OntoSTEP: Enriching product model data using ontologies

Raphael Barbau[1], Sylvere Krima, Xenia Fiorentini, Sudarsan Rachuri, Anantha Narayanan, Sebti Foufou, Ram D Sriram
Manufacturing System Integration Division,
Manufacturing Engineering Laboratory,
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
{raphael.barbau, sylvere.krima, rachuri.sudarsan, anantha.narayanan, ram.sriram}@nist.gov,
xenia.fiorentini@gmail.com, sebti.foufou@u-bourgogne.fr

Abstract: *The representation and management of product lifecycle information is critical to the success of any manufacturing organization. Different modeling languages are adopted to represent different product information, for example EXPRESS for geometry as seen in the Standard for Exchange of Product model data (STEP), and the Unified Modeling Language (UML) for beyond-geometry information as defined in the Core Product Model (CPM). It is necessary to consolidate product information created using these different languages to create a coherent knowledge base. In this paper, we present an approach to enable the translation of STEP schema and its instances to Ontology Web Language (OWL). This gives a semantically rich model, which we call OntoSTEP, that can easily be integrated with OWL ontologies. A plug-in for Protégé is developed to automate the different steps of the translation, and the integration of beyond-geometry information. As additional benefits, reasoning, inference procedures, and queries can be performed on the enriched legacy CAD model. We describe the mapping rules for the translation from EXPRESS to OWL, and illustrate the benefits of OWL translation with an example. We will also describe how these mapping rules can be implemented through meta-model based model transformations.*

1 Introduction

The Product Lifecycle Management (PLM) approach enables organizations to manage their product portfolio from conception to disposal, in an integrated fashion [1]. Manufacturing organizations spend a considerable amount of resources to understand and apply the PLM approach. The PLM concept is gaining acceptance primarily because of the emergence of the networked firm and the networked economy, in contrast to the market- or hierarchy-based organizations that typically use a transactions cost model as the cornerstone for the choice of organizational structure [2]. PLM support entails the modeling, capturing, manipulating, exchanging, and using of information in all product lifecycle decision-making processes, across all application domains. Proper representation and management of product information is the key for a successful implementation of PLM.

To enable the exchange of product data throughout a product's lifecycle, the International Organization for Standardization (ISO) has developed the Standard for Exchange of Product model data (STEP) [3] (ISO 10303), which is still evolving to meet the needs of modern Computer-Aided Design (CAD), Computer-Aided Engineering (CAE), and Product Data Management (PDM) systems. Application Protocols (APs) represent the implementable data

[1] Corresponding Author

specification of STEP. Examples of the most widely used APs are AP203 (Configuration controlled 3D design of mechanical parts and assemblies) [4], AP214 (Core data for automotive mechanical design processes) [5] and AP239 (Product life cycle support)[6]. These APs mainly focus on product data management and geometry information. Unfortunately, the representation of beyond-geometry information such as the function and behavior of the product is usually not part of the information contained in these APs.

The STEP APs are modeled using the EXPRESS (ISO 10303-11) [7] language. EXPRESS was developed for representing product models and providing support to describe the information required for designing, building, and maintaining products [8]. Data models (or schemas) are represented in EXPRESS as a network of concepts (entities) and relationships between concepts (attributes). Entities and attributes are therefore the basic constructs of EXPRESS. STEP Part 21 [9] defines the syntax for representing instance data according to a specific EXPRESS schema.

Many tools have been developed to check the syntax of EXPRESS information models and the validity of the instances against the information models. Unfortunately, these tools are specifically developed for implementers and consumers of STEP, so their usage is restricted to the field of product modeling in EXPRESS. Lack of explicit semantics and contexts in the information content to be shared across PLM applications is a major problem. Making data semantics explicit and context aware and sharable among product lifecycle applications is a major challenge. For an evolving organization to function, an information infrastructure that supports well-defined information exchange among the participants is critical.

The development of a high-level interoperable model poses many challenges, such as, i) the complex nature of interactions in product modeling and ii) representation of the information content and abstraction principles used. The model for representing mechanical assemblies (products) is inherently complex due to:

- the variety of information elements to be represented: function, behavior, structure, geometry and material, assembly features, tolerances, and various levels of interaction of these concepts
- the abstraction principles needed to represent the information model: generalization, grouping, classification, and aggregation.

A recent paper [10] outlines a method for evaluating the appropriate level of expressiveness to capture both the information content and the abstraction principles, discussed with the aim of developing a consistent formal model for product assemblies. The report also evaluates the application of OWL [11] (Ontology Web Language) for the development of ontologies for manufacturing products, in terms of expressivity and the use of SWRL (Semantic Web Rule Language) [12] for representing domain-specific rules.

Based on our previous work [10,11], this paper describes how to translate STEP models from EXPRESS to OWL to semantically enrich the models. This translation allows the application of

logic mechanisms to check the validity of the models and data, to check the consistency of the instances, and to infer new knowledge. These logic mechanisms are performed by software tools called reasoners [10]. We refer to the translated STEP as OntoSTEP in the remainder of the paper. OntoSTEP [13] could be used to express and semantically enrich product information available in STEP files. In this paper, we describe the translation of STEP AP 203 data models and Part 21 CAD files into OWL. The methodology followed is fully applicable to any other STEP AP and corresponding Part 21 file.

OntoSTEP is a step towards developing semantically enriched product models. Semantic models support both the representation of product geometry concepts and beyond-geometry concepts. Figure 1 shows the different steps involved in building a semantic product model. We outline three important steps: 1) a STEP AP (AP203 in this paper) written in EXPRESS is translated into an OWL schema, 2) data instances are extracted from a Part 21 file (exported from a CAD system) and translated into OWL individuals, 3) this ontology is now combined with another product model (CPM-OAM in this paper) to add beyond-geometry information, resulting in a new semantically enriched product model. The semantic product model also allows us to perform reasoning, inferences, and queries, as shown in the figure (this will be described in Section 2.3).

