so an individual can be linked several times to other individuals by using the same property. To define the usage of a property, it is possible to restrict its cardinality through the “ObjectExactCardinality” construct and its values through the “ObjectAllValuesFrom” construct. In the case of an optional attribute, the

"ObjectAllValuesFrom" construct is used to link the entity to the union of the attribute type and the class owl:Nothing. This solution is adopted to explicitly express the semantics of the OPTIONAL keyword: a value is not required for this attribute.

An ontology may contain statements related to both classes (TBox) and individuals (ABox). In our translation, a schema is translated into an ontology that contains mainly classes and property definitions [20]. The following table summarizes our proposed translation of the basic concepts from EXPRESS to OWL.

Table 1: Translation of the basic concepts from EXPRESS to OWL

EXPRESS	OWL
Schema	Ontology
Entity	Class
Subtype of	Subclass of
Attribute with an entity type	ObjectProperty. The domain of the property is the class that corresponds to the entity that contains the attribute. This class is restricted to have ObjectExactCardinality equal to 1 and ObjectAllValuesFrom equal to the entity type for that property.
Attribute with a simple data type	DataProperty. The domain of the property is the class that corresponds to the entity that contains the attribute. This class restricted to have ObjectExactCardinality equal to 1 and ObjectAllValuesFrom equal to the data type for that property.
Optional attribute	The range of the property is restricted to have ObjectAllValuesFrom equal to the union of the attribute type and the class Nothing.
Attribute with an aggregation type	The range of the property is restricted to have, for that property, minimum and maximum cardinalities corresponding to the aggregation size.

We also need to redefine the naming conventions for the properties. Consider, as an example, the entities `product` and `product_category`, both having the `name` attribute. In EXPRESS attributes are defined to be within the scope of the entity. In OWL properties have a global scope, so the property `name` would be the same for the `product` and the `product_category`. We choose to prefix the attribute names with the entity names in order to differentiate attributes. As a result, the entities `product` and `product_category` will contain, respectively, the attributes `product_has_name` and `product_category_has_name`. The following OWL statements, expressed in functional syntax, are the translation of the previous entity definitions:

```
SubClassOf(product ObjectAllValuesFrom(product_has_description text))
SubClassOf(product ObjectExactCardinality(1 product_has_description))

SubClassOf(product ObjectAllValuesFrom(product_has_name label))
SubClassOf(product ObjectExactCardinality(1 product_has_name))

SubClassOf(product ObjectAllValuesFrom(product_has_id identifier))
SubClassOf(product ObjectExactCardinality(1 product_has_id))


SubClassOf(product_related_product_category
ObjectAllValuesFrom(product_related_product_category_has_products product))

SubClassOf(product_related_product_category product_category)
SubClassOf(product_related_product_category ObjectMinCardinality(1
product_related_product_category_has_products))

SubClassOf(product_category ObjectAllValuesFrom(product_category_has_name label))
```

```
SubClassOf(product_category ObjectExactCardinality(1 product_category_has_name))

SubClassOf(product_category ObjectAllValuesFrom(product_category_has_description
ObjectUnionOf(owl:Nothing text)))
SubClassOf(product_category ObjectExactCardinality(1 product_category_has_description))
```

3.2 Mapping instances

An EXPRESS schema is instantiated by creating a file as defined in “Clear Text Encoding of the Exchange Structure -10303-21,” or Part 21. CAD packages can export data in STEP format that conform to the AP203 schema and conform to STEP Part 21’s constraints. In this paper we refer to these export files as “Part 21 files.” A Part 21 file that contains instances of the example previously introduced is depicted below.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(
/* description */ ('example'),
/* implementation_level */ '2;1');

FILE_NAME(
/* name */ 'gear',
/* time_stamp */ '2008-01-01T00:00:00-05:00',
/* author */ ('Sylvere Krima'),
/* organization */ ('NIST'),
/* preprocessor_version */ ' ',
/* originating_system */ ' ',
/* authorisation */ ' ');

FILE_SCHEMA (('CONFIG_CONTROL_DESIGN'));
ENDSEC;

DATA;
#1=PRODUCT_RELATED_PRODUCT_CATEGORY('part',$,(#2));
#2=PRODUCT('Gear','Gear','This is a gear');
...
ENDSEC;
END-ISO-10303-21;
```

A Part 21 file includes a header and a data section. The header contains meta-data such as the file’s author and the schema reference. In our example the schema is the AP203: Configuration Controlled Design. The data section contains objects that are instances of the entities defined in the schema. The above Part 21 file defines two objects, each is identified by a number. The type of the instances is supplied together with the value of its attributes. For example, the first object has the identifier 1 and is an instance of `PRODUCT_RELATED_PRODUCT_CATEGORY`. Its first two attributes are related to the super-entity `PRODUCT` while the third attribute is specific to the `PRODUCT_RELATED_PRODUCT_CATEGORY` entity. The string `'part'` is the value of the attribute `name` while the attribute `description`, declared as optional in the schema, does not contain any data ($ symbol). The third attribute type is an aggregation (surrounded by the parenthesis) and contains only the object 2. This object 2 is declared in a similar manner: it is a `product` containing a `name`, an `id`, and a `description`.

The translation to OWL is similar to the process described in the previous section and summarized in the previous table. In STEP the schema and the instances are declared in different files:

the related schema is specified in the Part 21 file in the FILE_SCHEMA section. OWL provides a similar mechanism of import. The instance file contains an import statement that relates instances to the schema ontology. This import mechanism allows us to maintain the schema ontology separate from the instance one. By having the final ontology containing both the TBox and the ABox, we are able to check the consistency of the instances against the schema. The namespace of the elements declared in the schema ontology indicates the shortened name of the schema: ap203 in our example.

While STEP considers all instances to be different, OWL does not have the unique name assumption, i.e., in OWL a same object can be identified with two different names. The solution to capture the semantics of EXPRESS is to declare all the created individuals as different. Our OWL version of the previously introduced Part 21 file is:

```
ClassAssertion(example:1 ap203:product_related_product_category)
ObjectPropertyAssertion(ap203:product_category_has_name example:1 example:_label_Name_1)
ObjectPropertyAssertion(ap203:product_related_product_category_has_products example:1
example:2)

ClassAssertion(example:_label_Name_1 ap203:label)
DataPropertyAssertion(ap203:to_string example:_label_Name_1 "part"^^xsd:string)

ClassAssertion(example:2 ap203:product)
ObjectPropertyAssertion(ap203:product_has_id example:2 example:_identifier_Id_2)
ObjectPropertyAssertion(ap203:product_has_description example:2
example:_text_Description_2)
ObjectPropertyAssertion(ap203:product_has_name example:2 example:_label_Name_2)

ClassAssertion(example:_identifier_Id_2 ap203:identifier)
DataPropertyAssertion(ap203:to_string example:_identifier_Id_2 "Gear"^^xsd:string)

ClassAssertion(example:_text_Description_2 ap203:text)
DataPropertyAssertion(ap203:to_string example:_text_Description_2 "This is a
gear"^^xsd:string)

