Semantic based framework for dynamic customization of PLM-related information models

Sylvère Krima

SPIM

Thèse de Doctorat

école doctorale sciences pour l'ingénieur et microtechniques
UNIVERSITÉ DE BOURGOGNE

Semantic based framework for dynamic customization of PLM-related information models

Sylvère KRIMA

N° X X X

THÈSE présentée par

Sylvère KRIMA

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne

Spécialité : **Informatique**

Semantic based framework for dynamic customization of PLM-related information models

Soutenue le 5 Juillet 2013 devant le Jury :

Benoit EYNARD	Rapporteur	Professeur à l'Université de Technologie de Compiègne
Abdelaziz BOURAS	Rapporteur	Professeur à l'IUT Lumière - Université Lyon 2
Lionel ROUCOULES	Examinateur	Professeur à l'ENSAM de Aix-en-Provence
Yacine OUZROUT	Examinateur	Professeur à l'IUT Lumière - Université Lyon 2
Sebti FOUFOU	Directeur de thèse	Professeur à l'Université de Bourgogne

Table of contents

Table of Figures

1 Introduction

1.1 Motivations

We live in the information age. Data has become an essential asset for most everyday situations. The ability to share data, to generate information, and create new knowledge from that data is common to all fields of research and all economic activities (Moon, Fewell, & Reynolds, 2008)(Irimia, 2011). Whether it is financial data, product data, health data, or disaster data, managing that data is a critical, and sometimes costly (Gallaher et al., 2004), process. To manage data well, we must understand that it has a life cycle that is composed of several steps including definition, instantiation, transformation, validation and archival. When not properly defined, data might become incomplete, inconsistent or even worse, unusable (Benson, 2010). Since we live in a dynamic world, requirements about data evolve and people meet the need for defining new data or updating existing ones over the data life cycle. This has became a technological challenge and an important issue since it is hardly possible to define, in advance, information structures that meet requirements you do not know yet.

This situation is especially true in domains such as manufacturing (Brunnermeier & Martin, 1999) where information exchange involves many actors and data is shared across multiple functions and software applications. In these situations, each function has its own needs and each application has its own input/output requirements. As a result, it becomes hard to find a common information structure for representing data. The challenge is even bigger when a temporal aspect has to be considered since it requires the ability to tailor the information structure dynamically over time. One area within the manufacturing domain that we have identified with these characteristics is Product Life-cycle Management (PLM). PLM involves many global actors using a myriad of software applications that perform a series of product management functions that can last from weeks to decades.

PLM has always required robust solutions for representing product data models. With the growth of the Model Based Enterprise (MBE) initiative product data representation becomes more important than ever. Product data models enable information exchange across different organizations, actors, processes and stages in the product life cycle. In this context, standardization of models plays a key role, since it ensures interoperability between the different systems that support information exchange. These standard models need to support diverse domain-specific requirements from the multitude of disciplines involved during a product life cycle. Due to this diversity of requirements, issues are to (1) develop multidisciplinary models, (2) extend these models to support new requirements over time (new products, new regulations, new materials, new processes, ...), and (3) implement the resulting gigantic information models. Because the mechanisms to extend models is static by its nature, requiring numerous updates to the initial information model, extending data models is expensive in cost and time. It requires an understanding of the entire initial model to insure correct extensions are developed. Software components need to be updated so they can exchange, understand, and use the information in the new model. Finding an alternative is crucial when dealing with complex products and multiple requirements, which is typical of PLM.

1.2 Thesis objectives

The objective of this PhD work is to provide an alternative solution, for PLM, to static extension of information models in a way that reduces complexity and cost, of development and implementation. To achieve this alternative we will identify existing frameworks for dynamic customization of information models and evaluate them, based on PLM-related requirements to prove the need for a new framework. This evaluation will help us to identify and understand the weaknesses and lacks in existing solutions in order to develop a more robust one. The new solution will benefits from recent and emerging technologies in the field of information representation, management and validation.

Developing such an alternative involves two major challenges. The first is to develop methodologies and tools that enable PLM users to build static information models, with a finite number of concepts, that can be extended easily to represent new concepts without major modifications to either the original models or the software that uses them. Because PLM users already have their “preferred technologies and languages” for building information models, the second challenge is to demonstrate simple transitions from what they use now to the methods and tools we recommend.

Addressing these challenges is a demanding effort because it requires (1) a high level understanding of PLM as one needs to identify PLM key characteristics, best practices and frequently used technologies and languages and (2) advanced knowledge of information modeling techniques to find the adequate mechanisms for PLM-related information models.

To achieve our objective, tackle these challenges, and overcome the drawbacks and weaknesses of existing approaches, we propose the following research plan.

- Define a new information model that when properly extended enables users to add new concepts and properties dynamically.
- Propose a novel method for defining, and enabling consistency checking of, new concepts and properties dynamically and formally using OWL.
- Propose a new UML based method and a tool for graphically defining instantiation patterns of the previously defined concepts.
- Develop a new method and a tool to map EXPRESS information models and data into OWL. This enables consistency checking and data validation using SPIN.
- Develop a new method for representing and executing business rules on dynamically created concepts and properties using SPIN.

- Define a new method for graphically and formally representing families of products.

1.3 Organization of the thesis

The rest of the thesis is organized as follows:

Chapter 2 gives a detailed presentation of the product lifecycle management (PLM) environment. This chapter also introduces the different issues related to data exchange in heterogeneous environment and how information standards solve some of these issues. Chapter 2 also presents related work performed in the field of dynamic extension of information models. This chapter describes research works performed at the National Institute of Standards and Technology and solutions provided by major standardization bodies including ISO and OASIS as well as industrial work with OAGi OAGIS framework.

Chapter 3 presents requirements in terms of information modeling for PLM. From these requirements, metrics have been developed and later been used to compare and evaluate the related work in this area. This chapter identifies concepts that can be reused from the related work. It also illustrates gaps that need to be fulfilled in order to meet the requirements for PLM.

Chapter 4 presents solutions to some of the gaps identified in Chapter 3. Our goal is to both provide new solutions wherever none exists and also improve existing solutions. In this section, our approach is to use recent and emerging technologies in the field of information representation and validation.

Chapter 5 discusses how different technologies can be used together, when necessary, to address PLM requirements. We will identify the role of these different technologies and motivate their use both from a technical perspective and human perspective. We then show how to use these technologies and our framework to solve two different PLM-related use cases: one to address a product modeling problem and one to address a sustainability problem.

Chapter 6 concludes this dissertation and summarizes the main contributions from the previous chapters. It briefly describes our problems and our solutions. Finally, it discusses some open questions that might possibly motivate some future work on the topic.

2 State of the Art

2.1 The PLM environment

For the past few decades, the evolution of computer systems has greatly enhanced mankind's ability to design and manufacture more advanced products. While these advanced products have made our lives easier, their lives have become much more complex. New requirements and regulations have changed their components and performances. For instance, the need to save natural resources and regulations regarding waste reductions and material compositions require significant changes the types of rare earth minerals and toxic chemicals in most electronic products. These products, and many others, must be more eco-friendly, contain more recyclable components and materials, and should be produced using more energy efficient manufacturing processes and machines.

These new requirements and regulations have also added stages to the life cycle of these products. Examples of new stages include life-cycle assessment, remanufacturing, and recycling. Unfortunately, these additional stages have two important impacts on the way the life cycle must be managed. First, these new stages will be executed by new entities, or actors, typically distributed around the world. Second, these different actors and the activities they perform create a demand for an enormous amount of information including 3D models, spreadsheets, and technical documentation. This information is generated by and processed by a large variety of business and engineering software systems. Managing the stages, actors, and information is the role of the Product Lifecycle Management (PLM) system.

2.1.1 Collaboration and data exchange

PLM, therefore, provides the backbone for collaboration within individual companies and across their extended enterprises. It makes product-related information accessible to all the actors and to all the software systems across the entire product life cycle. And by collaboration we include both intra-process collaboration such as designer to designer and inter-process collaboration such as designer to manufacturer. This collaboration is based on the exchange of information among the different actors and processes involved. Automation has had a great impact on the ways collaboration and information exchange take place. Both take place using software applications that process, produce, and communicate only digital data. Consequently, traditional collaborations that used to depend on people-to-people interactions now depend on computer-to-computer interactions.