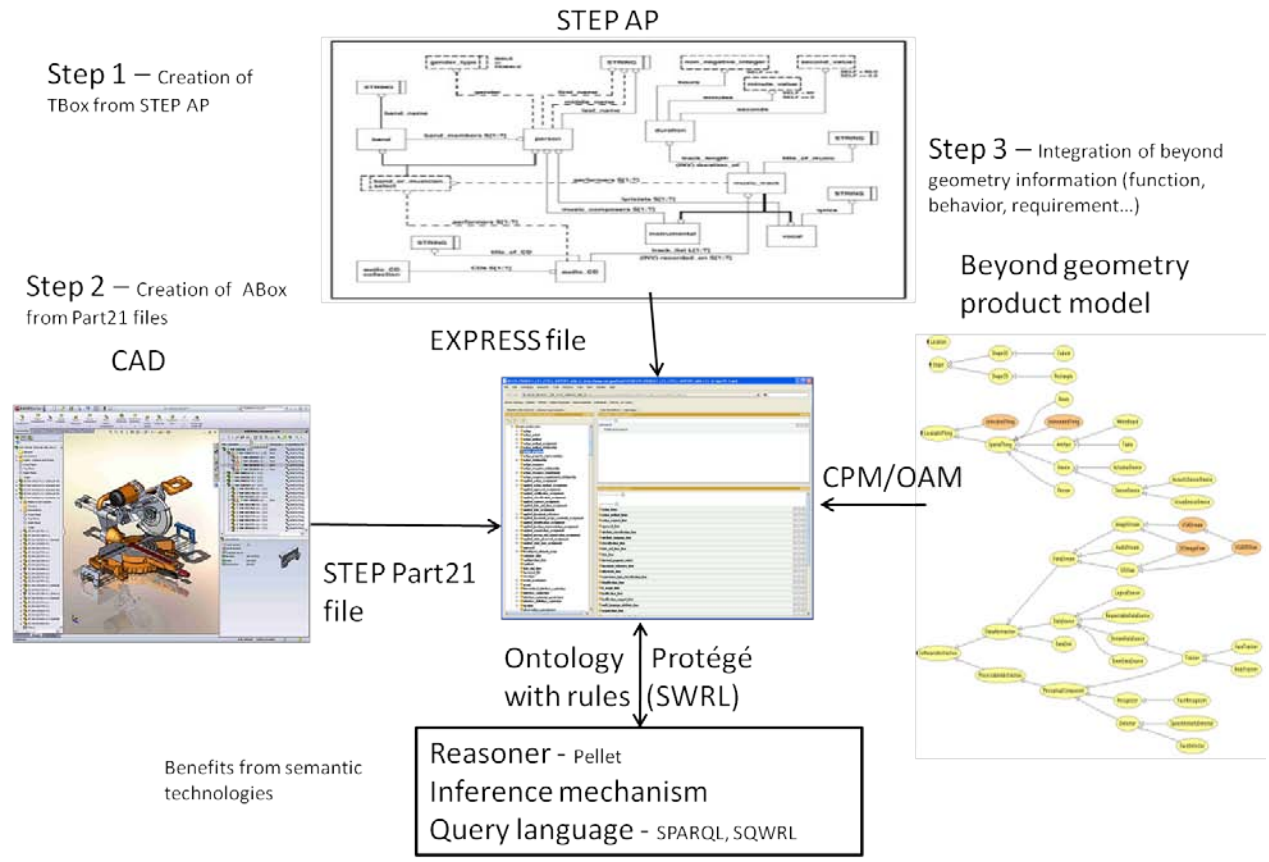


Figure 1: Building semantic product model

In our earlier work [14], we created a semantic model for including beyond-geometry concepts (such as function, behavior, requirements and sustainability factors) from the Core Product Model (CPM) [15] and Open Assembly Model (OAM) [16] developed at NIST (National Institute of Standards and Technology). This model was developed in OWL-DL 1.0 [11] and enriched with SWRL rules [12]. A third party reasoner allows reclassifying the input instances, and the SWRL rules allow refinement and improvement of the model.

In this paper, we combine OntoSTEP with the semantic models of CPM and OAM. A plug-in for Protégé [17] will enable the beyond-geometry information of the designed product to be included in the OntoSTEP representations. The traditional geometry information and beyond-geometry information would then be represented in a unified, consistent OWL model.

The paper is organized as follows. We introduce the OntoSTEP concepts and benefits in Section 2. We present the details of our implementation (plug-in for Protégé) and the tools used to realize it in Section 3. We illustrate the integration of beyond-geometry information with a use case in Section 4. In Section 5, we introduce a high-level model transformation specification for the OntoSTEP translation. Finally, we present our conclusions and future work in Section 6.

2 OntoSTEP: translating STEP data model into OWL

This section describes how a STEP data model (schema) is translated into OWL. The term STEP data refers to the models written in EXPRESS and instances of these models. For example, the data model would be an AP203 schema, while an instance would be a Part 21 CAD file containing the 3D representation of a product. We use `Courier New font` to denote EXPRESS code and we use Arial Narrow font to denote OWL code.

2.1 Schema Mapping (TBox)

In our translation, EXPRESS entities and instances map to OWL classes and individuals respectively. Attributes are mapped to OWL properties. The OWL language defines two kinds of properties, object properties and data properties. Object properties link classes together, while data properties link classes to data types. The domain of a property defines which classes have this property. Without restrictions, OWL properties are aggregations, so an individual can be linked several times to other individuals using the same property. However, it is possible to restrict the cardinality and the range of values of a property. We used the "ObjectExactCardinality" and "ObjectAllValuesFrom" constructs to represent these restrictions. ObjectExactCardinality asserts how many times an object property must be used for a given individual. ObjectAllValuesFrom defines what the range of the property is. In the case of an optional attribute, the "ObjectAllValuesFrom" construct is used to link the entity that has the attribute 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 in EXPRESS, which means that a value is not required for this attribute.

An ontology may contain statements related to both classes (TBox - terminological box) and individuals (ABox – assertion box). In our translation, a schema is translated into an ontology that contains mainly classes and property definitions [10]. The following table summarizes our proposed 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 is 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 owl:Nothing.

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

We also need to redefine the naming conventions for properties. In EXPRESS, attributes are defined to be within the scope of the entity; in OWL, properties have a global scope. We prefix the attribute names with the entity names in order to differentiate attributes that have the same name but belong to different entities.

2.2 Instance Mapping (ABox)

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 complies with the AP203 schema and the constraints of STEP Part 21. In this paper, we refer to these files as "Part 21 files."

The translation to OWL is performed by parsing all the instances declared within the Part 21 file, and then creating individuals and property assertions. 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. In a similar fashion, we generate two different ontologies during the translation: a schema ontology for the EXPRESS schema and an instance ontology for the Part 21 file. OWL provides a mechanism to import statements declared in an external OWL ontology. We use this feature in the instance ontology to import the schema ontology, so that we maintain the schema ontology separate from the instance ontology while accessing both simultaneously. 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.

All the EXPRESS instances contained in a Part 21 file are distinct, which means that any two EXPRESS instances represent two different real world objects. Conversely, OWL individuals are not inherently distinct. Unless explicitly declared as distinct, two OWL individuals may represent the same real world object. To translate the EXPRESS instances correctly, it is then necessary to declare explicitly all the OWL individuals contained in the same file to be distinct using OWL constructs.

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, let us consider two EXPRESS entities called `product` and `product_category`. If an instance of `product` is

not declared as an instance of `product_category`, then the system assumes it is not. This behavior is called the Closed 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 as `optional`, so it must be instantiated for all the instances of product. In EXPRESS, the lack of data will raise an error. In OWL, even if the `id` is not specified for an instance of `product`, the reasoner will 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 a missing `id`, it would be required to declare explicitly that that instance of product has no `id`.

The translation of some additional concepts, such as derived data types, is also required before completing the translations of STEP APs. Our approach for translating these concepts is presented in the next section.

2.2.1 Mapping additional concepts

Some EXPRESS constructs, such as functions, cannot be automatically translated: these constructs usually define entity constraints and attributes computation, and may rely on complex algorithms. 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. More details regarding the translation of some EXPRESS concepts, such as `SELECT`, `ENUMERATION`, and `UNIQUE` can be found in our earlier report [13].

2.2.2 Data types

EXPRESS includes all the data types required to capture product information. OWL inherits the data types defined in the XML Schema Definition (XSD) language. Some EXPRESS types, e.g., `Boolean` and `String`, have an exact equivalent in OWL, while other types, e.g., `Number` and `Real`, are represented in a slightly different way in OWL. For example, we translated the EXPRESS `Real` type as a `double` in OWL, even though the precision of those two data types is different. The translation of the `Logical` and `Binary` data types is outside the scope of this paper since these data types are not contained in AP203.

EXPRESS allows the derivation of data types from simple types. In order to deal with these derived types in OWL, we build a type hierarchy and apply the concept of data wrapping.

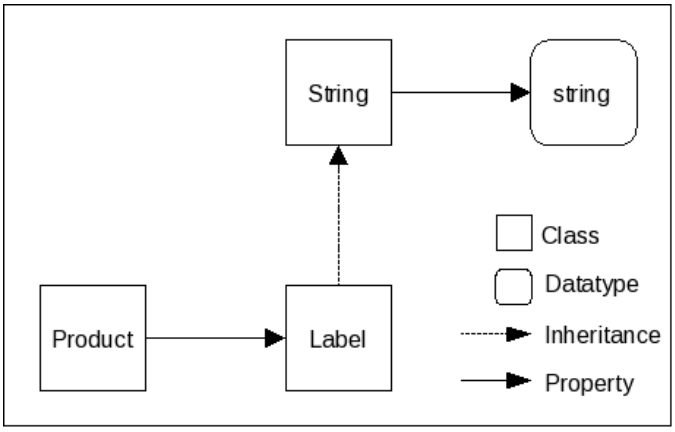


Figure 2: Attributes

In the example in Figure 2, we define a class String that has a DataProperty relation to the string data type. It is then sufficient to subclass the class String to translate all the user-defined data types (Label in this case) derived from string. Because of the possible use of functions, we cannot guarantee an automatic translation of data type restrictions. Using a manual case-by-case translation, most of the types defined in AP203 can be translated.