ClassAssertion(example:_label_Name_2 ap203:label)
DataPropertyAssertion(ap203:to_string example:_label_Name_2 "Gear"^^xsd:string)
```

The treatment of an unknown fact is another major difference between EXPRESS and OWL. In EXPRESS, any unknown fact is supposed to be false. For example, if an instance of `product` is not known to be instance of `product_category`, the system assumes it is not. This behavior is called the Close World Assumption (CWA), because it supposes that the world is limited to what is stated. OWL uses the Open World Assumption (OWA): unless a reasoner proves a fact is false, that fact is unknown. Hence the translation sometimes requires additional information to capture the semantics of EXPRESS in OWL. The difference between CWA and OWA causes a translation problem when an instance is constrained to have one attribute. The attribute `id` of the entity `product` is not declared optional, so it should be instantiated for all the instances of `product`. In the EXPRESS logic, the lack of data will raise an error. In OWL, even if we do not define an `id` for an instance of `product`, the reasoner does not detect an inconsistency: the instance is still considered to have an unknown id. To allow the reasoner to detect an inconsistency in case of missing `id`, it would be required to declare explicitly that that instance of `product` has no `id`.

To fully translate the STEP APs, the translation of some additional concepts, such as derived data types, is required to be introduced. Our proposed translation from EXPRESS to OWL for these additional concepts is presented in the next section.

3.3 Mapping additional concepts

Let us now consider some additional concepts of EXPRESS and, when possible, propose their translation in OWL. Unfortunately, some constructs of EXPRESS, such as functions, cannot be automatically translated: these constructs usually define entity constraints and attributes computation and may rely on complex algorithms (see Section 2) OWL, as it is based on Description Logic, does not contain any procedural aspects. This section focuses on the EXPRESS language aspects that can be automatically translated to OWL concepts.

3.3.1 Data types

In our previous schema example the types `label`, `identifier`, and `text` are derived from the simple type `string`. EXPRESS defines other simple data types to cover a wide range of information. These simple types are presented and the definition of derived types is explained.

3.3.1.1 Simple data types

EXPRESS includes all the data types required to capture the common product information. OWL inherits the data types defined in the XML Schema Definition (XSD) language. Table 2 presents the equivalence between the EXPRESS types and the XSD types.

Table 2: Translation of simple data types from EXPRESS to OWL

EXPRESS type	Example or possible values	XSD equivalent
`number`	2.33	`decimal`
`real`	2.33	`double`
`integer`	2	`integer`
`logical`	true, false, or unknown	---
`boolean`	true or false	`boolean`
`string`	“Hello world”	`string`
`binary`	101011	---

In EXPRESS, some types, like `boolean` and `string`, have the exact equivalent in OWL, while other types, like `number` and `real`, are represented in a slightly different way in OWL. For example, we translate the `real` data type in EXPRESS as a `double` in OWL, even if the precision of those two data types is different. This solution should not lead to major problems since a 32-bit approximation of `real` numbers is usually sufficient in the product domain.

A few problems arise when translating the logical and binary data types. In EXPRESS, the `logical` type allows the assertion of a true, false or unknown value. XSD does not provide any similar data type. The `binary` type is used to store a sequence of 0 and 1. With XSD it is possible to restrict the data value to allow only 0s and 1s, but there is no way to specify that this sequence is a number in base 2. A possible solution would be to store the `binary` number using only an unsigned integer. Fortunately, since these two data types are not present in the AP203, the missing translation of them has no influence on our work.

3.3.1.2 Constructed data types

EXPRESS allows the creation of data types derived from the simple types previously presented.

First, user-defined data types are used to differentiate the kind of information being stored. In our previous example, both `text` and `label` refer to the `string` type, but are used in a different context. The following example shows the definition of the type `label`, which refers to a `string`.

```
TYPE label = STRING;
END_TYPE;
```

Second, user-defined data types can be used to restrict the simple types. For instance, the type `day_in_week_number` in AP203 is defined as an integer between 1 and 7.

```
TYPE day_in_week_number = INTEGER;
WHERE
wr1: ((1 <= SELF) AND (SELF <= 7));
END_TYPE;
```

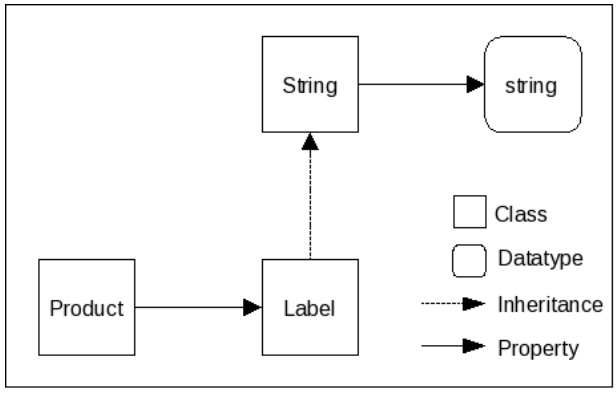

Figure 2: Attributes

In order to deal with these derived types in OWL, we build a type hierarchy and we apply the concept of data wrapping (see Figure 2.) In this example, we define a class `String` that has a DataProperty to the `string` data type. It is then sufficient to subclass the class `String` to translate all the user-defined data types related to `string` (`Label` in this case). This concept organization allows us to translate all the user-defined data types related to `string` only by subclassing the class `String`.

Because of the possible use of functions, we cannot guarantee the correctness of an automatic translation of data type restrictions. Using a manual case-by-case translation, most of the types defined in AP203 can be translated. As an example, the following OWL statements show how the type `day_in_week_number` is translated:

```
SubClassOf(integer attribute)
DataPropertyDomain(to_integer integer)
DataPropertyRange(to_integer xsd:integer)

SubClassOf(day_in_week_number integer)
DatatypeRestriction(dataRange minInclusive “1”^^xsd:integer)
DatatypeRestriction(dataRange maxInclusive “7”^^xsd:integer)

SubClassOf( day_in_week_number DataSomeValuesFrom(to_integer
DatatypeRestriction(xsd:integer maxInclusive "7"^^xsd:integer minInclusive
"1"^^xsd:integer)))
```

3.3.2 Aggregations

EXPRESS provides four different types of aggregations. Each one has a different management of the content: set, bag, list, and array. Each type of aggregation has order policies and duplication policies. For the attribute declarations, the type of content and the number of elements of the aggregation are defined.

3.3.2.1 Set

Set is chosen for ordered aggregation when the duplication of the aggregated elements is not permitted. This is the case of our first example, where a `product_related_product_category` contains a collection of `products`. ObjectProperty in OWL is also an unordered set of unique values.

However, because of the OWA, cardinalities restrictions can rarely be checked. For example, if a `product_related_product_category` is linked to two `products`, the OWA assumes these two `products` could be the same. This case is resolved by explicitly declaring all the individuals as different. Moreover, the OWA assumes the ontology to be incomplete, so new products could be added in the future. To get the same behavior in OWL-DL as in EXPRESS we would need to assert that the relationship involves exactly two individuals. In that case no new individuals can be added in the future and the cardinality is checked correctly.

3.3.2.2 Bag

Bags are unsorted collection of elements. The only difference between sets and bags is the duplication policy: the same element can be repeated several times in a bag. Because object properties in OWL do not allow duplications, we create the following concepts structure (Figure 3.)

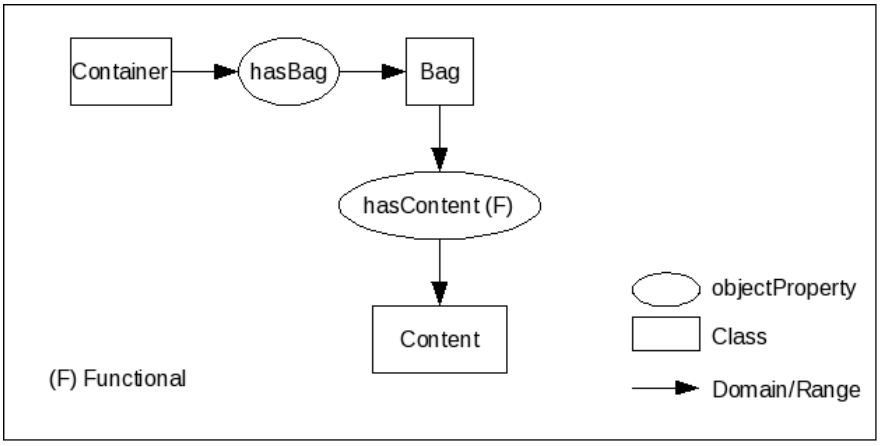

Figure 3: Bag (Class level)

A new class, called `Bag`, is inserted between the `Container` class and the `Content` class. The property `hasContent` is declared functional in order to associate only one element for each instance of `Bag`.

Figure 4 represents the instantiation of the schema presented in Figure 3. An instance of the container (`cont`) is linked to two different instances of the new class (`b1` and `b2`). Each of these two instances is then linked to the same instance of the content class (`elem1`). The ontology contains the fact that the `elem1` element is present twice in the aggregation since `b1` and `b2` are different.

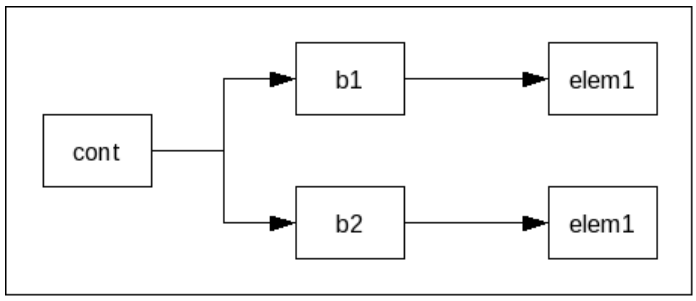

Figure 4: Bag (Individual level)

3.3.2.3 List and Array

Lists and arrays are collections of ordered elements, unlimited in the case of lists and fixed in size in case of arrays. The bounds of a list define the minimum and maximum number of elements, while the bounds of an array define the range of index that addresses its elements. Since both lists and arrays refer to ordered collections, we choose the same data structure to translate both.

Consider, as an example, the definition of a Cartesian point as a list of one to three coordinates, depending on the dimension. The order of the coordinate's numbers is essential to identify the point.