These new computer-to-computer interactions depend critically on information exchanges. For these exchanges to be successful, a transformation is required - a transformation from human interpretable information to computer interpretable information. Currently, this transformation is achieved using models that represent information as a set of domain-related concepts and relationships. These models can take several different forms including text, graphs, and mathematical formulas. Some are more understandable by humans than others; some are more formal than others. But, they all can be understood by computers. They can all generate a technology-dependant 'data schema' or 'information model'. Software systems use those schemas - with the appropriate translators and parsers- to produce and exchange information as instances of the concepts and relationships of the domain. Since there is no absolute notion of correctness, software vendors, over time, have developed their own proprietary data schemas to represent the same domain information. Even though these schemas tend to have some overlaps, they are not the same.

This diversity of information models, and the software applications that produce and use them, are significant barriers to successful interoperability within PLM. To make interoperability possible for PLM, numerous mappings are required. Some mappings are between different information models while others are between different representations of the information. Unfortunately, not all the information can be mapped from one representation to another. This can lead to information loss during the mapping. To avoid this loss, it is necessary to reduce the number of distinct information models. One way to achieve this reduction is through standards.

2.1.2 The role of standards for data exchange

Information standards provide information models and data schemas that can be used to simplify data exchange. Figure 2-1 shows the mapping strategies necessary for data exchange in a heterogeneous environment composed of 4 systems (A, B, C, D) when there is no standard (left), and when all actors agree to base their mappings on a given standard (right). In the left figure, each actor must develop three mappings; in the right figure that number is reduced to one.

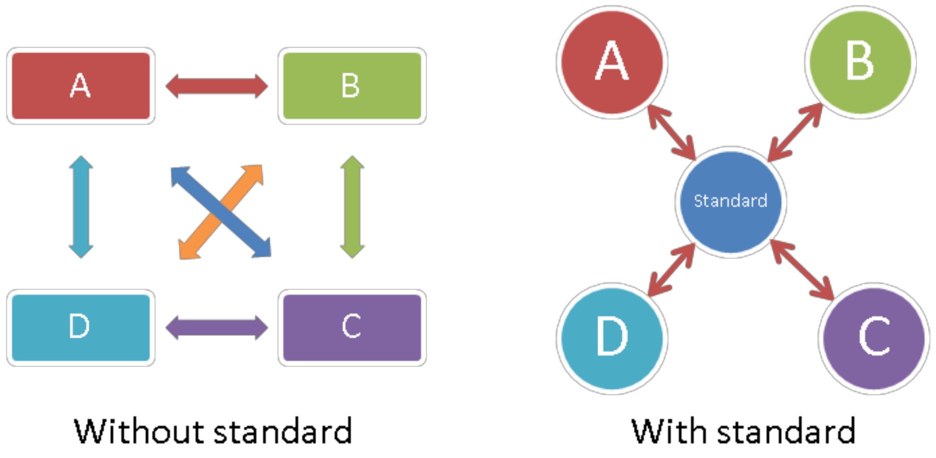


Figure 2-1 Role of standards for mapping in heterogeneous environment

More importantly, as the number of actors increases, the number of mappings increases in the left figure. But, the number remains one in the right figure. This is the benefit of standards.

2.1.3 Summary

PLM is a domain in which numerous software applications exchange information. Because of this, collaboration is a crucial aspect of PLM. Unfortunately, heterogeneity makes the collaboration complex and expensive. Information standards are a commonly used solution to heterogeneity. This solution, however, has not worked for PLM because (1) it is hardly possible to reach an agreement on a single standard to cover all of the information exchanged within the PLM context, (2) even if this is possible, such a standard would be too big to be implemented, (3) standardized information models tend to be static representations of knowledge and, therefore, could not handle the dynamic aspect of PLM. Nevertheless, people have been trying to overcome these 3 aspects by providing mechanisms to customize information model standards dynamically.

2.2 Related work

Customization of information model for product data is not a new research area. Back in 1995, West (West, 1995) described the need for information modelers to be able to build flexible and extensible models to respond to the evolving businesses. At this time West had already developed a solution, the Generic Entity Framework (GEF) (West, 1994) to support generic modeling of information. This framework enables development of dynamically customizable information model and served as a building block for ISO 15926 (ISO, 2003), in 2003. ISO 15926 modeling architecture and techniques were inspired by West's work and provide foundation for developing domain-specific information models that enable data exchange related to Oil and Gas facilities. The ISO 15926 reference architecture is semantically rich and make a strong usage of semantic technologies, especially ontologies to introduce domain-specific terms, which respond to the business needs and specialize initial generic terms. Since 1994 the International Organization for Standardisation (ISO) has been grappling with modeling issues in its ISO 10303 standard (ISO, 1994a; Pratt, 2001) (informally known as STEP - STandard for Exchange of Product model data) and published, in 2004, ISO 10303-1275 (ISO, 2004), a module that enables dynamic extension of STEP information models in a similar fashion as ISO 15926. This approach has been intensively used with 10303-239: Product Life Cycle Support (ISO, n.d.-a), which is a highly generic information model, that needs external libraries of domain specific terms for specialization and tailoring of the information model (Price & Bodington, 2004). The Organization for the Advancement of Structured Information Standards (OASIS) has then, in 2005, developed a methodology, known as DEXlib (OASIS, 2010a), for formalizing domain specific data exchange specifications based on 10303-239. 1994 was also the year when the Open Applications Group Inc (OAGi) started working on its Open Application Group Integration Specification (OAGIS) and faced the same issues of dynamic customization of information model.

In parallel to these industrial efforts, the research community has been very much active on solving the same issues. In 2000, the National Institute of Standards and Technology was developing the NIST Design Repository (Szykman, Racz, Bochenek, & Sriram, 2000) with the goal of providing a generic information modeling framework for building design repositories. This work has driven Fenves, in 2002, and his Core Product Model (CPM) (Fenves, 2002), a generic information model for representing design information. In 2005, CPM has been slightly remodeled to extend its support to PLM (Foufou, Fenves, Bock, Rachuri, & Sriram, 2005).

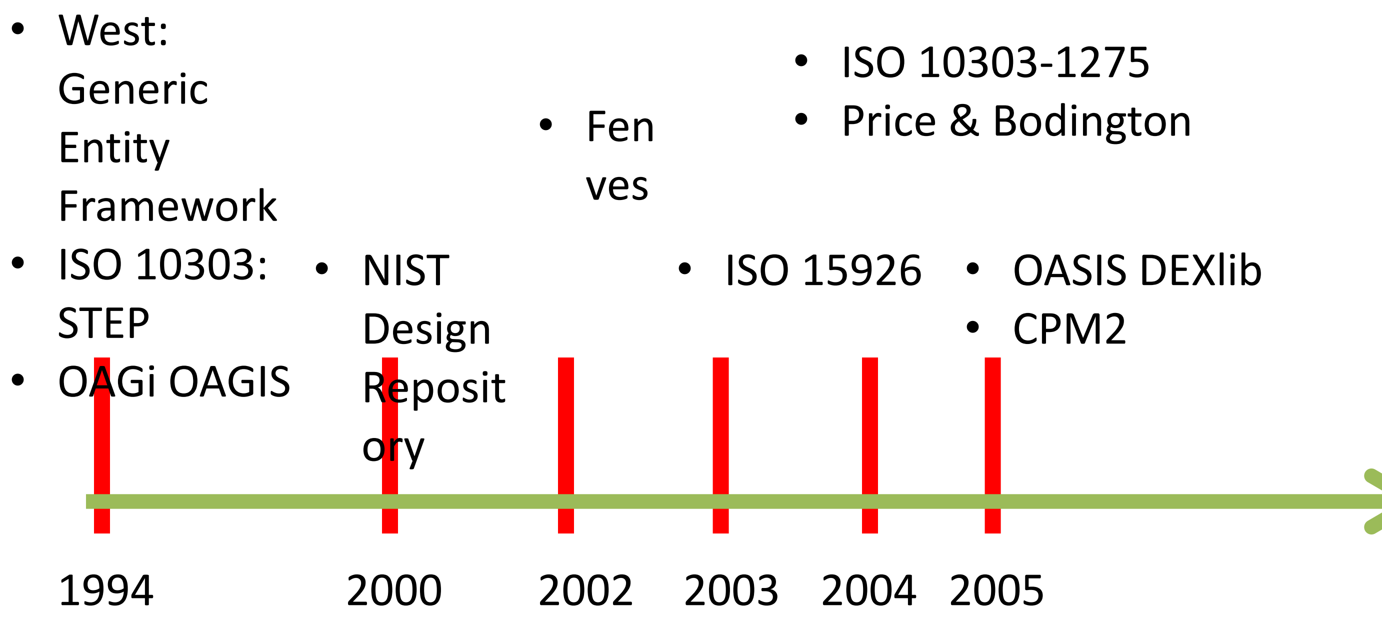


Figure 2-2 History of the different product information modeling frameworks with dynamic customization feature

Other large domains have seen similar efforts, such as the National Information Exchange Model (NIEM)[1], developed by the United States Department of Justice. NIEM is a generic information modeling framework for exchange of information related to justice, public safety, emergency and disaster management, intelligence and homeland security. NIEM has been built on top of the Global Justice XML Data Model (GJXDM)[2] that was designed only to represent and exchange information within the justice and public safety communities. The GJXDM extension mechanism is reused in NIEM, and is similar to the OAGIS mechanisms (See 2.2.4). Other frameworks have been developed but they do not cover a larger domain or are not technically different from the previously cited.