2.2.3 Aggregations

EXPRESS provides four different kinds of aggregations: `set`, `bag`, `list`, and `array`. Each of these aggregations has order and duplication policies. When an actual aggregation is used in a schema, the type of its content and the number of elements it shall have are defined. The detailed mapping of aggregations is explained in [13]. Here, as an example, we provide the detailed mapping of `bag`.

Bags are unsorted collections of elements. The only difference between sets and bags is the duplication policy: the same element can be repeated several times in a bag. As object properties in OWL do not allow duplications, we create the concept structure shown in Figure 3 to correctly map bags from EXPRESS to OWL. Consider an EXPRESS bag called `Container`, which can contain items of type `Content`. To correctly map this aggregation to OWL, we perform the following steps.

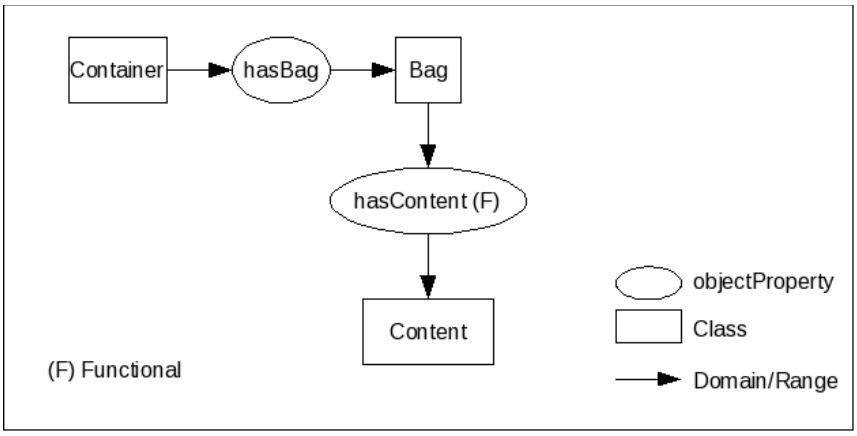


Figure 3: Bag (Class level)

We create a class Container in OWL. A new class, called Bag, is inserted between the Container class and the Content class. The class Bag is used to represent an occurrence of Content using the property hasContent. The property hasContent is declared as functional, which means that there can be only one instance of Content linked to an instance of Bag by the property hasContent. Figure 4 represents the instantiation of the schema presented in Figure 3, illustrating how an EXPRESS bag containing a duplicated element is converted to OWL. An instance of Container (cont) is linked to two different instances of the Bag class (b1 and b2). Each of these two instances is then linked to the same instance of the Content class (elem1). Since b1 and b2 are different, the ontology contains the fact that the elem1 element is present twice in the aggregation.

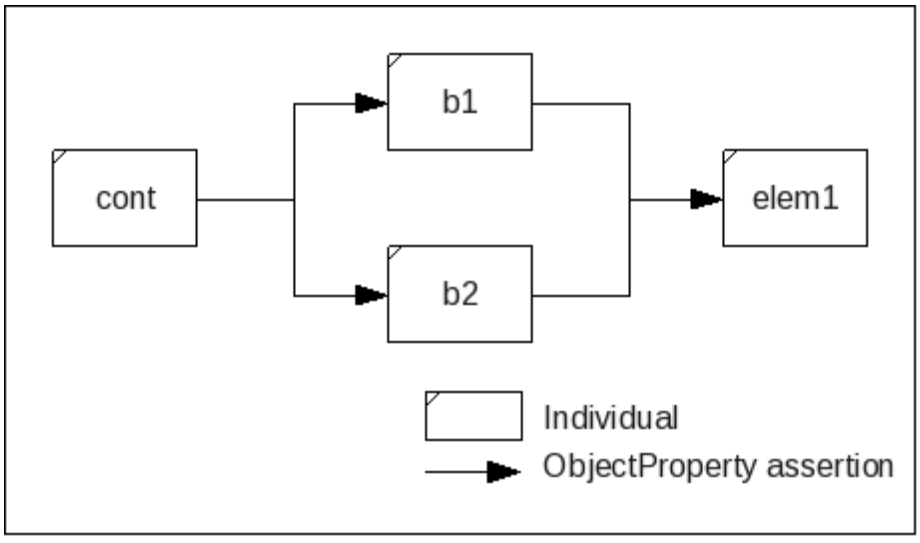


Figure 4: Bag (Individual level)

2.2.4 Abstract Entity and Class

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

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

In this example extracted from AP203, the entity `document_reference` is declared as abstract, and the entity `cc_design_specification_reference` is defined as its subtype. This means that the entity `document_reference` cannot be directly instantiated, but `cc_design_specification_reference` can be instantiated.

OWL does not provide any feature to translate the `ABSTRACT` keyword, i.e. an OWL class cannot be declared as abstract. Using an OWL class to represent an abstract entity causes a problem: because of the Open World Assumption, we cannot assume that the OWL class will never be instantiated. To overcome this problem, we can declare the subtype classes as partitions 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 equivalent to the set of instances of all its subtypes. In that case, if an individual is 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 works only when the supertype and all the subtypes are declared within the same schema. Because of these reasons, we choose to ignore the `ABSTRACT` keyword.

2.2.5 Inheritance

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.

`ONEOF`: The `ONEOF` keyword takes a list of entities as its parameter, and it specifies that only one of these entities can be instantiated. An equivalent behavior in OWL is obtained by defining the subclasses as disjoint: an inconsistency is detected when an individual is an instance of two of these subclasses. We mark the set of classes contained in a `ONEOF` list as disjoint. Another

solution could be to use the logical definition of XOR. We could also use the OWL intersection, union and complement operations, to translate AND, OR, and NOT. However, this increases the complexity of the ontology, as the length of the formula increases dramatically with the number of elements involved. For this reason, we choose the first solution.

ANDOR: When no specific constraints are defined, the default keyword for the instantiation is ANDOR. This means that the instance can belong to more than one subclass. In OWL a set of entities joined by an ANDOR is translated by the union of the corresponding classes in OWL. We first represent the union of the subclasses by using the ObjectUnionOf construct and then declare this union to be equivalent to the parent class.

AND: The AND operator imposes that the object be an instance of all the subclasses. In order to represent this constraint in OWL, we use the ObjectIntersectionOf construct to link the subclasses.

2.3 Benefits of OWL ontology for STEP

OWL 2 RL (Rule Language)[18] is based on Description Logics (DL), a family of knowledge representation languages. These languages can be used to define domain concepts according to a predefined and well-understood formalism. Concepts are used to represent the domain's objects, while roles are used to represent relationships between these concepts. The OWL representation provides benefits subject to the level of OWL expressivity. The expressivity of OWL is denoted using different characters. For example OWL 2 RL is SROIQ(D). The explanation of this expressivity is given in [10]. In [10] the authors show examples of DL semantic axioms in product modeling and highlight that DL semantics can be implemented in a reasoner engine to:

- Check its consistency.
- Perform inference on the class hierarchy.
- Perform inference on the membership of the individuals to the classes.
- Query and search the model.

2.3.1 Consistency checking

The consistency checking procedure can be applied at two different levels, namely the schema level and the instance level. At the schema level, the consistency checking will determine whether an instantiation of a concept would create an inconsistency in the model. At the instance level, the consistency checking procedure checks whether an individual declared as an instance satisfies the definition of the class it is instantiated.