```
ENTITY cartesian_point
SUBTYPE OF (point);
coordinates : LIST [1:3] OF length_measure;
END_ENTITY; -- cartesian_point
```

Our proposed solution to represent ordered aggregations in the ontology is inspired from [21]. The idea is to build a chained list of elements. To achieve this goal, we create a class `List` that is linked to its content and to the next `List`. The end of the chain is defined by an `EmptyList`, a subclass of `List`.

Figure 5 presents the classes and the properties involved in this solution. Because the names in our translation are long, we use a short form in our explanations (see Table 3.)

Table 3: Mapping between the names in short and long forms

Short form	Long form
Point	`cartesian_point`
List	`list_of_length_measure`
Coordinate	`length_measure`
isFollowedBy	`list_of_length_measure_is_followed_by`
EmptyList	`emptylist_of_length_measure`
hasContent	`list_of_length_measure_has_content`
hasNext	`list_of_length_measure_has_next`
hasElement	`list_of_length_measure_has_element`
hasCoordinate	`cartesian_point_has_coordinates`
hasCoordinate2	`cartesian_point_has_coordinates2`

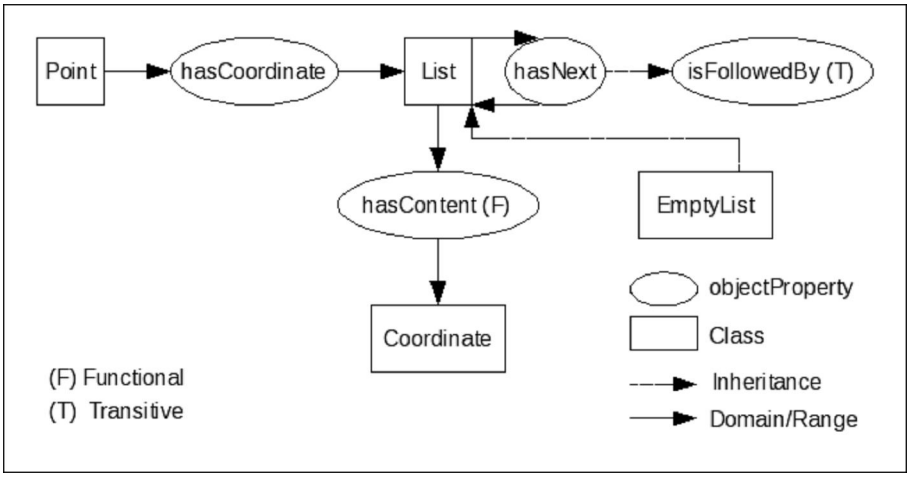

Figure 5: list (Class level)

In this example, the class `Point` is linked to the class `List` with the ObjectProperty `hasCoordinate`. The `List` has a link to the next `List` (`hasNext`) and also to the content (`hasContent.`) `hasNext` is a sub property of `isFollowedBy`, so that if `hasNext` occurs, `isFollowedBy` will do so. Moreover this last property is transitive, so for a given list, `hasNext` links the `List` to its imminent next `List`, while `isFollowedBy` to all the following `List`s. `EmptyList` is defined as a `List` that has no next element and no content.

The following statements are used to create the list structures in OWL.

```
SubClassOf(cartesian_pointObjectAllValuesFrom(cartesian_point_has_coordinates
list_of_length_measure))

SubClassOf(list_of_length_measure
ObjectAllValuesFrom(list_of_length_measure_is_followed_by list_of_length_measure))

EquivalentClasses(emptylist_of_length_measure
ObjectIntersectionOf(ObjectComplementOf(ObjectSomeValuesFrom(list_of_length_measure_is_foll
owed_by owl:Thing)) list_of_length_measure))
SubClassOf(emptylist_of_length_measure list_of_length_measure)
SubClassOf(emptylist_of_length_measure ObjectMaxCardinality(0
list_of_length_measure_has_content))

SubObjectPropertyOf(list_of_length_measure_has_next
list_of_length_measure_is_followed_by)

ObjectPropertyDomain(cartesian_point_has_coordinates cartesian_point)

ObjectPropertyDomain(list_of_length_measure_has_content list_of_length_measure)
FunctionalObjectProperty(list_of_length_measure_has_content)
ObjectPropertyRange(list_of_length_measure_has_content length_measure)

ObjectPropertyDomain(list_of_length_measure_is_followed_by list_of_length_measure)
TransitiveObjectProperty(list_of_length_measure_is_followed_by)
ObjectPropertyRange(list_of_length_measure_is_followed_by list_of_length_measure)
```

To this pattern we add new axioms to link the content of the `List`s directly to the `Point`. From the following example in Figure 6, which describes a concrete point, we add some chained

properties that achieve this goal.

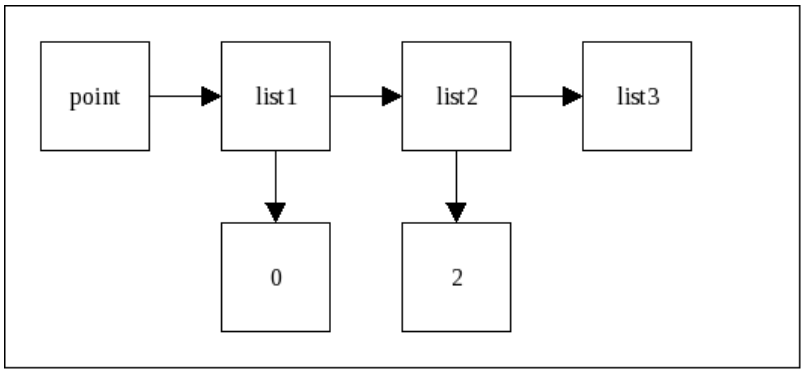

Figure 6: List (Individual level)

`point`, an instance of `cartesian_point`, contains a list of coordinates: 0 and 2. It is linked to `list1`, which has as content the value 0. `list1` also contains the next list, `list2`. In the same manner, `list2` has as content the value 2 and is linked to the `EmptyList` `list3`.

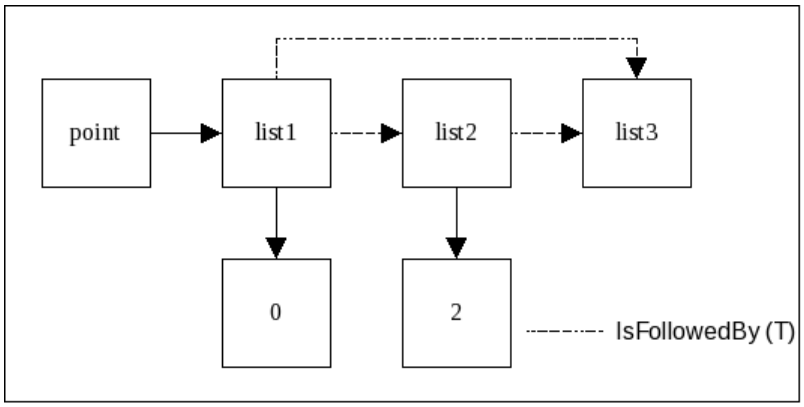

Figure 7: List (Individual level)

As `isFollowedBy` is a super property of `hasNext` and is transitive, `list1` is linked to `list2` and `list3` through `isFollowedBy` (see Figure 7.)

The first property chain, called `hasElement`, is created as the composition of `isFollowedBy` and `hasContent`.

```
SubObjectPropertyOf(SubObjectPropertyChain(list_of_length_measure_is_followed_by
list_of_length_measure_has_content) list_of_length_measure_has_element)
```

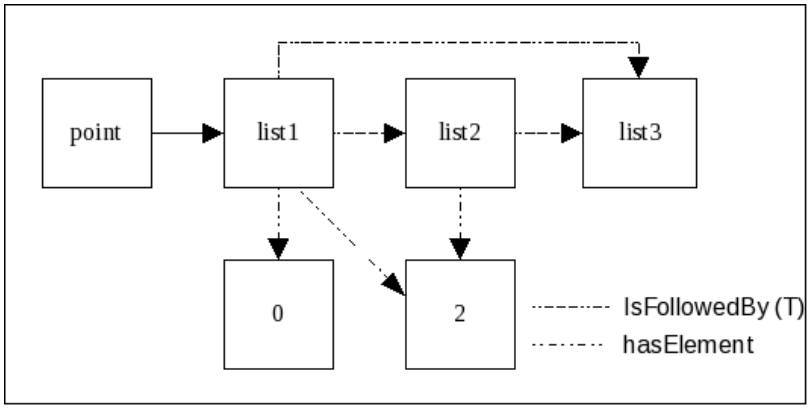

Figure 8: List (Individual level)

The result is `list1` being linked to all the coordinates values (see Figure 8.)

Finally the second property chain is created as the composition of `hasList` and `hasElement`.