[1] More information can be found at http://www.niem.gov
[2] More information can be found at http://it.ojp.gov/jxdm/

Two recurrent types of customization mechanisms have been used in these efforts: extension and specialization. Extension expands the information model by adding new concepts and/or relationships. Specialization uses specific data (information instances) values to classify existing concepts and/or relationships. The following sections describe important efforts related to product information modeling. Section 2.2.1 presents the NIST Core Product Model and its two extension mechanisms. Section 2.2.2 introduces ISO 10303 and the customization mechanisms it provides. Section 2.2.3 describes the OASIS framework for specializing ISO 10303-239. Finally, Section 2.2.4 introduces OAGi OAGIS and its different extension mechanisms.

2.2.1 NIST Core Product Model (CPM)

CPM is an extensible conceptual representation of a product (Foufou et al., 2005). More importantly, it is not tied to any particular engineering domain or implementation technology. CPM is intended to serve as a basis for whatever extensions and specializations might be needed to meet domain-specific requirements. Extensions and specializations differ in how they refine the conceptual model. Extensions are achieved by adding new concepts to the initial conceptual model, thus increasing the conceptual model's scope. An example of an extension to CPM is the Open Assembly Model (Rachuri et al., 2006), which adds concepts for representing assembly structure and kinematics information. Specializations, on the other hand, add domain-specific semantics to the concepts initially present in the model. This results in a model that is no longer purely conceptual but rather is tied to a particular application area or business context.

This distinction between extension and specialization is made possible within CPM because of its 3-layer architecture. The top layer is a conceptual representation of the product model without domain-specific semantics. The intermediate layer is an instantiation of the conceptual model. It contains domain-specific semantics, but those layers are not tied to any specific technology. The bottom layer is a technology-specific representation of the intermediate model.

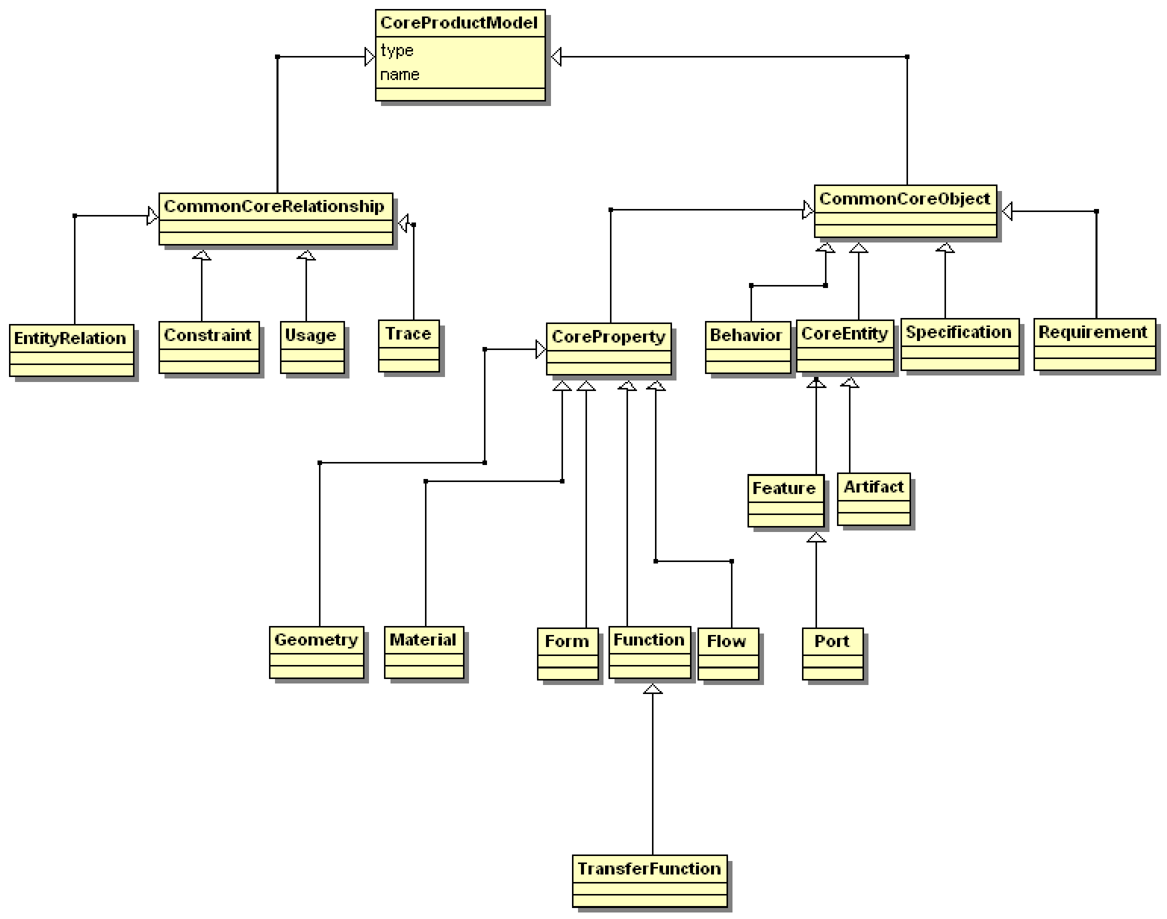


Figure 2-3 Simplified Core Product Model (CPM)

Figure 2-3 shows a simplified version of the CPM conceptual model, the top layer. This version is simplified because it only displays UML classes and UML generalizations. If we look at this model more carefully, we can see that an Artifact can be an aggregation of sub artifacts (Figure 2-4).

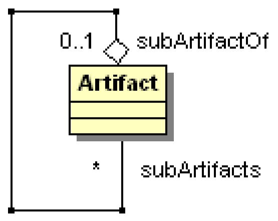


Figure 2-4 CPM Artifact

Using this aggregation, we can construct an intermediate model of a car as a set of 4 wheels, 1 engine and 1 body (Figure 2-5). This intermediate model represents a Car as an instance of the class Artifact named Car. The instance is as an aggregation of 6 other instances of Artifact named Engine, Body, Wheel1, Wheel2, Wheel3 and Wheel4.