Currently, libraries are available to check the consistency of EXPRESS schemas and Part 21 files. With OntoSTEP, a DL reasoner performs both kinds of consistency checking. Checking the logical consistency of the OWL classes and individuals resulting from a translation is a necessary condition to use an inference procedure. As we will see later in Section 4, the integration of

beyond-geometry information allows us to perform advanced consistency checking that is not available in EXPRESS.

2.3.2 Inference procedure

An inference procedure is a mechanism not available to the EXPRESS community. An inference procedure uses the data evidence in a context and draws conclusions using certain problem solving strategies [19]. An inference procedure is the process of reaching these conclusions, and is performed by a reasoner. Reasoners use a knowledge base as a source of data, such as concepts, roles, and axioms, to reach a conclusion. The expressivity of the axioms and concept definitions is dependent on the logic language used.

Once the reasoner has applied all the inference procedures on our ontology, new knowledge and data can be made available. This new data and knowledge can be both at the schema and at the instance level. These dynamic modifications cannot be done in EXPRESS. One can then use a querying mechanism to query the new data, which represents an enriched version of the original ontology.

2.3.3 Queries

Queries are performed to retrieve specific data from a large amount of information. This mechanism does not readily exist with STEP, though some mechanisms have been developed [20]. In our case, we perform queries to retrieve some specific product information from an OntoSTEP file. The information contained in a CAD file is first translated into OWL representation, then checked for consistency, inferenced upon, and finally queried.

There are two approaches in practice today to perform queries on OWL ontologies. The first approach uses a language called SPARQL Query Language (SPARQL) [21], and the second approach uses the Semantic Query-Enhanced Web Rule Language (SQWRL) [22]. OntoSTEP gives users the freedom to choose any query language.

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. SPARQL has two major drawbacks. First, the translation from OWL to RDF increases the computational time. Second, the Pellet [23] reasoner does not support several SPARQL built-in functions, such as DESCRIBE, OPTIONAL, or FILTER. While SPARQL was developed for RDF, SQWRL was specifically developed for OWL. Unlike SPARQL, SQWRL is based on SWRL [12] and does not need any RDF bridge. SQWRL provides not only many built-in functions, but also some classical aggregation functions like maximum, minimum, sum, or average [24], which are missing in SPARQL. While being more appropriate to OWL, we found two issues with SQWRL. First, it is based on the proprietary engine Jess [25], which is the only option currently available to process SQWRL queries. Second, because it is based on SWRL, SQWRL does not allow combining functions together, e.g., it is not possible to query the maximum of averages.

3 Implementation of OntoSTEP

The previous section discussed the transformation rules and mappings necessary to generate an ontology from EXPRESS schemas and instance files. We now present an implementation of these rules and mappings. The main goal here is to create tools to generate ontologies from STEP data. These tools translate both schema files and instances files (Part 21).

We use the Protégé [17] editor to implement OntoSTEP. Protégé is a free, open source ontology editor and knowledge base framework. It is one of the most widely used tools to edit and manage knowledge bases. The Protégé architecture allows third party developers to write their own extensions in Java.

The implementation involves the following steps: generation of the OWL Schema from the EXPRESS Schema (schema translation for creating TBox), generation of the OWL individuals from the Part 21 file (schema instantiation for creating ABox), and development of a plug-in to integrate the TBox and ABox within the Protégé environment. We will also combine OntoSTEP with CPM-OAM (Step 3 in Figure 5) to create an ontology representing beyond-geometry information.

3.1 Schema translation for creating TBox

Translation of an EXPRESS schema file is carried out in two stages. First, we retrieve the syntax tree of the schema. Second, we translate this information by applying the rules described in Section 3. The following paragraphs explain these two stages.

In order to obtain the syntax tree of the EXPRESS schema, we use an open source EXPRESS Parser [26]. This parser is implemented using the ANTLR (ANother Tool for Language Recognition) [27] 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.

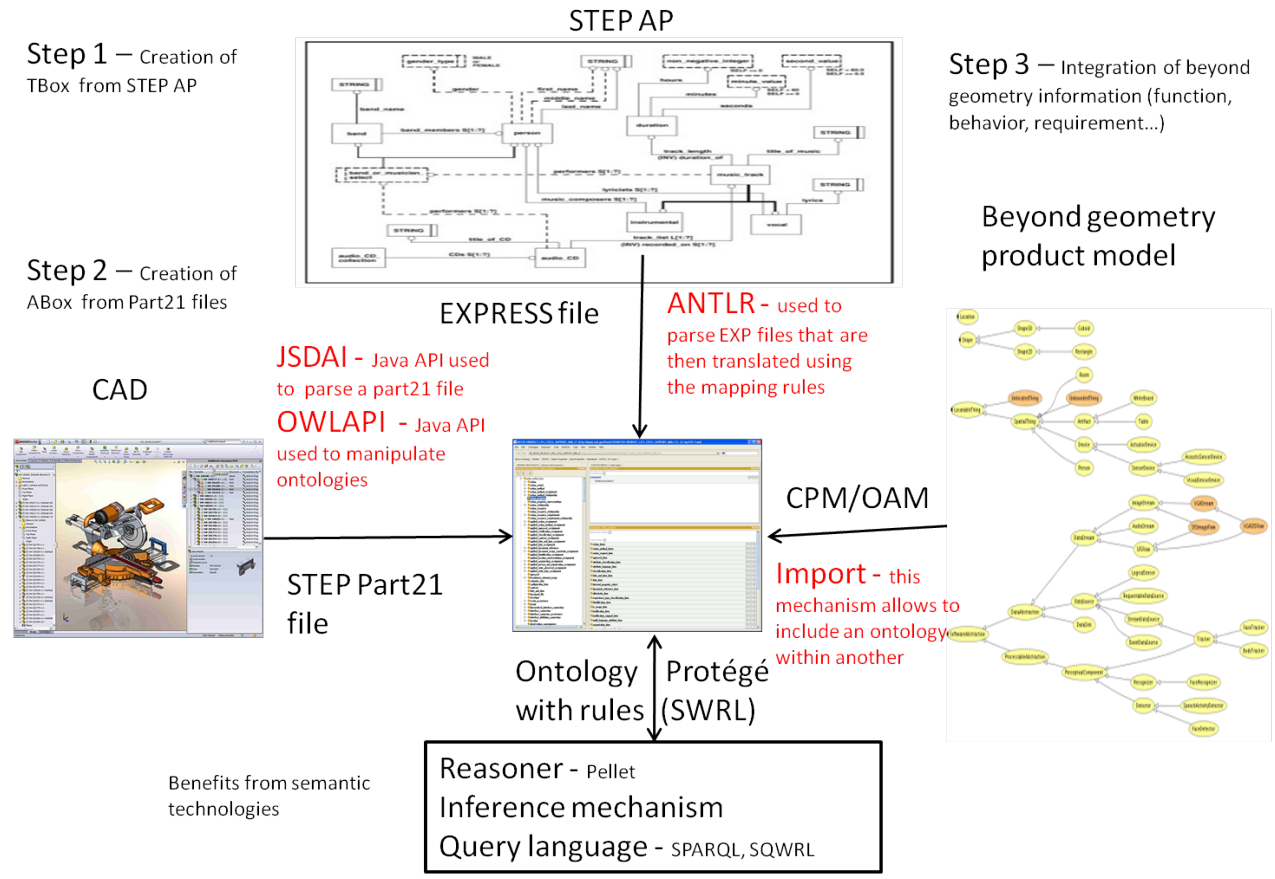
We used the open source OWL Application Programming Interface (OWL API) [28] for OWL2 to create the ontology. This API is used by Protégé 4 [17].

3.2 Schema instantiation for creating ABox

To map the EXPRESS instances contained in a Part 21 file to OWL individuals, we first retrieve the EXPRESS instance information from the Part 21 file. We then process this information to translate them into OWL individuals.