```
SubObjectPropertyOf(SubObjectPropertyChain(cartesian_point_has_coordinates
list_of_length_measure_has_element) cartesian_point_has_coordinates2)
```

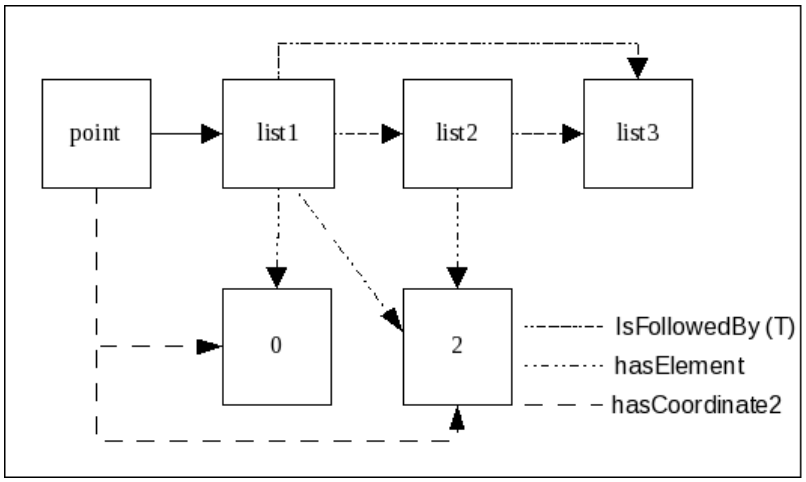

Figure 9: List (Individual level)

At this point, the two coordinate values are directly linked to the point (see Figure 9.)

If this pattern is used correctly, it is possible to check the number of elements. The idea is to assert the different possibilities for the structure. A point can contain, depending on whether it is in 2D or 3D space, from two to three coordinates. Therefore, the first two elements are mandatory, the third is optional and an `EmptyList` shall end the `list`. The difference between a `List` containing an element and an `EmptyList` is that the former has "`has_next`" property while the second has not. The goal is to force the use of a container for the first n elements, where n is a lower bound. Then, if the `List` has an upper bound, force the choice of either a container or an `Empty_list` up to this limit. Two expressions are required: one on the existence and one on the universality.

```
SubClassOf(cartesian_point ObjectAllValuesFrom( cartesian_point_has_coordinates
ObjectAllValuesFrom(list_of_length_measure_has_next
```

```
ObjectAllValuesFrom(list_of_length_measure_has_next
ObjectUnionOf(ObjectAllValuesFrom(list_of_length_measure_has_next
emptylist_of_length_measure) emptylist_of_length_measure)))))

SubClassOf(cartesian_point ObjectSomeValuesFrom( cartesian_point_has_coordinates
ObjectSomeValuesFrom(list_of_length_measure_has_next
ObjectSomeValuesFrom(list_of_length_measure_has_next
ObjectUnionOf(ObjectSomeValuesFrom(list_of_length_measure_has_next
emptylist_of_length_measure) emptylist_of_length_measure)))))
```

This pattern has some limitations: in OWL the bounds are specified at the level of the entity, but in EXPRESS they are defined at the level of the aggregation type. Moreover the formula can be very long if the bounds are high.

3.3.3 Select

SELECT is a keyword used in EXPRESS to choose from a set of different types.

```
ENTITY placement
SUPERTYPE OF (ONEOF (axis1_placement,axis2_placement_2d,
axis2_placement_3d))
SUBTYPE OF (geometric_representation_item);
...
END_ENTITY;

TYPE axis2_placement = SELECT
(axis2_placement_2d,
axis2_placement_3d);
END_TYPE;
```

In this example, there are three different kinds of placement available in the schema and the `axis2_placement` SELECT type chooses between `axis2_placement_2d` and `axis2_placement_3d`. To translate the keyword SELECT to OWL we create a new class (`axis2_placement`) corresponding to the SELECT type, and then define this class as the equivalent of the “selected” entities:

```
EquivalentClasses(axis2_placement ObjectUnionOf(axis2_placement_2d axis2_placement_3d))
```

The equivalence asserts that no other classes can be added to this type in the future.

3.3.4 Enumeration

Enumerations are defined in EXPRESS as a finite set of values. Consider, as an example, this enumeration:

```
TYPE ahead_or_behind = ENUMERATION OF
(ahead,
behind);
END_TYPE; -- ahead_or_behind
```

In this example, an attribute having a type `ahead_or_behind` will only contain either `ahead` or `behind`. The equivalent translation in OWL is obtained through the OneOf construct: it enables us to define a class by exhaustively enumerating its individuals.

```
EquivalentClasses(ahead_or_behind ObjectOneOf(behind ahead))
```

The values are represented by the individuals ahead and behind, and the class `ahead_or_behind` is composed exactly of these individuals.

3.3.5 Abstraction

A supertype in EXPRESS may be declared as abstract. The meaning is the same as in object-oriented programming: an abstract entity cannot be directly instantiated. Consider, as an example, the following entities:

```
ENTITY document_reference
ABSTRACT SUPERTYPE;
assigned_document : document;
source            : label;
END_ENTITY;
ENTITY cc_design_specification_reference
SUBTYPE OF (document_reference);
items : SET [1:?] OF specified_item;
END_ENTITY; -- cc_design_specification_reference
```

The entity `document_reference` and its subtype `cc_design_specification_reference` are defined in the AP203: the subtype not only contains an `assigned_document` and `source` but also a set of `items`.

OWL does not provide any feature to translate the ABSTRACT keyword and, even if it had, it would not work as expected. Because of the OWA, an ontology is assumed to be incomplete so that the non-instantiation of a concrete entity does not lead to inconsistency. To overcome this problem, we could have declared the subtypes classes as the partition of the supertype. A partition forces the instances of the supertype to belong to at least one subtype. This is achieved by declaring that the set of instances of the supertype is covered by the sets of instances of the subtypes. In that case, if an individual was declared as an instance of `document_reference` and not an instance of `cc_design_specification_reference`, the reasoner would detect an inconsistency. However this solution only works when both the supertype and all the subtypes are declared within the same schema. Because of these reasons, we choose to ignore the ABSTRACT keyword. It is then impossible for the reasoner to check that abstract entities are not directly instantiated.

3.3.6 Inheritance

Because it is intended to meet product data modeling requirements, EXPRESS can represent complex inheritance relationships. In the interest of brevity we do not give details on advanced EXPRESS language features in this paper and instead refer readers to [8].

In order to specify the allowed combination of subtypes for an entity, EXPRESS provides three keywords: ONEOF, ANDOR, and AND. Along with the ABSTRACT keyword, they restrict the usage of the instantiation mechanism. When the subtype definition occurs in EXPRESS, the involved subtypes are explicitly enumerated, and no other subtype can be declared. In OWL, defining the super-entity as equivalent to a specific combination of its sub-entities respects this meaning.

3.3.6.1 ONEOF

The ONEOF keyword takes as parameter a list of entities and it specifies that only one of these entities can be instantiated. Consider, as an example, the definition of the `conic` entity:

```
ENTITY conic
```

```
SUPERTYPE OF (ONEOF (circle,ellipse,hyperbola,parabola))
SUBTYPE OF (curve);
position : axis2_placement;
END_ENTITY; -- conic
```

In this example, an object instance of `conic` has to be also an instance of `circle`, `ellipse`, `hyperbola`, or `parabola`. No combination is allowed, for example, this object cannot be a `circle` and an `ellipse` at the same time.

An equivalent behavior in OWL is obtained by defining the subclasses as disjoint: an inconsistency is detected when an individual is an instance of more than one of these subclasses. We mark the set of classes contained in a ONEOF as all disjoint.