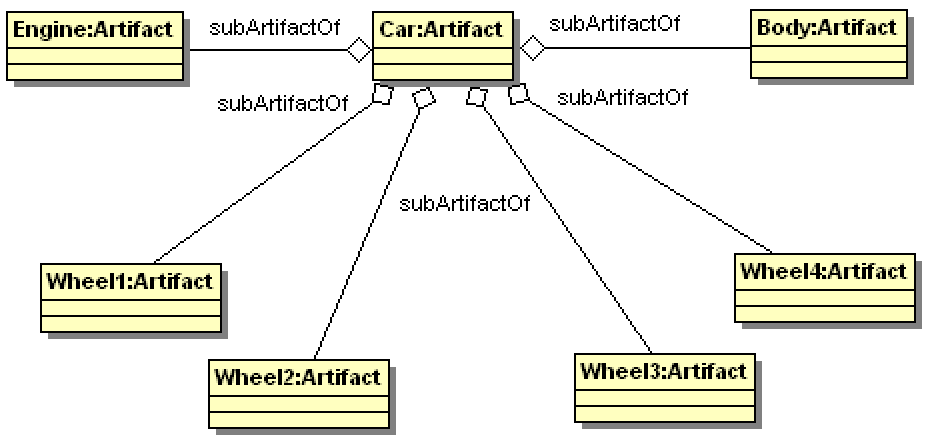
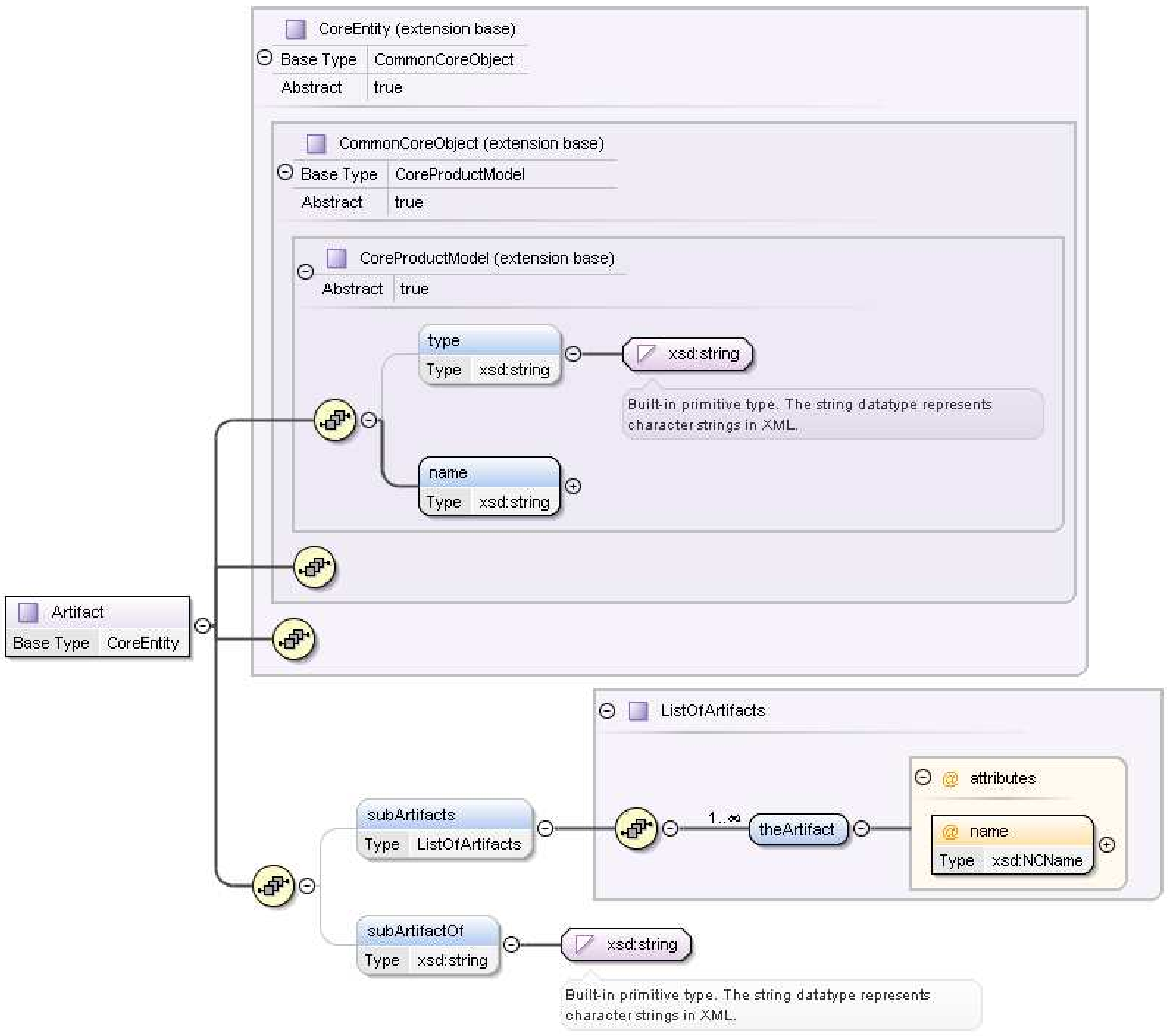


Figure 2-5 CPM intermediate model example

The intermediate model is used to generate implementation models. One way to represent a CPM implementation model is in XML Schema. In (Foufou et al., 2005) the authors provide such an XML schema, called Core Product XML Schema (CPXS). Figure 2-6 shows the Artifact class and its inherited attributes in CPXS.



CoreEntity (extension base)
Base Type CommonCoreObject
Abstract true
CommonCoreObject (extension base)
Base Type CoreProductModel
Abstract true
CoreProductModel (extension base)
Abstract true
type
Type xsd:string
xsd:string
Built-in primitive type. The string datatype represents character strings in XML.
name
Type xsd:string
Artifact
Base Type CoreEntity
ListOfArtifacts
attributes
1..∞
theArtifact
name
Type xsd:NCName
subArtifacts
Type ListOfArtifacts
subArtifactOf
Type xsd:string
xsd:string
Built-in primitive type. The string datatype represents character strings in XML.