In STEP there is a mechanism called Standard Data Access Interface (SDAI) [29] to manage data defined in EXPRESS schemas. Bindings to several popular languages (C, C++, and Java) are specified by SDAI. We used the open source SDAI implementation from Java SDAI (JSDAI) [30] to translate EXPRESS instances and attributes into OWL individuals and their properties (ABox). For each instance, an individual is created, and its attributes are obtained and translated.

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



CAD - Computer Aided Design **CPM** - Core Product Model **OAM** - Open Assembly Model **ANTLR** - ANother Tool for Language Recognition **SWRL** - Semantic Web Rule Language **SQWRL** - Semantic Query-Enhanced Web Rule Language **SPARQL** - SPARQL Protocol And RDF Query Language **OWL** - Web Ontology Language

Figure 5. Implementation of OntoSTEP

3.3 Protégé plug-in of OntoSTEP

The third and last step of this implementation is to integrate the schema translation and the Part 21 translation within the Protégé editor. This integration is done via the plug-in platform

provided by the editor. This platform allows developers to extend the capabilities of Protégé by developing their own plug-ins. The OntoSTEP plug-in we developed provides two new menu options in Protégé. One option allows translation of a schema, and the other option allows instantiation of a schema. The resulting ontology is displayed through Protégé and users are then free to manipulate the ontology. This implementation of OntoSTEP is summarized in the Figure 5.

4 Extending STEP to include beyond-geometry information using OntoSTEP

OntoSTEP can be extended, as OWL schema and individuals can be enriched by the inclusion of other extensions containing beyond-geometry information. In [14] the authors explain the translation of CPM from UML to OWL. Extensions written in other languages such as UML can be translated in OWL, but it is out of the scope of this paper. We use *this font* to denote CPM terms.

4.1 Motivation for including beyond-geometry information

NIST developed a Core Product Model (CPM) [15] that provides an open, generic, and expandable product model. CPM aims to capture all product information shared throughout the product lifecycle. The key concepts of CPM are *Artifact* and *Feature*. *Artifact* represents a product. A sub-artifact relationship allows product decomposition. *Feature* is defined as a portion of an artifact's form that has specific functions. Different characteristics of the *Artifact* are also described in CPM. *Function* describes what the artifact is supposed to do. *Form* describes a design solution for the problem specified by a *Function*. A *Form* in CPM is composed of *Geometry* and *Material*. *Behavior* describes how an *Artifact* implements a *Function*. CPM also defines different kinds of relationships, such as *Constraint* or *Usage*, to link all the previous concepts.

Because CPM is a generic product model, an extension called the Open Assembly Model (OAM) [31] was created in order to include assembly representation. Some generic concepts defined in CPM are specialized into assembly-specific concepts. For instance, *Artifact* is specialized into *Assembly* and *Part*. Particular focus is given to the connections between artifacts (*MoveableConnection*, *FixedConnection*, *IntermittentConnection, etc*.). These connections can also describe parametric assembly constraints, and can include tolerance information.

We use both CPM/OAM and STEP AP203 to capture product design information. CPM and AP203 share several notions, for example the class *Artifact* and the entity `Product` refer to the same concept. Moreover, both models can specify relationships between *Artifacts*. The goal of the integration is to make sure that information representing the same knowledge is actually recognized as such.

In order to make the integration possible, the data has to be expressed in the same language. To get an integrated representation in OWL, we used the translation tools from EXPRESS to OWL previously introduced, and the OWL version of CPM/OAM explained in [14].

4.2 An OntoSTEP example

In this section, we show how to combine AP203 and CPM through a clock design. AP203 is used to create a 3D CAD model of a product, while CPM and OAM are used to represent the functional decomposition of this product and the relationships between the parts.

4.2.1 Functional Decomposition

The first step is to perform a functional decomposition of the clock assembly. The different parts of the assembly are identified, and their functions are provided. The result of this design phase is an assembly composed of six subassemblies. In CPM, one *Artifact* is created for the assembly and for each subassembly, and the relationship *subArtifacts* is used to relate them. Then for each *Artifact*, a *function* is created and linked to it by the relationship *functionOfArtifact*. Table 2 presents these subassemblies and their corresponding functions.

Subassembly	**Function**
Speed-reduction	Change the angular speed
Escapement	Transmit motion
Weight	Provide energy
Pendulum	Create motion
Face	Show the reading
Frame	Support

Table 2: Functional decomposition of the clock

4.2.2 Geometric definition of the clock

The next step is the design of the product using a CAD system. The assemblies were created using Pro/ENGINEER [32], and then saved in Part 21 files conformant to AP203. The parts are set to have the same name in the CAD file and in the CPM model, so that they can be easily mapped. Figure 6 shows the Speed-reduction subassembly.

Figure 6: 3D representation of the Speed-reduction subassembly

4.2.3 Data Integration of geometry and beyond-geometry information

The previous steps allowed the designer to create two different assemblies using two different data models: the AP203 and the OAM models. The benefit of using semantic technologies is the capability to bring in external knowledge to create a richer model. For instance, in this case we can assert that the clock that has a geometric representation in AP203 model is the same as the clock that has a function in CPM. Then, all the knowledge about the product in each of the two models will be shared.

In order to integrate the geometric data, we follow the steps defined in the previous section. We first have to generate an OWL representation of AP203, which is the EXPRESS schema used by the CAD file to represent the product. This is achieved by translating the EXPRESS schema using the OntoSTEP plug-in in Protégé. The translation of the schema is saved in a file, so that it can be reused by all the relevant Part 21 translations. The second step is to use the OntoSTEP plug-in to convert our CAD file into OWL. This translation requires the schema to be already translated, so the ontology previously created is selected and imported by the plug-in. Once the translation is completed, all the information described in the CAD file is available in Protégé.

Next, it is necessary to identify what concepts are equivalent in CPM and in AP203. The main equivalence is between the class *Artifact* in CPM and the entity `Product` in the AP203. Our goal will be to match the individuals of these two concepts in order to get a single view of the product. We consider three levels of complexity regarding the way the integration should be performed. The easiest is an algorithm that is able to "guess" which parts are equivalent in the two models from the information available. It could, for instance, compare the assembly tree in the two models. The hardest approach is performing the integration manually because there would be no way to "guess. " This would make the designer assert the equivalence in CPM each time a part is created in CAD. A simple tradeoff is to have equivalent parts have the same name in both models, in which case a simple rule that makes this equivalence based on the name is enough. We used the Semantic Web Rule Language *(*SWRL) to make this rule work:

```
cpm:Artifact(?art) ^ cpm:hasName(?art, ?artName) ^ ap203:product(?prod) ^
ap203:product_has_name(?prod, ?prodName) ^ ap203:toString(?prodName,
?prodNameVal) ^ swrlb:equal(?prodNameVal, ?artName) => sameAs(?art, ?prod)
```