```
EquivalentClasses(conic ObjectUnionOf(circle ellipse hyperbola parabola))
DisjointClasses(circle ellipse hyperbola parabola)
```

Another solution could be to use the logical definition of XOR. The following example shows how the formula is obtained for two and three elements:

```
(Circle or ellipse) and not (circle and ellipse)
```

```
((Circle or ellipse) and not (circle and ellipse) or hyperbola) and not ((Circle or ellipse) and not (circle and ellipse) and hyperbola)
```

Such a pattern can be translated in OWL by using the intersection, the union and the complement to translate `and`, `or`, and `not`. However this increases the complexity of the ontology, as the length of the formula increases drastically with the number of elements involved. For this reason we choose the first solution.

3.3.6.2 ANDOR

When no specific constraints are defined, the default keyword for the instantiation is ANDOR: the instance can belong to more than one subclass. Consider, as an example, the declaration of the `address` entity, supertype of both `organizational_address` and `personal_address`:

```
ENTITY address;
...
END_ENTITY;

ENTITY organizational_address
SUBTYPE OF (address);
organizations : SET [1:?] OF organization;
description   : text;
END_ENTITY;

ENTITY personal_address
SUBTYPE OF (address);
people      : SET [1:?] OF person;
description : text;
END_ENTITY;
```

In this example, the entity `address` could have also been declared in the following way:

```
ENTITY address
SUPERTYPE OF (organizational_address ANDOR personal_address);
...
```

An instance of `address` can be instance of `organizational_address`, `personal_address`, or both. In OWL a set of entities joined by an ANDOR is translated by a union of the corresponding classes in OWL.

```
EquivalentClasses(address ObjectUnionOf(organizational_address personal_address))
```

In this example, we first represent the union of the subclasses by using the `ObjectUnionOf` construct and we then declare this union to be equivalent to the parent class.

3.3.6.3 AND

The AND operator imposes that the object be an instance of all the subclasses. Consider a different version of the previous example:

```
ENTITY address;
SUPERTYPE OF (organizational_address AND personal_address);
...
```

END_ENTITY;

In this new example, an address must be both organizational and personal.

In order to respect this constraint in OWL, we use the ObjectIntersectionOf to link the subclasses.