Figure 2-6 CPXS representation of CPM Artifact

Authors of CPXS provide a methodology for writing a CPXS compliant XML file. By applying this methodology to the car model shown in Figure 2-5, we obtain the following XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns="http://namespace.nist.gov/msid/cpm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="file:/C:/cpm2Schema.xsd">
    <artifact>
        <name>Car</name>
        <subArtifacts>
            <theArtifact name="Engine"/>
            <theArtifact name="Body"/>
            <theArtifact name="Wheel1"/>
            <theArtifact name="Wheel2"/>
            <theArtifact name="Wheel3"/>
            <theArtifact name="Wheel4"/>
        </subArtifacts>
    </artifact>
    <artifact>
        <name>Engine</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact>
        <name>Body</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact>
        <name>Wheel1</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact>
        <name>Wheel2</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact>
        <name>Wheel3</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact>
        <name>Wheel4</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
</model>
```

This XML file in particular and implementation models in general, serve multiple purposes. They can be used for data exchange, knowledge representation, and archival of engineering data among others.

The CPM conceptual layer limits the types of domain-specific semantics that can be incorporated into the intermediate layer. Furthermore, these semantics are only human interpretable, which means that they can be understood only from the name of the instances. To overcome these limitations, CPM provides mechanisms for extensions at the conceptual layer and specializations at the intermediate layer. We describe these mechanisms in the next two sections.

2.2.1.1 Extension

CPM allows users to extend the conceptual model by adding new concepts and relationships. These new concepts and relationships are represented in a UML class diagram containing new classes and associations. This new UML class diagram is an extension of the CPM if there is at least one UML association shared by both the CPM and the new UML class diagram. An example of such an extension is the Open Assembly Model (OAM)(Rachuri et al., 2006). The OAM extends the CPM to include assemblies and tolerances.

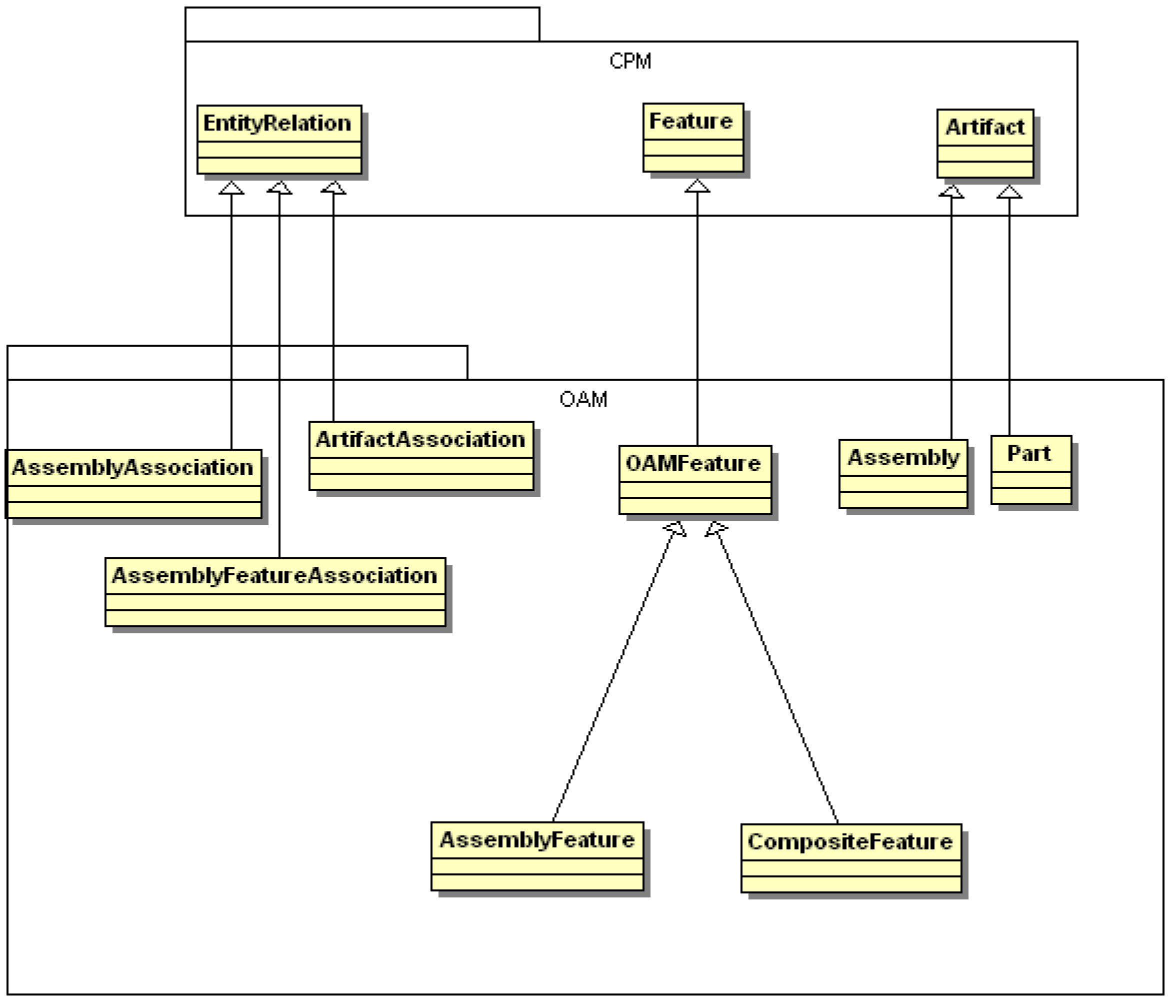


Figure 2-7 Simplified Open Assembly Model (OAM)

Figure 2-7 shows a simplified subset of the OAM conceptual model and associations between the OAM conceptual model and the CPM conceptual model as follows:

- OAM::AssemblyAssociation is a subclass of CPM::EntityRelation
- OAM::AssemblyFeatureAssociation is a subclass of CPM::EntityRelation
- OAM::ArtifactAssociation is a subclass of CPM::EntityRelation
- OAM::OAMFeature is a subclass of CPM::Feature
- OAM::Assembly is a subclass of CPM::Artifact
- OAM::Part is a subclass of CPM::Artifact

The existence of these associations is enough to consider the OAM conceptual model as a hierarchical extension of CPM.

Another extension could be developed for representing specific needs from the automotive industry. In order to model cars as we did previously in the introduction, a CPM extension could be developed as shown in Figure 2-8.

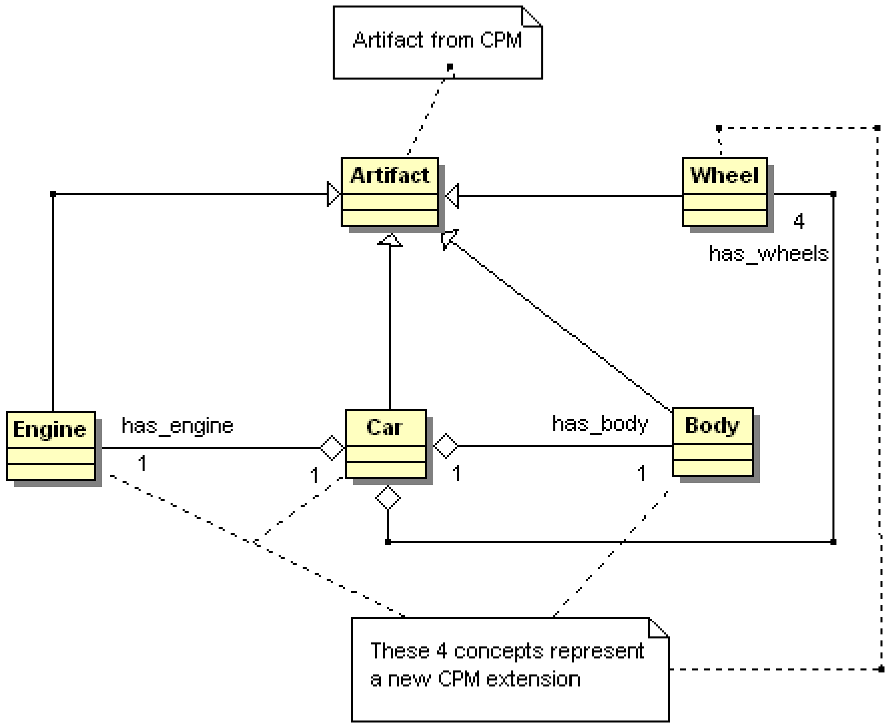


Figure 2-8 Core Product Model extension

This extension introduces 4 new concepts: Car, Body, Wheel and Engine. It highlights how extensions can be used to embed domain-specific semantics and make it computer readable using the expressiveness of UML. A Car is formally defined as an aggregation of Wheels, Body and Engine. This semantics could not be expressed previously in the way we modeled a car in the introduction.

To create this extension, we would end up with an intermediate model like Figure 2-9 where all the 4 wheels are instances of Wheel, the engine is an instance of Engine, the body is an instance of Body and the car itself is an instance of Car.

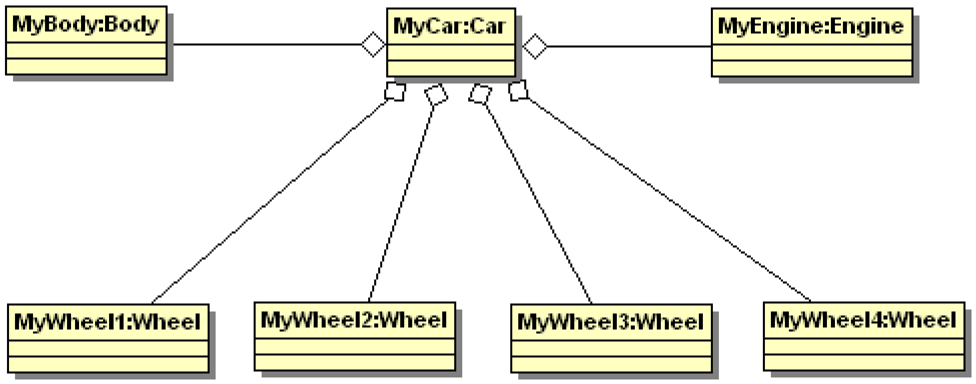


Figure 2-9 CPM extension instantiation

In a scenario of data exchange, the information from the intermediate model would be exchanged through the implementation model. Two challenges arise if the implementation model is based on CPSX. First, CPXS itself must be modified, since it was not designed to represent the new concepts introduced by the extension. Second, CPXS interpreters must also be modified since they will not be able to read the new data unless they understand the new extension.

2.2.1.2 Specialization

To overcome the aforementioned challenges, CPM has a specialization mechanism, which allows users to add more semantics while conserving the initial implementation model. The specialization mechanism makes use of a string attribute called <u>type</u> in the CoreProductModel class. Because of the inheritance property, any CPM class will share its attributes meaning that any CPM class has the attribute <u>type</u>. This attribute <u>type</u> can be used as a classifier meaning its value classifies the instance which owns it. If the attribute <u>type</u> of an instance of Artifact has its value set to “Car”, this implies that this instance is a car. The car example previously developed is reused in this section and its instantiation is shown with an intermediate model in Figure 2-10.

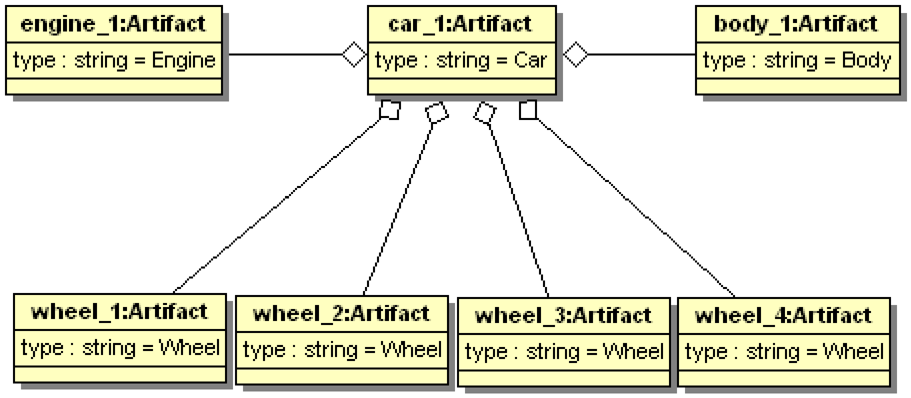


Figure 2-10 Instantiation of CPM

In this intermediate model, each object uses the attribute type as a classifier. Authors of CPM recommend to value this attribute using terms from externally defined taxonomies that include the semantics of the terms. By externalizing the formal descriptions of concepts with taxonomies, we can use the original CPXS to represent the implementation model (see the following example). Being a hierarchical classification of terms, taxonomies do not have as much expressiveness as UML. So, the semantics they define is limited to generalizations and specializations relationships.

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns="http://namespace.nist.gov/msid/cpm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="file:/C:/cpm2Schema.xsd">
    <artifact type="Car">
        <name>car_1</name>
        <subArtifacts>
            <theArtifact name="engine_1"/>
            <theArtifact name="body_1"/>
            <theArtifact name="wheel_1"/>
            <theArtifact name="wheel_2"/>
            <theArtifact name="wheel_3"/>
            <theArtifact name="wheel_3"/>
        </subArtifacts>
    </artifact>
    <artifact type="Engine">
        <name>engine_1</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact type="Body">
        <name>body_1</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
        <artifact type="Wheel">
        <name>wheel_1</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact type="Wheel">
        <name>wheel_2</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact type="Wheel">
        <name>wheel_3</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
    <artifact type="Wheel">
```

```
        <name>wheel_4</name>
        <subArtifactOf>Car</subArtifactOf>
    </artifact>
</model>
```