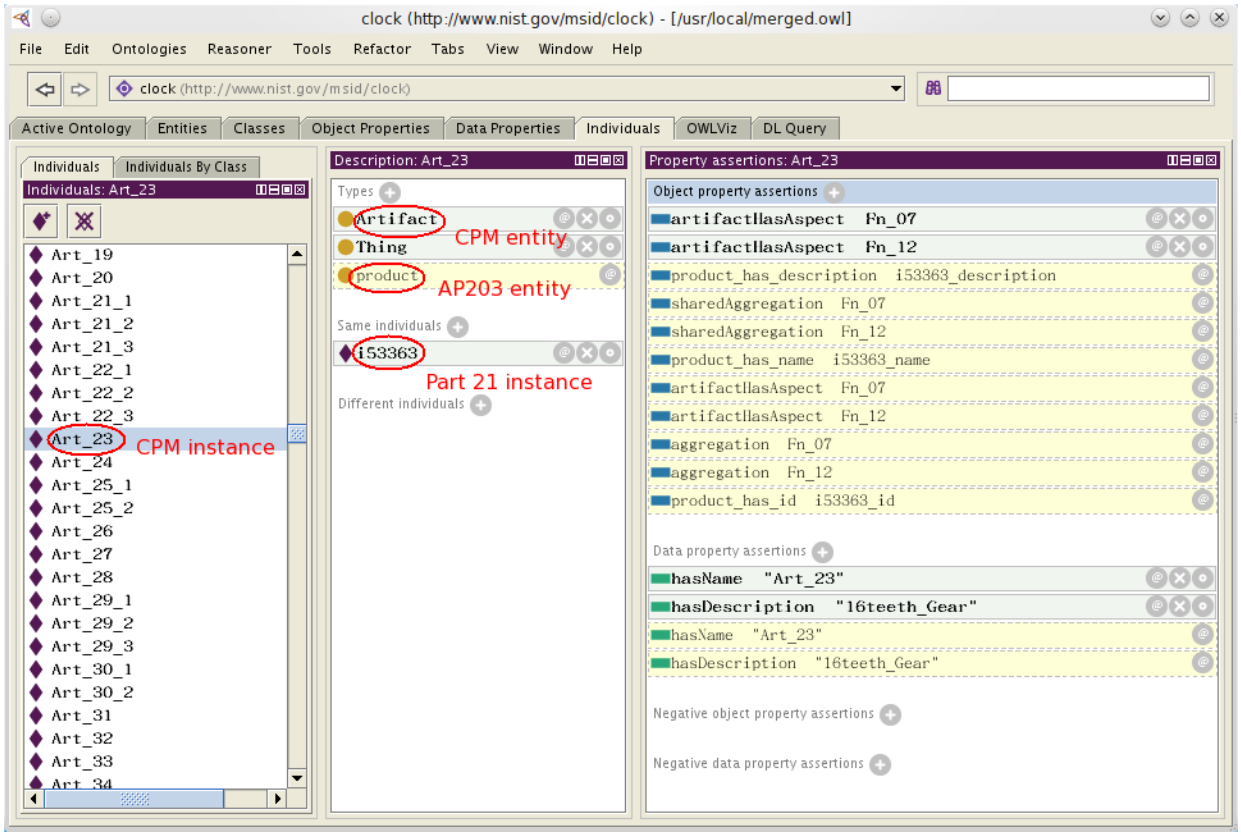


Figure 7: 16-teeth gear in Protégé

4.3 OWL benefits

The benefits described in the Section 2.3 are especially useful in this example. The consistency checking ensures that each class, whether it is the translation of an entity or the translation of a type, is actually instantiable (they can be used). In addition, the Part 21 translation is checked in order to determine if the instances are used as intended.

Once the ontology is verified, the inference mechanisms use logic to determine the types of each translated individual. SWRL rules are also executed by the reasoner. A single integrated view of the product is then generated, as the ontology combines geometry and beyond-geometry information. Figure 7 shows the result in Protégé for the 16-teeth gear (named “Art_23”), which is part of the Speed-reduction subassembly. The figure was annotated to indicate the different sources for the individuals (Part 21 and CPM instance) and for their classes (AP203 and CPM).

The window in this figure is vertically split into three portions. The portion on the left contains the list of all the OWL individuals in the ontology. Here the 16-teeth gear is selected, so the individual named “Art_23” is highlighted. The part in the middle shows all the classes the selected individual is instance of. The statements written as bold text have been explicitly asserted while the statements written as normal text have been inferred. In this example, the individual Art_23 has been explicitly declared as an *Artifact*. As Art_23 is also the same individual as `product` i53363, the type `product` is inferred for Art_23. The portion of the window on the right contains all the object property assertions and all the data property assertions for the selected individual. As an example of object property assertion, the gear is linked to the function Fn_07 (Transmit_motion) by the CPM object property *artifactHasAspect*. The gear is also linked to an identifier `i53363_id` by the AP203 object property `product_has_id`. In regards to data property assertions, the gear has a property *hasName* that contains the value “Art_23”.

It is now possible to take advantage of this integrated system to query the ontology. We present two queries that use both CPM and STEP information to answer simple questions a designer may ask.

The first query is about retrieving the function of a part defined in a CAD system. The input of this query is the identifier of the part defined in the CAD system. The expected output is a string representing the function of that particular part. Here is the SQWRL query that retrieves the function of the product i53363:

```
cpm:Artifact(i53363) ^ cpm:Function(?func) ^ cpm:artifactHasAspect(i53363,
?func) ^ cpm:hasName(?func, ?name) => sqwrl:select(?name)
```

Because of the integration previously performed, the `product i53363` is recognized as being the *Artifact* Art_23. Hence `i53363` gets the properties of Art_23, including the functions Fn_07 (Transmit_motion) and Fn_12 (Change_angular_speed) (see Figure 7). Thus, we are able to infer that the 16-teeth gear performs the functions of transmitting motion and changing angular speed, which would not have been possible with the CAD model alone.

The second query is about retrieving the parts that are connected to a particular part via a fixed connection. The CPM class *FixedConnection* is a kind of *ArtifactAssociation* that is used to state that two parts are physically connected and have a fixed joint. The relationship *artifactAssociation2Artifact* links the connection with the two involved parts. As in the previous query, the input will be a particular part defined in a CAD system: the 16-teeth gear. Here is the SQWRL query that gives the intended result:

```
cpm:Artifact(i53363) ^ cpm:FixedConnection(?fc) ^
cpm:artifactAssociation2Artifact(?fc, i53363) ^
cpm:artifactAssociation2Artifact(?fc, ?art)=> sqwrl:select(?art)
```

This query returns the individuals representing a shaft in CPM and in the CAD system. Since the class *FixedConnection* is defined only in CPM, this knowledge cannot be represented in CAD alone.

5 Meta-model Based Model Transformations

So far, we have described our mapping rules from EXPRESS to OWL, and its implementation in Java. The translation scheme shown in Table 1 is a high-level description that is easy to understand. However, the implementation requires us to deal with low-level details such as language grammar and programming libraries. A more straightforward method to implement the translation is using high-level model transformations, which will allow domain experts to concentrate on the high level mapping rules, and not worry about low-level language details.

A model transformation is a sequence of well-defined rules that takes an input model (such as an EXPRESS model) and produces an output model (such as an OWL model). While these transformations are usually conceived as high-level mappings as described here, they are often implemented using an imperative programming language such as Java. However, recent developments in model transformation allow us to specify these mappings using more formal, high level, visual representations. Using these technologies, the transformations themselves can be specified as models. Specifying the transformation itself as a model allows us to better understand, analyze, and archive the transformation. This is particularly significant when considering long term retention of these specifications.

A meta-model in systems engineering is a model that captures the salient concepts and relations of a domain, along with the rules that define the domain. Meta-model based transformations provide a high-level specification of a mapping between two domains (called *source* and *target* domains), such as from EXPRESS to OWL, that is easier to understand and analyze. The high-level specifications are also easier to evolve when changes are made to the source and target domains.

Section 2 described the mapping rules we used to translate STEP models into OWL. In this section, we create a model transformation from these mapping rules. As an example, we created meta-models for EXPRESS and OWL (derived from a subset of the previously published Meta-Object Facility (MOF) meta-models for these languages in [33] and [34]). We specified model transformation rules based on our previous mapping descriptions, and used an automatic model transformation tool to execute the transformation. This example is described next.

5.1 Example using GME/GReAT

As a prototype study, we have used the Generic Modeling Environment (GME) [35], a tool for domain modeling developed at Vanderbilt University, to create meta-models for subsets of EXPRESS and OWL. The meta-models are defined using stereotyped Unified Modeling Language (UML) class diagrams, as shown in Figure 8 and Figure 9. The notation used in these

meta-models is in the native meta-language of the GME tool, which is derived from UML class diagrams. More details on the meta-language can be found in [36]. The meta-models for EXPRESS and OWL used in this example were created by extracting a small subset of the MOF meta-models that were defined in [33] and [34]. We use the Arial Unicode MS font to denote concepts defined in GMEFor example, Figure 8 shows a subset of the EXPRESS meta-model, which defines concepts called EntityType and Attribute. Attribute has a relation called attributetype, which connects it to *ParameterType* (*ParameterType* is shown in italics to denote that it is an abstract type) . Connections are modeled as UML association classes, and the rolenames ('srcattributetype' and 'dstattributetype') on the connections are used by the GME toolset to keep track of the source and destination objects in the instance models. The *ParameterType* has three subtypes namely SimpleType, DefinedType, and EntityType. Figure 9 shows a subset of the OWL meta-model. It defines concepts such as OWLClass and Property. OWLRestriction is a subtype of OWLClass. OWLClass and Property have a relation called DomainForProperty. The rolenames 'srcDomainForProperty' and 'dstDomainForProperty' for this connection are used by GME to keep track of the source and target objects of this connection.