```
EquivalentClasses(address ObjectIntersectionOf(organizational_address personal_address))
```

ObjectIntersectionOf construct restricts the individuals of address to be both organizational and personal addresses.

3.3.7 Uniqueness

Entities can contain UNIQUE clauses: they are composed of one or more attributes that form a key. Two different instances with the same key are not allowed. The id property of a product is an example of a UNIQUE attribute: two product instances with the same id represent the same product.

In a Part 21 file, all the instances are supposed to be different. As previously stated, we declare OWL individuals having the same specific attribute value to be the same. If two individuals have the same attributes, this would lead to incoherence. The two possible ways to achieve this goal are either using SWRL rules, or to using the "key" construct that will be part of the final version of OWL2 [22]. The unique key is not currently translated.

3.4 Benefits

OWL-DL semantics is based on Description Logic (DL), a family of knowledge representation languages. These languages are used to define domain concepts according to a predefined and well understood formalism. Concepts are used to represent the domains objects, while roles are used to represent relationships between these concepts. Concepts and roles are the main components of the knowledge base[1].

OWL DL provides the "maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time)" [2]. Expressiveness, computational completeness, and decidability enable reasoning mechanisms, e.g., consistency checking. These mechanisms are applied by reasoners to find implicit

[1] The terms "knowledge base" and "ontology" are used interchangeably for the purpose of this paper.

consequences based on the explicit information provided in a knowledge base.

Many reasoners have already been developed. For the purpose of our project we choose Pellet [23].

3.4.1 Consistency checking

The consistency checking procedure can be applied at two different levels: schema level and instances level. At the schema level, the consistency checking procedure checks whether a concept can be instantiated at least once. At the instances level, the consistency checking procedure checks whether an individual declared as an instance of a concept is really instance of that concept.

At the schema level, the reasoner uses the concept definitions to determine whether the schema is consistent or not. We present here an example of an inconsistent schema:

- `Product`, `Gear`, `Category`, and `MechanicalProduct` are classes,
- `Gear` is a sub-class of `Product`,
- `hasCategory` is an ObjectProperty,
- `Category` and `MechanicalProduct` are disjoints,
- All instances of `Gear` are connected to only one instance of `MechanicalProduct` through `hasCategory`
- All instances of `Product` are connected to only one instance of `Category` through `hasCategory`

`Category` and `MechanicalProducts` being disjoints, an instance of `Gear` being connected to an instance of `MechanicalProduct` cannot be an instance of `Product`, which should be connected to an instance of `Category`. In this particular situation the reasoner finds an inconsistency concerning the class `Gear`.

At the instance level, the reasoner uses the individual values to determine whether an individual declared as an instance of a class is consistent with the definition of that class. We present here an example that includes an inconsistent instance:

- `Product` is a class,
- `hasCategory` is a DataProperty, connecting a product with its category id represented as an integer,
- `product#1` is an instance of `Product`, connected through `hasCategory` to the value `'vehicle'`

In this example, since the property `hasCategory` is wrongly used to link a product with a string, the reasoner declares `product#1` as inconsistent.

Currently, libraries are available to check the consistency of EXPRESS schemas and Part21 files but with OntoSTEP, both kinds of consistency checking are performed by a DL reasoner. Checking the logical consistency of the OWL classes and individuals resulting from the translation is the necessary condition to use an inference procedure.

3.4.2 Inference procedure

An inference procedure uses the data evidence in a context and draws conclusions using certain problem solving strategies [24]. The inference procedure is the process to reach these conclusions and it is performed by a reasoner. Reasoners use a knowledge base as a source of data: concepts, roles, and axioms are elaborated by the reasoner to reach the conclusion. The expressivity of the axioms and concepts definitions is dependant on the logic language used.

OWL2 is based on a SROIQ(D) [19] expressivity. All the operators included in the SROIQ(D) expressivity e.g., transitivity, can be used and combined to express axioms. These axioms can be computed only with a reasoner that supports SROIQ(D) expressivity, e.g., Pellet.

In our work, for example, we use inference procedures to represent ordered lists of instances, i.e., to elaborate functional properties, transitive properties, properties hierarchies, and properties chains (see Section 3.3.2.3). All the axioms used in this representation are provided by the SROIQ(D) expressivity.

Once the reasoner has applied all the inferences procedures on our ontology, new knowledge and data become available. Then one can use a querying mechanism to query these new data, which represents an enriched version of the original ontology.

3.4.3 Queries

Queries are performed to retrieve specific data from a large amount of information: in our case we perform queries to retrieve some specific product information from a CAD file. The information contained in a CAD file is first translated into OWL representation, then checked for consistency and inference, and finally queried.

There are two approaches in vogue today to perform queries on OWL ontologies: the first approach uses a language called SPARQL Protocol and RDF Query Language (SPARQL) [25]. while the second approach uses the Semantic Query-Enhanced Web Rule Language (SQWRL) [26]. None of them is used in OntoSTEP for the reasons explained below.

SPARQL was specifically developed for Resource Description Framework (RDF) models, so we would need to translate our OWL ontology to RDF before performing SPARQL queries. We do not adopt this solution because of two reasons. First, the translation from OWL to RDF increases the computational time. Second, the Pellet reasoner does not support some SPARQL built-ins functions, such as DESCRIBE, OPTIONAL, or FILTER [27], or some classical aggregation functions, such as maximum, minimum, sum, or average.

While SPARQL was developed for RDF, SQWRL was specifically being developed for OWL. Unlike SPARQL, SQWRL is based on SWRL [12] and does not need any RDF bridge: the computation of SQWRL queries is then faster. SQWRL provides not only many built-in functions, but also some classical aggregation functions like maximum, minimum, sum, or average [28], which are missing in SPARQL. We do not adopt this solution for two reasons. First, only a proprietary engine, i.e., Jess, is currently available to process SQWRL queries. Second, SQWRL does not allow combining functions together.

To overcome these drawbacks we choose to perform our queries by using the OWL Application Programming Interface (API) [29] , which is a Java API. This API enables us to manipulate our ontology and to query it. The next section provides more details about this API and its usage.

4 Implementation

The previous section discussed how to generate an ontology from EXPRESS schemas and instance files. This part presents an implementation of the translation rules previously described. The goal is to create tools that perform this generation automatically, and then use these tools to translate both the AP203 and Part 21 CAD files. Three kinds of technologies are used to achieve the above goal: 1) those related to the EXPRESS schemas translation, 2) those related to the instances translation, and 3) those related to a web application that facilitates the use of our tools. The Java language is used in our implementation.