2.2.1.3 Summary

In this subsection, we presented the NIST Core Product Model (CPM), which is a generic UML information model for PLM. Due to the generic nature of CPM, its authors provide 2 mechanisms to support domain-specific semantics. The first, extension, adds new concepts and more semantics using the expressiveness of UML. This, of course, changes the underlying model, which changes the resulting implementation model. The second mechanism, specialization, refines the semantics of the existing CPM concepts by classifying them with external taxonomies. This solution is less expressive than an extension; but it does not change the underlying data model. Consequently, specialization provides better support for data exchange.

2.2.2 ISO STEP

The International Organization for Standard (ISO) worked for years to develop a standard for representing and exchanging product model data. This standard is ISO 10303 *Automation systems and integration — Product data representation and exchange.* It is also informally known as STEP or STandard for the Exchange of Product model data. STEP has been divided into subsets, called Application Protocols (APs), to ease its use and implementation (Pratt, 2001). Although STEP has a wide scope it does not cover all the user's needs. To overcome this issue, STEP provides two mechanisms that enable customization for domain-specific needs. First, users can define and add new attributes to existing concepts. Second, users can classify STEP instances with an externally controlled vocabulary - this is called external classification. Both mechanisms are implemented using STEP modules, which are common information models reused by different APs. These modules are developed using a standard language, ISO 10303-11 *Description methods: The EXPRESS language reference manual* (ISO, 1994b). The EXPRESS language has a textual representation and a graphical notation, called EXPRESS-G. The implementation method for EXPRESS information models is known as ISO 10303-21, *Implementation methods: Clear text encoding of the exchange structure* (simply called Part 21) (ISO, n.d.-b). Both STEP customization mechanisms are implemented with this method. They are described in the following sections.

2.2.2.1 User defined attributes

As noted above, users can add new properties to some entities. This is done using the mechanism described in ISO 10303-41 *Integrated generic resource: Fundamentals of product description and support* (ISO, 2005). This mechanism is known informally as "user defined attributes" (UDA). These new attributes are represented as key-value pairs. They can be dynamically connected to

- A Part or a Product
- A specific instance of a component in an assembly
- A portion of a part shape

To associate a new property with an instance of a type that accepts new properties, four steps are required. First, we must create a type of property using the **general_property** entity. Next we create a **property_definition** that will be used to characterize an object. Then, we connect the **property_definition** to a type of property represented by an instance of **general_property** through an instance of **general_property_association.** Finally, the value of the property is expressed with an instance of a subtype of **representation_item** and connected to the property with an instance of **representation** and **property_definition_representation**. Figure 2-11 shows an example of the creation of a property 'type' with a value 'Car'.

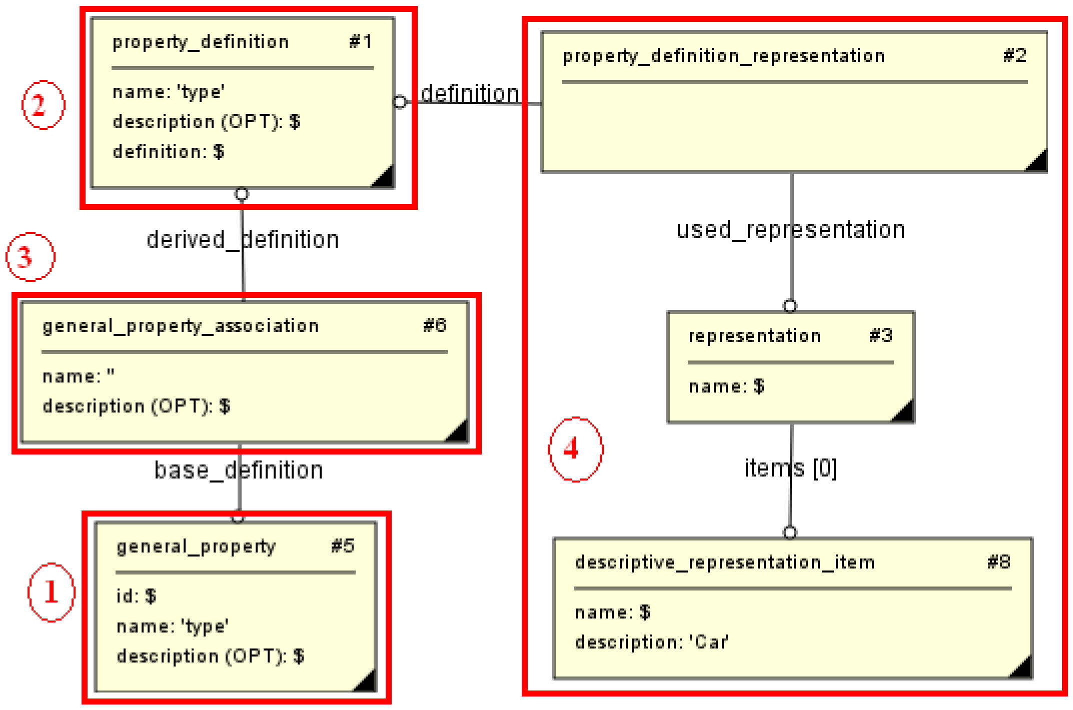


Figure 2-11 STEP instances

Once a property is created, it must be connected to an object. To do so, the instance of **property_definition** is connected to the characterized object through the **property_definition.definition** relationship. As shown in the following example (Figure 2-12) we create a new property "Year" that represents the production year of a product. We then instantiate the entities required by ISO 10303-41 to represent a product and connect this product to the "Year" property.