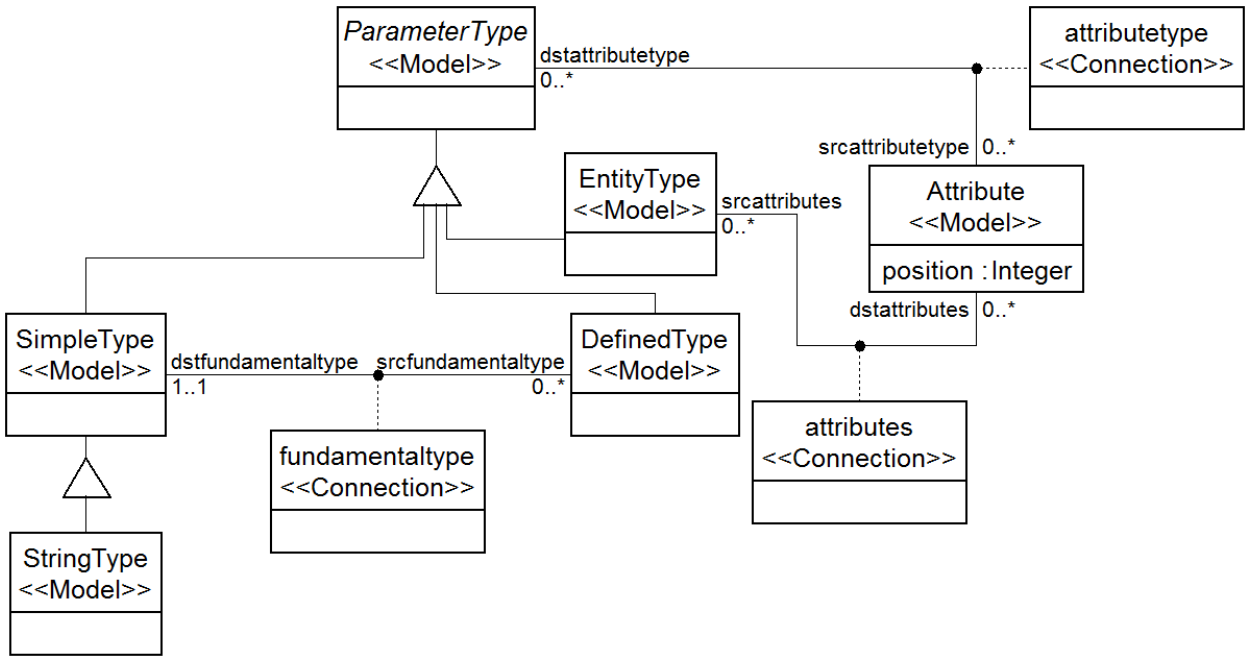


Figure 8: Representative Subset of Express Meta-model

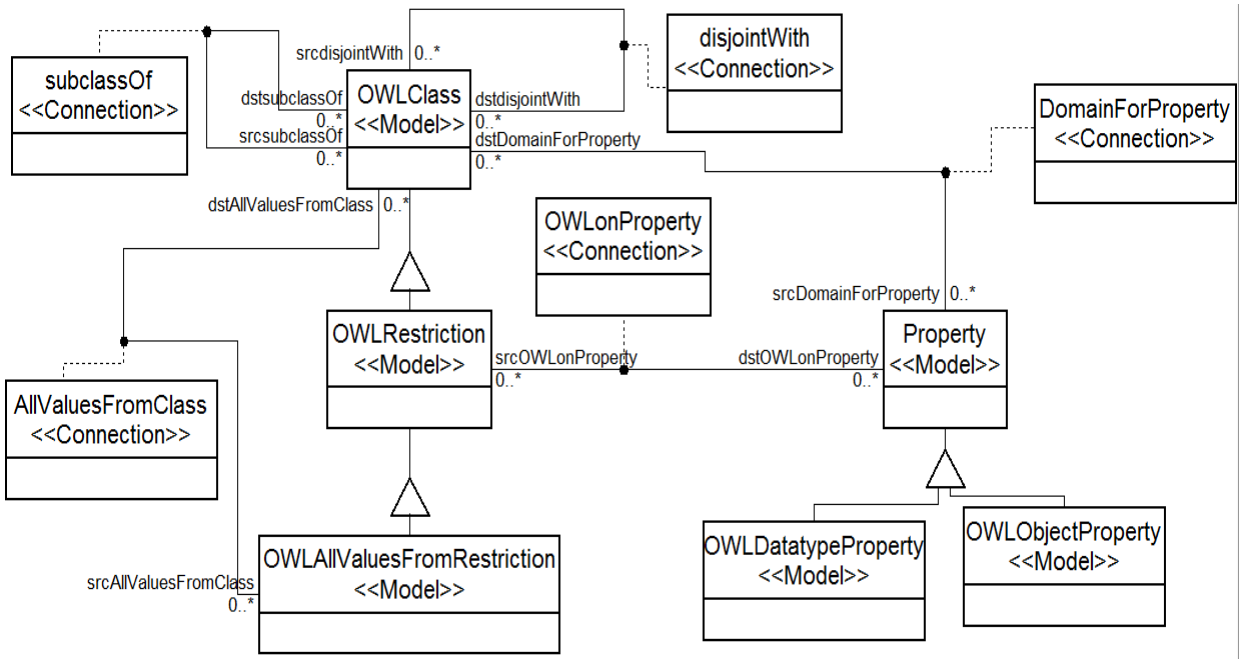


Figure 9: Representative Subset of OWL Meta-model

Next, we specified the mapping rules described in this paper as high-level model transformation rules. We used the Graph Rewriting And Transformation (GReAT) tool suite [37] to specify and execute the model transformation. GReAT allows us to specify the transformation rules as graph transformations, by treating the concepts and relations in the input and output models as nodes and edges of a graph. The rules specify portions of the input that must be *matched*, and portions of the output that must be *newly created*. Figure 10 shows the transformation specified in GReAT, with two of the rules expanded for detail. The first rule, *CreateOWLClass*, specifies the creation of an OWL class corresponding to an entity in the EXPRESS schema. This rule denotes that instances of EntityType must be matched in the input EXPRESS schema, and for every matched entity, an OWLClass must be created in the output OWL schema. The rule *AssignDomains* assigns the domain for a previously created OWL property in the output model, corresponding to the attribute type in the input model.