4.1 Schemas

The process of translating an EXPRESS schema file is done in two stages. In the first stage we retrieve the information contained in the file, and in the second stage we translate this information by applying the rules previously presented.

In order to obtain the structure of EXPRESS schemas, we use an open source EXPRESS Parser [30]. This parser is implemented using the ANTLR [31] parser generator: from EXPRESS grammar rules, ANTLR creates a parser that retrieves the structure of any EXPRESS schema. The result is a syntax tree representing the information contained in the schema.

ANTLR also provides facilities to scan syntax trees and to trigger specific actions depending on the encountered element. For instance, in our implementation, the detection of the keyword ENTITY leads to the creation of a class in OWL.

OWL API for OWL2 is the open library used to create the ontology.

The following figure sums up the process:

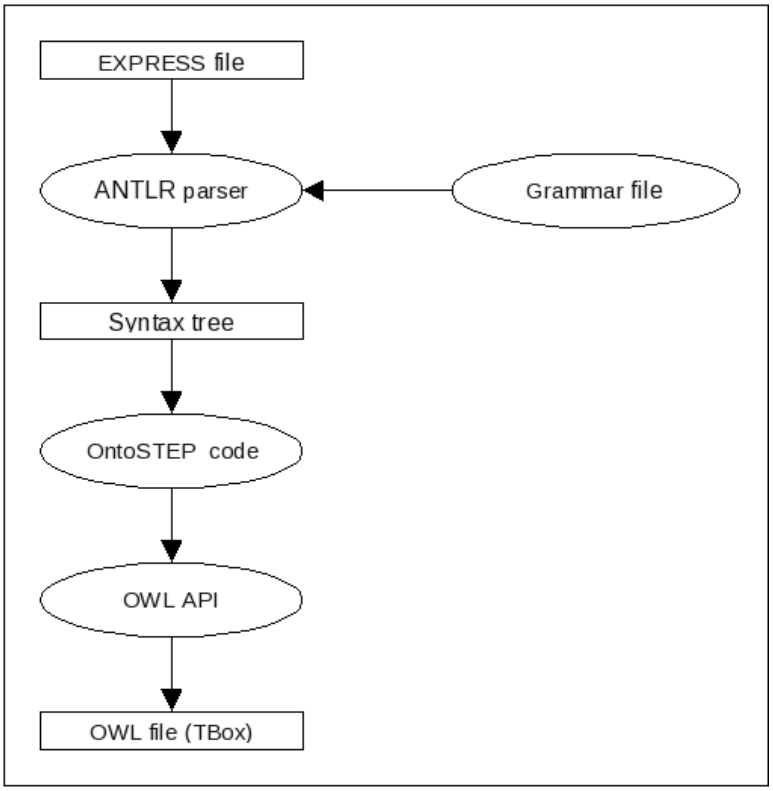

Figure 10: Schema translation process

4.2 Instances

The second part of our work is mapping Part 21 files to OWL. In STEP there is a mechanism called Standard Data Access Interface (SDAI) [32] to manage data defined in EXPRESS schemas. Bindings to several popular languages (C, C++, and Java) are specified. We used the SDAI implementation from STEPTools [33] to translate instances into OWL individuals and properties (ABox.) For each instance, an individual is created and for each type of this instance, the attributes are obtained and translated.

The result is a file containing all the assertions on the individuals (ABox.) The T-Box translation of the EXPRESS schema is also imported as it contains the definition of the OWL classes.

4.3 Web Application

The last step of our work is the creation of an interface for the tools previously described. A web application is built to allow users to see and manipulate the translation of their CAD files. This application can be seen as a product repository, as the user can manage all the files uploaded in previous sessions. The different services offered to the users are implemented using the Google Web Toolkit framework [34]. We use this framework to create our Java web application and to compile it into an optimized JavaScript [35] code. This JavaScript code is compatible with common browsers.

The web application contains basic login capabilities and uploads a CAD file. Once the file is retrieved, the translation process is launched, and the resulting file containing the A-Box is created. The user can view all the files previously uploaded, and load them to view their content. The classes and instances related to a product can be accessed. A query engine selects the products and parts corresponding to the criteria input by the user.

5 Use case

In the previous sections we described the concepts and the technologies we used to create a mapping from STEP to OWL-DL. The expressivity of the language and constructs used in the ontology allows us to perform reasoning and queries on the STEP models and on their instantiations.

In the use case shown in this section we describe a user interaction to obtain and process the OWL translation of three STEP output files generated by CAD systems: 4pinplug.stp, test_step.stp, and test_step2.stp. Through the web application, the user can translate and visualize his/her files, create and manage his/her own repository, and query his/her files. Figure 11 represents the main view of the web application. It is composed of two tabs located at the top, which provide different capabilities. The first one (Ontology) is used to show information about the ontology, to log in to the system, and to manage the files while the second one (Visualization) provides the information specific to the file selected on the first tab.

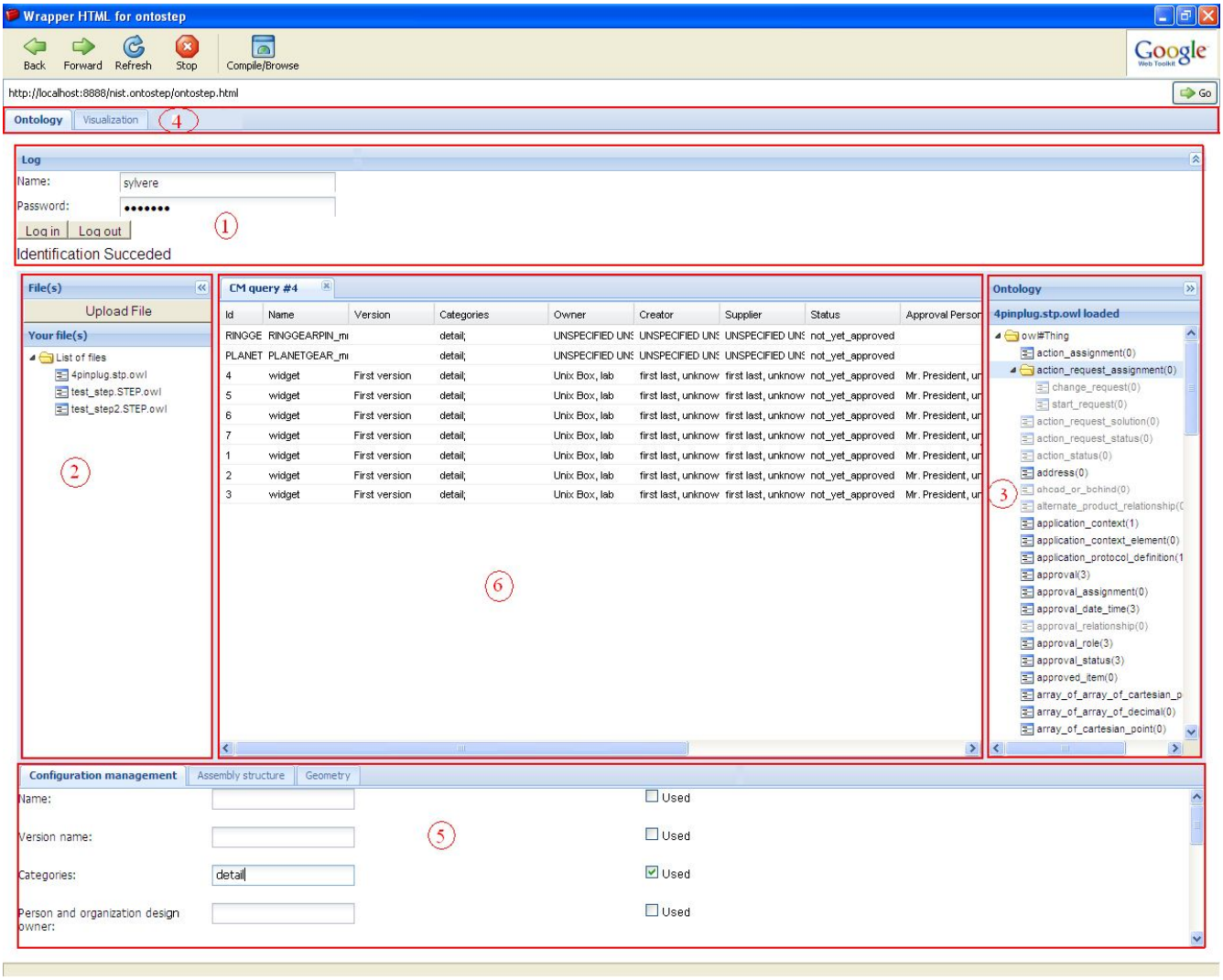

Figure 11: Web application

The user first extracts the Part 21 files from the CAD system and then runs the web application. The interaction is described below (see Figure 11, where bullet numbers refer to the circled numbers):

1) The user logs in to the system: the log in mechanism allows us to track the files uploaded by each user.

2) The user uploads files, and the application translates them. The “upload file” button opens a file dialog that enables the user to upload several files at the same time. Once uploaded and translated, the files appear in a tree under the “List of files” folder. When the user selects a file, all the displayed information is related to this file except for the query results, which are based on all uploaded files. Once the reasoner is applied on the translated version of a file, both consistency checking and inference are performed. To show the differences between the input and output files, we upload them in Protégé [36] (see Figure 12.) The two screenshots of Protégé highlight the result of an inference mechanism executed on a list structure: an instance (1000_Direction_ratios_0) of List has more properties after the reasoning procedure (those properties are shown in the lower part of the screenshot). For more information about the list structure, please refer to Section 3.3.2.3.

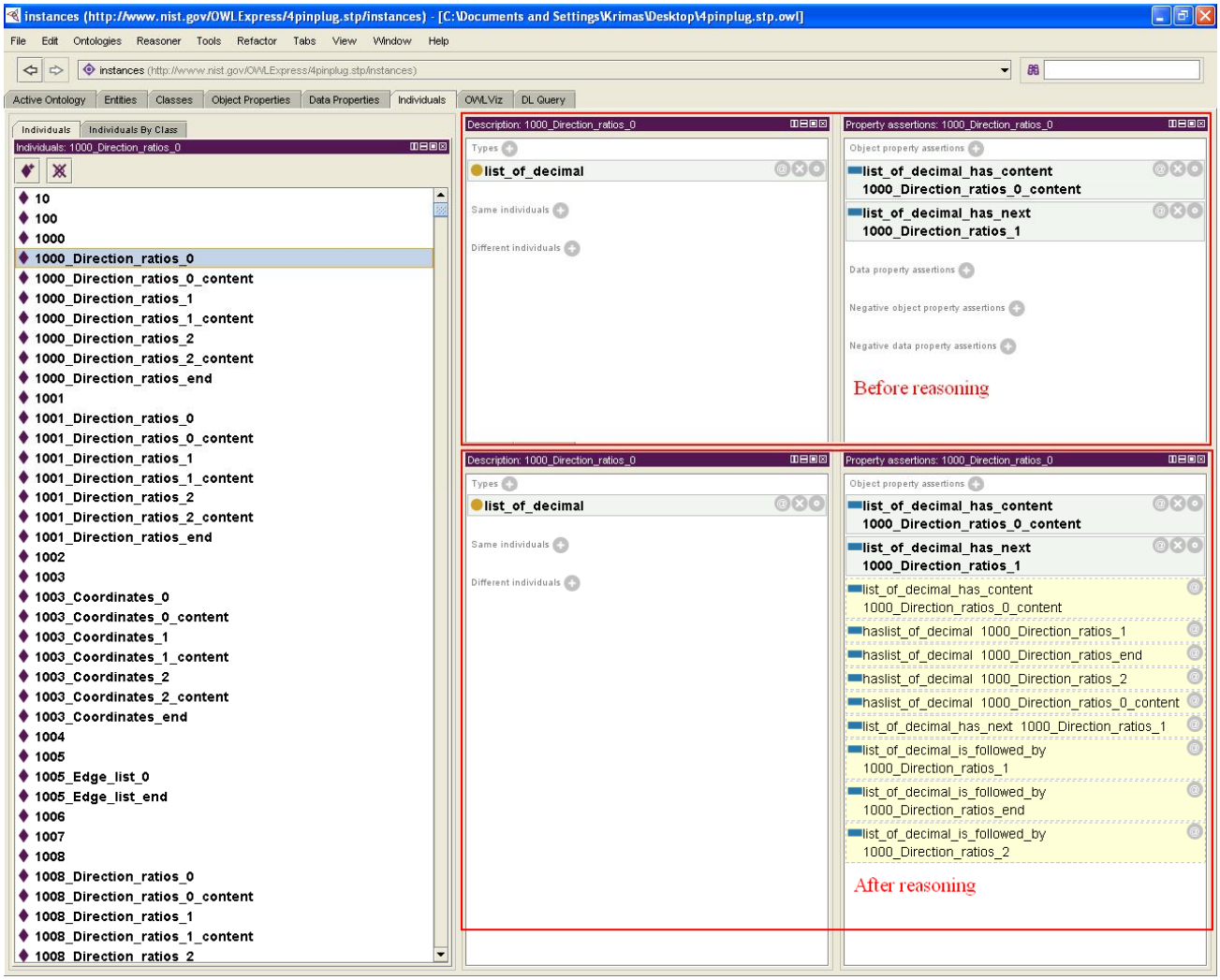

Figure 12: Ontology in Protégé editor: before and after reasoning

3) To visualize the ontology in the web application, a hierarchical tree is displayed in the main tab. This tree shows both instances and classes. Tree nodes are displayed in gray or black: gray is used for classes that do not have either subclasses or instances while black is used for classes that have subclasses. The number between brackets corresponds to the number of instances contained in each class. Clicking on a black class transforms it in a folder that the user can expand and collapse to explore the tree.

4) To obtain more information about the selected file, the user can navigate the second tab (the one named "Visualization", see Figure 11.) At the bottom of this tab the ontology is displayed in two different ways (4A and 4B in Figure 13.) The top of the tab is currently blank: we plan in the future to fill this area with a graph representing the ontology.

   4A) In the first display, a tree represents the assembly decomposition contained in the selected file, where products are represented by their identifiers.

   4B) In the second display, a table is used to display the information related to the configuration management of each component of the assembly. This information includes identifier, name, version, categories, etc.

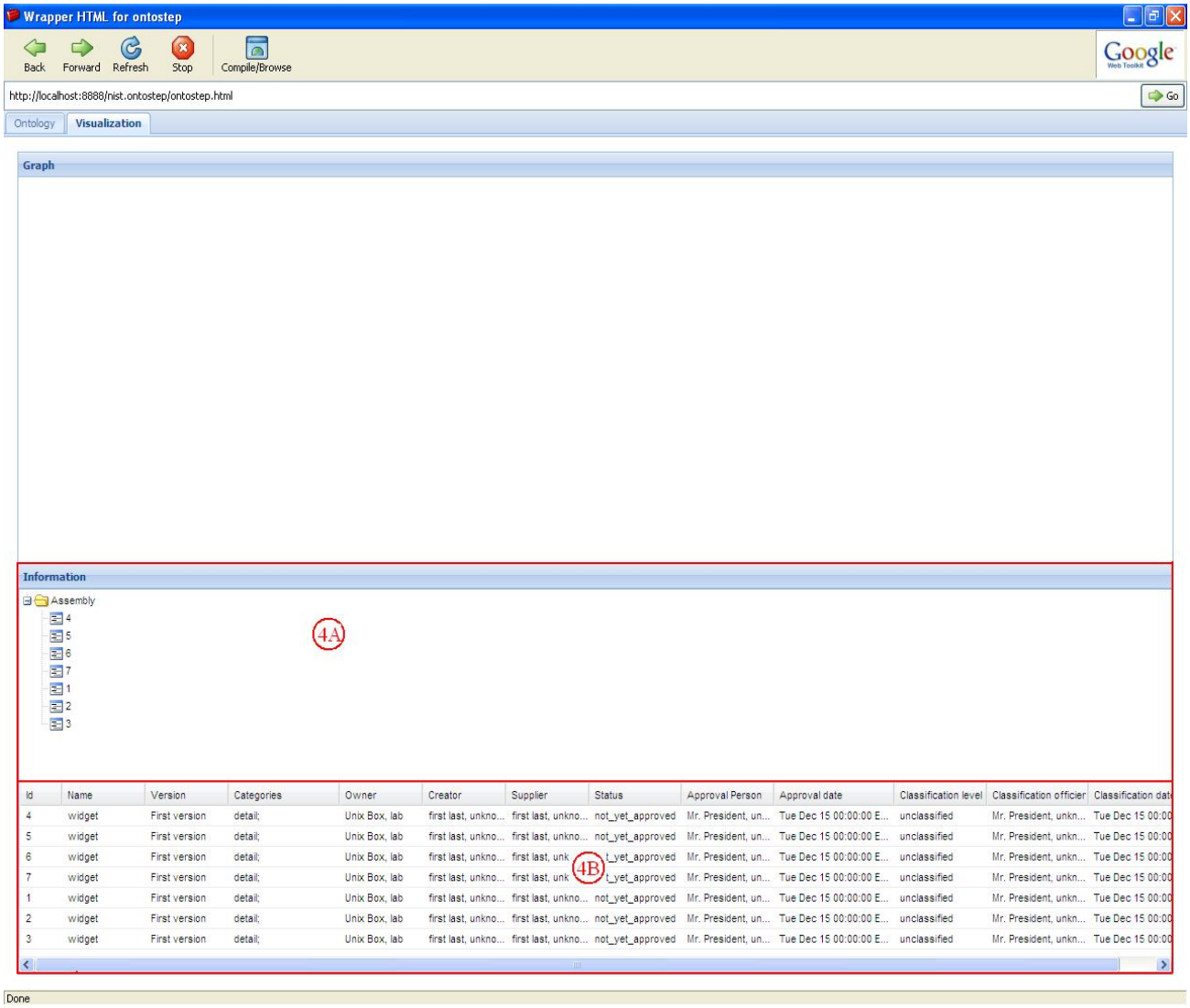

Figure 13: Products and assemblies visualization

5) A query form is provided to allow the user selecting the criteria to compose a query. Criteria are grouped by category and each category is represented by a tab. The "Configuration management" tab is the only one currently available. Other categories, such as Geometry and Assembly structure, will be added later. The current category allows the user to define queries based on 12 product related criteria: Product name, version name, categories, person and organization design owner, person and organization creator, person and organization supplier, approval status, person and organization approval, approval date, classification level, person and organization classification officer, and classification date. To select a criterion, the user needs to check the corresponding checkbox, label the criterion as "used", and input its value in the corresponding textbox. In this simple case, the user wants to retrieve the products the category of which is "detail".

6) The results of the query are displayed in a table. The table used to present these results is organized in the same way as the one in the "Visualization" tab: the rows contain the products that match the query while the columns contain the configuration management information of the products. The results of the query are shown at the center of the screen of the web application. Nine products match the query: one from the test_step.stp file, one from the test_step2.stp file and seven from the 4pinplug.stp file.