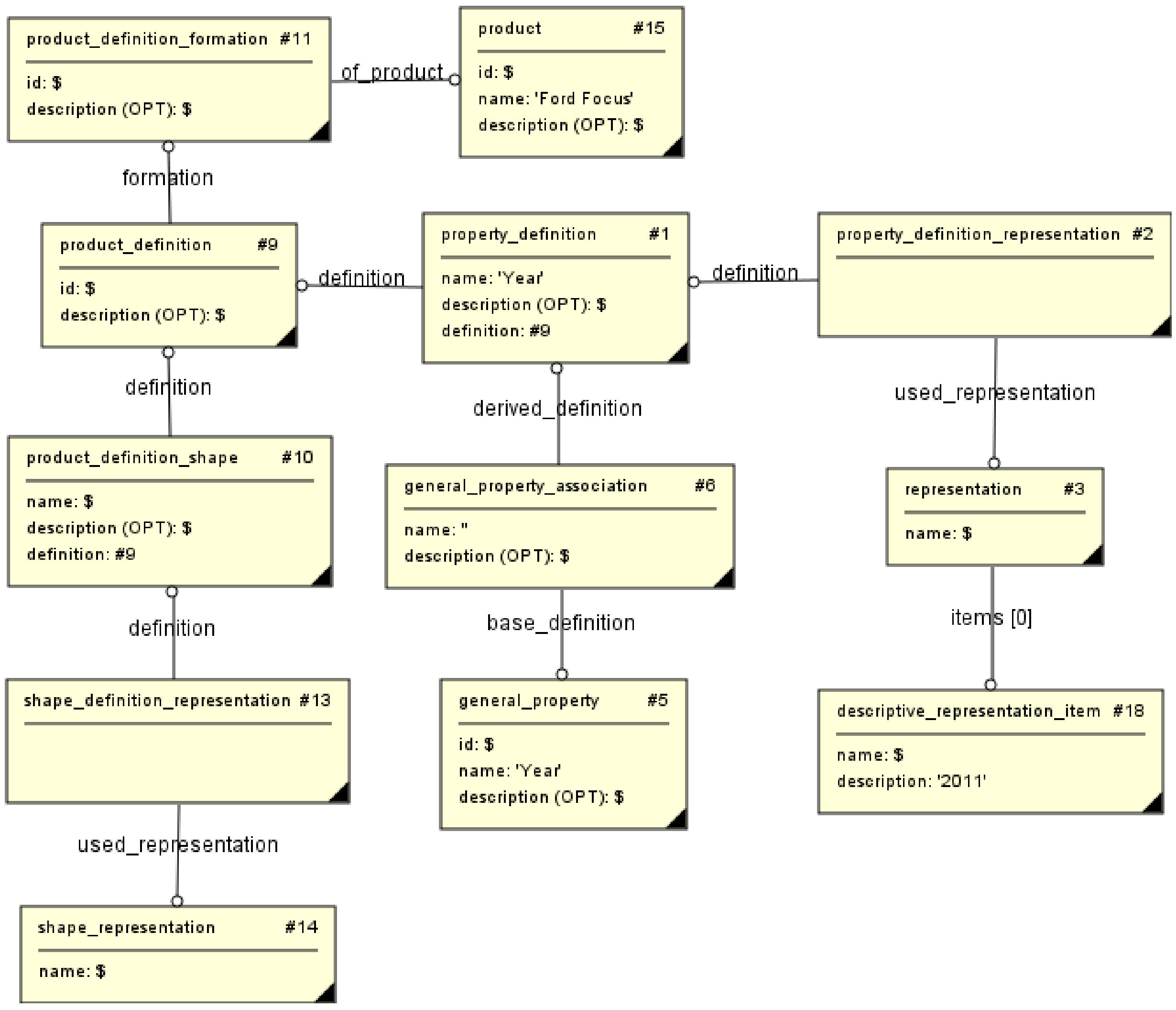


Figure 2-12 STEP instantiation of a user defined attribute

In this example, the semantics of the property "Year" resides only in its name, which makes it only human interpretable. Since machines do not, in general, understand English, they will not be able to process the information and understand its meaning. To overcome this limitation, ISO 10303 also provides a mechanism based on external classification.

2.2.2.2 External classification

Although the UDA mechanism gives users the possibility to add new properties to instances, those properties have no formally defined semantics. STEP provides a mechanism to (1) define the semantics formally with an external classification - such as a taxonomy or controlled vocabulary and (2) use it to classify instances so each instance will contain a link to its formal definition.

To establish links between an instance and its external definition, STEP uses three entities: **Classification_assignment**, **External_class** and **External_class_library**. **External_class_library** represents an external classification, **External_class** represents a classifier formally defined in the external classification and **Classification_assignment** is the way to apply the external classifier to an instance. Figure 2-13 shows an example of classification where an instance of **Product** is classified as a 'Car', 'Car' being an external concept formally defined in the external classification whose identifier is 'http://myontology.org'.

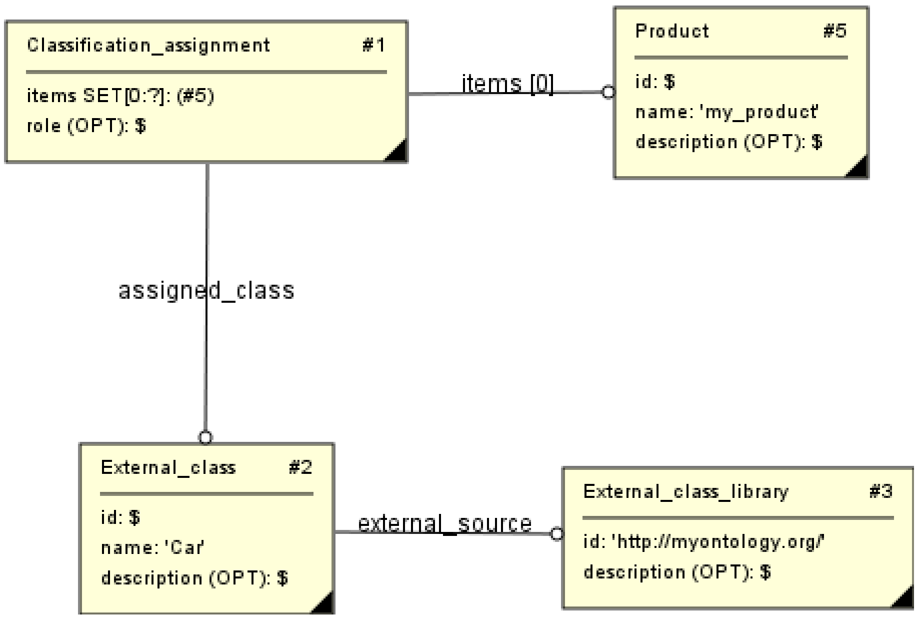
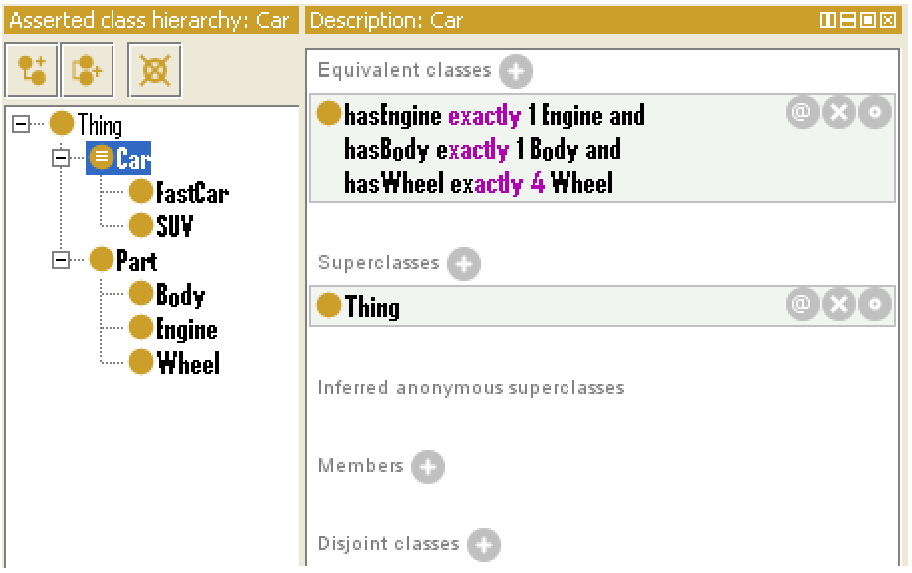


Figure 2-13 STEP instantiation of an external classification

In this example we chose to represent the external classification using an ontology since STEP does not provide any restriction on the formalism to use. Figure 2-14 shows a screenshot of the Protégé editor used to create our ontology. This ontology formally defines 'Car' and few other concepts. Any of these concepts can be used to classify STEP instances.



hasEngine exactly 1 Engine and
hasBody exactly 1 Body and
hasWheel exactly 4 Wheel

Figure 2-14 Controlled vocabulary for external classification

2.2.2.3 Summary

In this section we described ISO's approach to enable customization of its ISO 10303 standard, informally known as STEP. STEP provides two customization mechanisms. One is based on the use of user-defined attributes, where end-users dynamically add new attributes to entities. Unfortunately, since the semantics of these new attributes is embedded in their names, they are not computer processable. To overcome deficiency, users can classify instances with externally defined controlled vocabularies, specifically ontologies. Even though these mechanisms differ in the level of semantics they provide, they both have no impact on the data schema.

2.2.3 OASIS DEX for PLCS

In 2005, ISO published a new AP, ISO 10303-239: *Industrial automation systems and integration -- Product data representation and exchange -- Part 239: Application protocol: Product life cycle support* (ISO, n.d.-a). This standard is known informally as PLCS. AP239 has a wide scope and is agnosticism with respect to any particular business context. This makes AP239 broadly applicable but hard to understand, implement, and use as a whole. OASIS provides a customizable architecture that allows users to work with a subset of the original information model. This approach is somewhat analogous to STEP conformance classes, which can be used to represent a static subset of the AP's information model. Rather than specify an inflexible and static set of conformance classes, the PLCS architecture enables the definition of business, context-specific subsets of AP239 called DEXs (Data EXchange specifications) (OASIS, 2010a). DEXs use templates (OASIS, 2010b) to define how PLCS entities and their attributes will be instantiated. These templates enable customizations without sacrificing breadth or interoperability. Instantiation uses an externally defined controlled vocabulary, usually found in a Reference Data Library (RDL) (OASIS, 2010c). Template instantiations are defined using an Instantiation Path (IP) (OASIS, 2010b). The IP uses a procedural language that describes, in a computer interpretable fashion, the information instantiations performed by a template. The overall architecture of PLCS DEXs is shown in Figure 2-15.