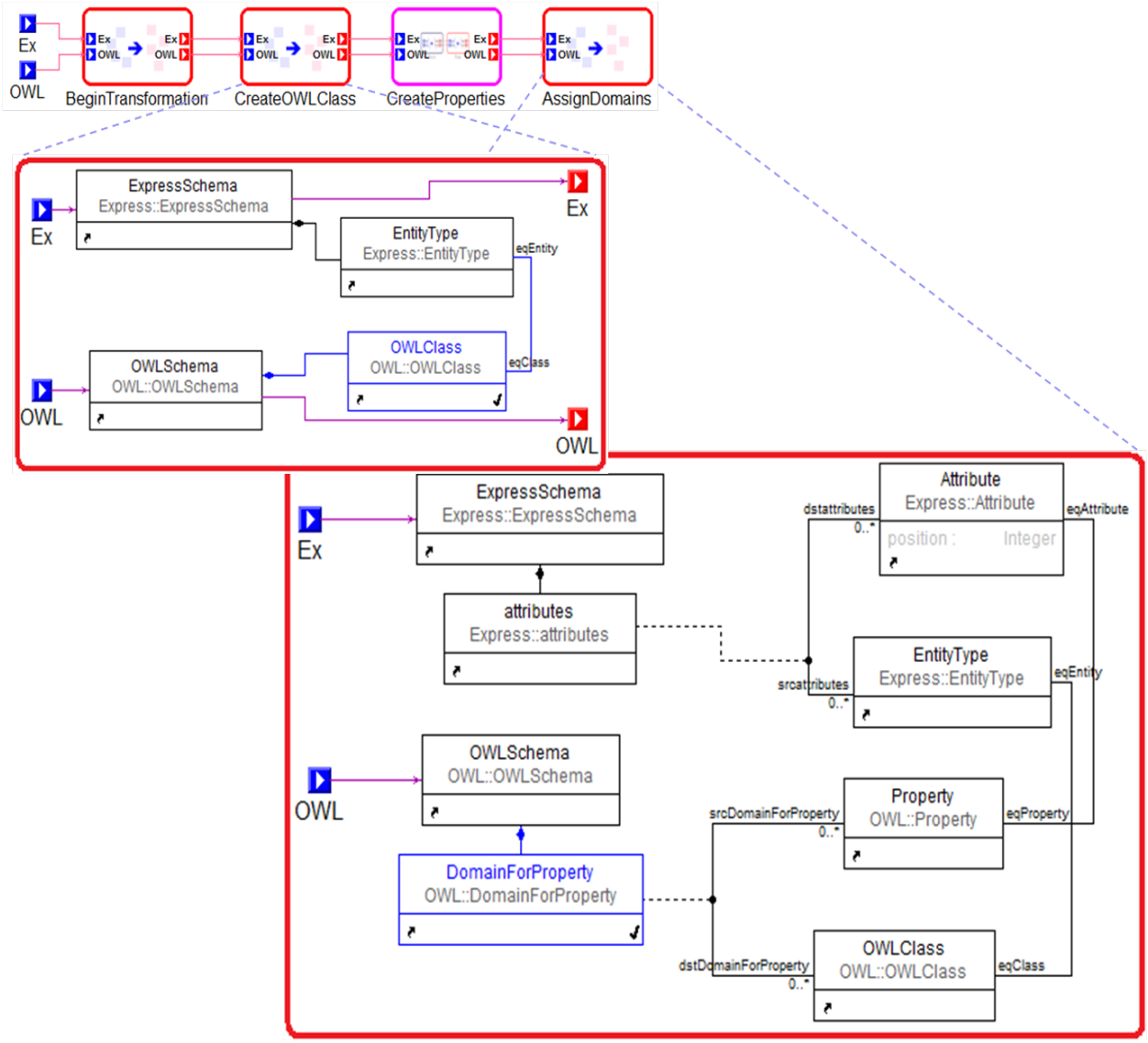


Figure 10: Model transformation from EXPRESS to OWL

The GReAT tool suite allows us to organize the transformation rules both sequentially and in parallel, and encapsulate them hierarchically, making it convenient to analyze and maintain them. Each of the elements in the transformation rule must come from one of the two meta-models. The patterns specified in the transformation rules cannot violate the meta-models. This ensures that the output model is conformant with the OWL meta-model, reducing the possibility of producing an erroneous output model. Other verification methods and technologies are available to perform further analysis, to ensure that the transformation produced the correct result.

6 Conclusions and Future Work

In this paper, we present an approach to enable the translation of STEP schema and Part 21 files defined in EXPRESS to OWL. We described the mapping rules for the translation from

EXPRESS to OWL, and outlined the benefits of OWL translation with an example. This mapping gives a semantically rich product model, which we call OntoSTEP, that can easily be integrated with OWL ontologies. Manufacturing systems today must deal with product lifecycle information that goes beyond basic 3D geometry. In this paper, we combine OntoSTEP with another ontology to add beyond-geometry information, resulting in a new semantically enriched product model. We also explained the additional benefits, reasoning, inference procedures, and queries that can be performed on the enriched model. A plug-in for Protégé is developed to enable the beyond-geometry information of the designed product to be included in the OntoSTEP representations. The Protégé plug-in can be downloaded from http://www.nist.gov/mel/msid/ontostep.cfm. We also described how these mapping rules could be implemented through meta-model based model transformations.

The OWL ontologies generated by this approach have several applications beyond the example described in this paper. One such application, for long term preservation of engineering data, is described below.

6.1 Long Term Knowledge Retention

The objective of the Long Term Knowledge Retention (LTKR) project at NIST is to define guidelines and information models to enable effective archival and retrieval of digital project model data and other engineering-related documents. One of the core challenges of long term knowledge retention is the interpretation of data stored in a format that has become obsolete.

Technologies such as OWL enable us to represent knowledge in a computer interpretable form that is suitable for long term retention. The OWL representations are generic and open, as opposed to proprietary formats. This enables widespread availability of tool support beyond the usual lifetimes of proprietary formats. The generic nature of OWL allows us to incorporate non-geometry information extracted from other documents (such as a requirement specification) into the information extracted from CAD files. This satisfies a major requirement by helping managing an archival of multiple engineering documents in a coherent manner. It also allows the extraction of descriptive information from archived documents, and the search and retrieval of archived documents.

Disclaimer

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.

7 References

1. 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, 2005, pp. 4-25.

2. Sudarsan, R., Subrahmanian, E., Bouras, A., Fenves, S., Foufou, S., and Sriram, R. D., "Information sharing and exchange in the context of product lifecycle management: Role of standards," *Computer-Aided Design*, 2008.

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

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

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

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

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

8. Schenck, D., and Wilson, P. R., *Information Modeling: The EXPRESS way*, Oxford University Press, New York,1994.

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

10. 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.

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

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

13. Krima, S., Barbau, R., Fiorentini, X., Rachuri, S., and Sriram, R., "OntoSTEP: OWL-DL Ontology for STEP," National Institue of Standards and Technology, NISTIR 7561, Gaithersburg, MD 20899, USA, 2009.

14. 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.

15. 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.

16. 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.

17. Protégé. http://protege.stanford.edu/ . 2008.

18. OWL 2 Web Ontology Language Profiles. http://www.w3.org/TR/owl2-profiles/ . 10-27-2009.

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

20. David Koonce, Lizhong Huang, and Robert Judd, "EQL an EXPRESS Query Language," *Computers & Industrial Engineering*, Vol. 35, No. 1-2, 1998, pp. 271-274.

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

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

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

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

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

26. Joshua Lubell, and Stephane Lardet. Open Source EXPRESS Parser. http://sourceforge.net/projects/osexpress/ . 2001.

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

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

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

30. LKSoft. JSDAI. http://www.jsdai.net/ . 1-14-2009.

31. Sudarsan, R., Young-Hyun, H., Foufou, S., Feng, S. C., Roy, U., Fujun W., Sriram, R. D., and Lyons, K. W., "A Model for Capturing Product Assembly Information ," *Journal of Computing and Information Science in Engineering*, 2005.

32. PTC. Pro/ENGINEER. http://www.ptc.com/ . 2010.

33. Object Management Group. Ontology definition metamodel. 2008.

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

35. Ledeczi A, Bakay A, Maroti M, Volgyesi P, Nordstrom G, Sprinkle J, and Karsai G, "Composing domain-specific design environments," *Computer*, Vol. 34, No. 11, 2001, pp. 44-51.

36. The GME User's Manual. http://www.isis.vanderbilt.edu/projects/GME . 2010.

37. Balasubramanian D, Narayanan A, vanBuskirk C, and Karsai G, "The Graph Rewriting and Transformation Language: GReAT," *Electronic Communications of the EASST*, Vol. 1, 2006.