Our web application allows users to obtain OWL-DL translations of their CAD files. These versions have an enriched expressivity that allows for checking the consistency of the files and inferring new knowledge (an example is provided in Figure 12). These ontologies present real benefits: the inferred ontologies contain more information than the directly converted one and can be queried. The currently implemented queries provide only an overview of the capabilities of OntoSTEP. We plan in the future to enrich this set of queries and to implement an interactive visualization system to represent the ontology as a graph.

6 Conclusion and future work

Semantic interoperability between the applications that exchange product information is required to achieve systems integration. STEP is the most known and accepted standard for the exchange of product geometry information: its aim is enabling interoperability between engineering applications.

The main benefits of the semantically enriched STEP information presented in this paper are the ability to check the consistency of EXPRESS schemas, the ability to check the validity of the Part 21 files against their schemas and the opportunity of performing queries on those files. In this paper we presented a mapping to OWL-DL from the STEP AP203 and Part 21 CAD files and we showed the principles we followed to create it. These same principles could be used to create OWL mappings of other STEP APs.

We also presented a web application to allow users to upload their CAD files, to translate them, and to manage and query their product ontologies.

In the future, we plan to combine OntoSTEP with the OWL-DL versions of the CPM/OAM, which are information models to support beyond geometry information. We also plan to develop a plug-in to CAD applications to allow the insertion of this information, which would then be checked for consistency along with the geometry information.

We also plan to strengthen OntoSTEP by formalizing at the MetaObject Facility (MOF) [37] level the translation between EXPRESS and OWL. For both these languages, a MOF-compliant metamodel has been developed [38] [39]. A translation between these metamodels would allow a bi-directional robust transformation between EXPRESS and OWL.

Disclamer

No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST nor does it imply the software identified are necessarily the best available for the purpose.

References

1. International Organization for Standardization. ISO 10303-11: Industrial automation systems and integration -- Product data representation and exchange -- Part 1: Overview and fundamental principles, 1994.

2. W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/ . 2004.

3. International Organization for Standardization. ISO 10303-203: Industrial automation systems and integration -- Product data representation and exchange -- Part 203: Application Protocol: Configuration controlled 3D design of mechanical parts and assemblies, 1994.

4. International Organization for Standardization. ISO 10303-21: Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure, 2002.

5. Subrahmanian, E., Rachuri, S., Fenves, S., Foufou, S., and Sriram, R. D., "Product lifecycle management support: A Challenge in supporting product design and manufacturing in a networked economy," *Int.J.Product Lifecycle Management*, Vol. 1, No. 1, pp. 4-25, 2005.

6. International Organization for Standardization. ISO 10303-214: Industrial automation systems and integration -- Product data representation and exchange -- Part 214: Application protocol: Core data for automotive mechanical design processes, 2003.

7. International Organization for Standardization. ISO 10303-239: Industrial automation systems and integration -- Product data representation and exchange -- Part 239: Application protocol: Product life cycle support, 2005.

8. International Organization for Standardization. ISO 10303-11: Industrial automation systems and integration - Product data representation andexchange - Part 11: Description methods: The EXPRESS language reference manual, 1994..

9. Fiorentini, X., Gambino, I., Liang, V., Rachuri, S., Mahesh, M., and Bock, C., "An ontology for assembly representation," National Institute of Standards and Technology, NISTIR 7436, Gaithersburg, MD 20899, USA, 2007.

10. Fenves, S., Foufou, S., Bock, C., Bouillon, N., and Sriram, R. D., "CPM2: A Revised Core Product Model for Representing Design Information ," National Institute of Standards and Technology, NISTIR 7185, Gaithersburg, MD 20899, USA, 2004.

11. Baysal, M. M., Roy, U., Sudarsan, R., Sriram, R. D., and Lyons, K. W., "The Open Assembly Model for the Exchange of assembly and tolerance information: overview and example," Proceedings of the ASME DETC/CIE'04 Conference, 2004.

12. SWRL, W3C Member Submission. http://www.w3.org/Submission/SWRL/ . 2004.

13. Klein, L., Liutkus, G., Nargelas, V., Sileikis, P., Baltramaitis, T., Schowe-von der Brelie, B., Alfter, A., and Wesbuer, C., "Ontologies derived from STEP data models," S-TEN, Deliverable D3.3, 2008.

14. S-Ten SemanticSTEP. http://www.s-ten.net/ . 2009.

15. Zhao, W. and Liu, J. K., "OWL/SWRL representation methodology for EXPRESS-driven product information model," *Computers in Industry*, Vol. 59, pp. 580-600, 2008.

16. Sandia National Laboratories. Jess Engine. http://herzberg.ca.sandia.gov/ . 2008.

17. W3C. Extensible Markup Language (XML). http://www.w3.org/XML/ . 2005.

18. W3C. eXtensible Stylesheet Language. http://www.w3.org/TR/xslt . 1999.

19. W3C. OWL 2 Web Ontology Language: Model-Theoretic Semantics. http://www.w3.org/TR/2008/WD-owl2-semantics-20080411/ . 2008.

20. Fiorentini, X., Rachuri, S., Mahesh, M., Fenves, S., and Sriram, R. D., "Description logic for product information models," Proceedings of the ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, 2008.

21. Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H., and Seidenberg, J., "Putting OWL in order: Patterns for Sequences in OWL," Proceedings of the OWLED '06 OWL: Experiences and Directions, 2006.

22. W3C. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/ . 2008.

23. LLC. Pellet. http://clarkparsia.com/pellet/ . 2008.

24. Sriram, R. D., *Intelligent Systems for Engineering: A Knowledge-Based Approach*, Springer, 1997.

25. W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/ . 2008.

26. SQWRL. http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL . 2009.

27. LLC. Pellet Features. http://clarkparsia.com/pellet/features/ . 2008.

28. SQWRL - Aggregation functions. http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL#nidA20 . 2009.

29. University of Manchester. OWL API. http://owlapi.sourceforge.net/index.html . 2008.

30. Open Source EXPRESS Parser. http://sourceforge.net/projects/osexpress/ . 2001.

31. ANTLR Parser generator. http://www.antlr.org/ . 2008.

32. International Organization for Standardization. ISO 10303-22: 1998. Industrial automation systems and integration -- Product data representation and exchange -- Part 22: Implementation methods: Standard data access interface.

33. STEP Tool, Inc. STEP and STEP-NC Software for e-manufacturing. http://www.steptools.com/ . 2009.

34. Google. Google Web Toolkit. http://code.google.com/webtoolkit/ . 2008.

35. ECMA international. ECMA 262 - ECMAScript Language Specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm . 1999.

36. Standford Center for Biomedical Informatics Research. Protégé-Owl. http://protege.stanford.edu/overview/protege-owl.html . 2009.

37. Object Management Group. Meta Object Facility (MOF) Specification, 2002.

38. Object Management Group. Reference Metamodel for the EXPRESS Information Modeling Language RFC, 2008.

39. Object Management Group. Ontology definition metamodel (ODM), 2008.