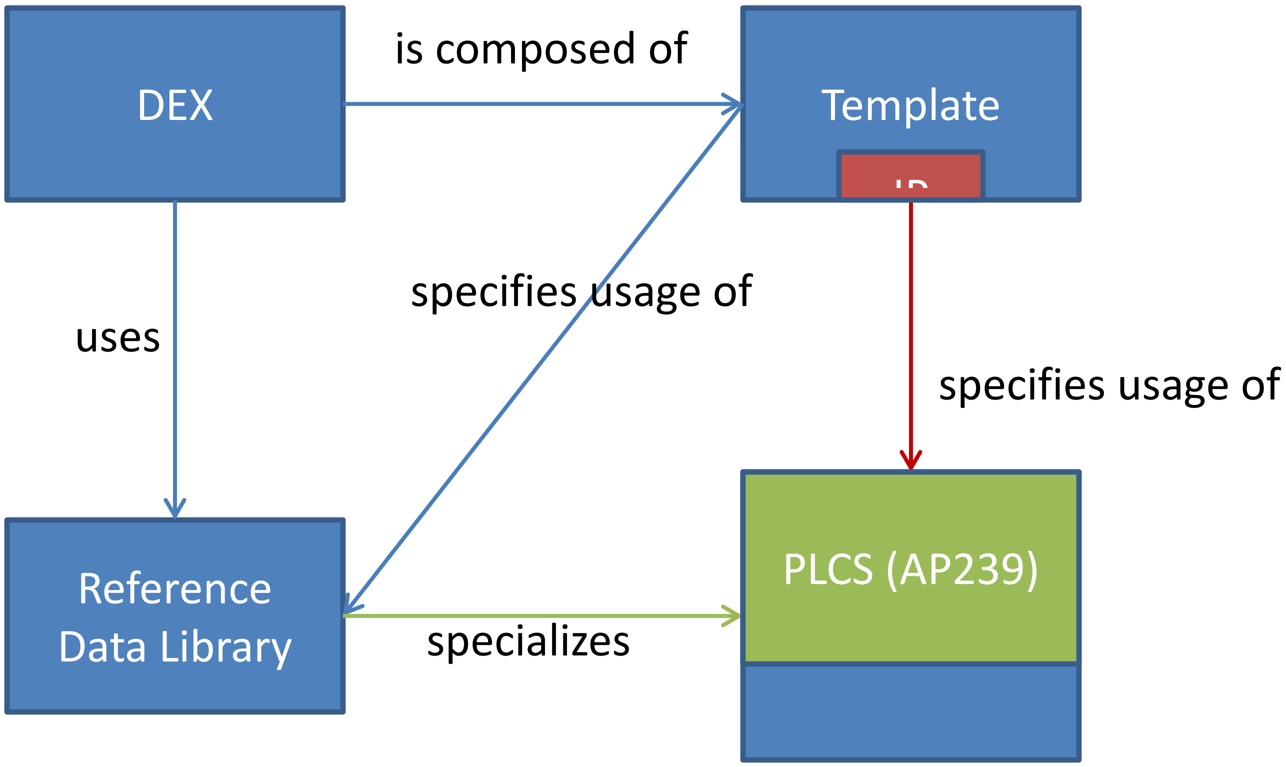


Figure 2-15 OASIS DEX for PLCS Architecture

2.2.3.1 Templates and RDL

A template is a way of making an abstraction of the AP239 data level since it represents higher-level information. In this context, templates represent business/engineering objects. The AP239 data model is a way of representing and exchanging these objects in a computer interpretable fashion. Templates combined with the Reference Data Library (RDL) provide a form of customization that allows users to control and expand the scope of PLCS, or to introduce domain-specific terminology.

Templates have a very special feature: they use only small subsets of the original AP239 data model. Even the bigger templates rarely use more than 50 concepts from AP239. OASIS also recommends naming conventions for templates. These naming conventions make it easy to introduce domain-specific terminology for human consumption. OASIS also classifies templates as 1) PLCS templates provided by OASIS as a core foundation to build new ones, 2) Business templates that are domain-specific templates created by users to satisfy their needs, and are the components where part of the customization happens.

Templates are instantiated using an externally defined controlled vocabulary, usually found in a Reference Data Library (RDL). The RDL for PLCS is based on the STEP external classification. It works in the same way and is implemented with the same entities. Any valid classification implemented with STEP is a valid PLCS classification. Although STEP does not provide recommendations on a formalism to use to define external classifications, OASIS makes it mandatory to use ontologies and the OWL language to represent the RDL. Moreover, PLCS external classifications must be derived from a PLCS proxy ontology, which is an ontologized version of the PLCS data model.

The mechanism for implementing template instantiations, called Instantiation path (IP), specifies invocation, and usage. But, it is only one component of a template. A template additionally contains:

- A textual documentation that describes the role of the template
- A textual description of the input parameters and the output
- A graphical information model expressed in an EXPRESS-G based graphical language
- Some uniqueness constraints
- One or more instance diagram(s)

Now let us consider the template Assigning_reference_data (OASIS, 2009) whose information model is shown in Figure 2-16. This template describes a classification of something, where the class's definition is specified in an external RDL. Because classification is fundamental to the usage of AP239, DEXs use this template more than any other. Readers familiar with EXPRESS-G will notice that this diagram includes the following non-EXPRESS-G annotations:

- The textual annotation beginning with the '^' character describes output parameters. For example, the External_class entity (in the bottom left of the picture) contains the ^ext_class annotation. This means that the template will create an instance of the External_class entity, instance named ext_class.
- The blue arrows are used to bind input parameters to attributes of entities. The blue arrow in the bottom right of the picture means that the user needs to provide an input parameter called assigning_reference_data.ecl_id which will be used to set up the value of the id attribute of the instance called ext_class_lib.

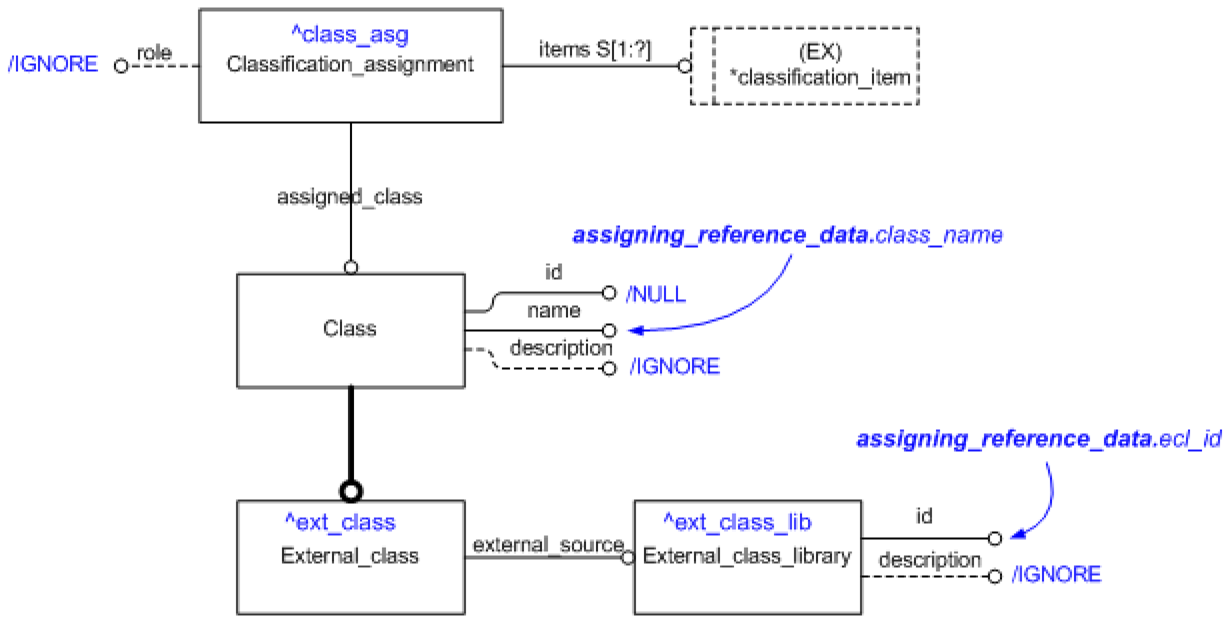


Figure 2-16 PLCS Assigning_reference_data template information model

More information on this extended EXPRESS-G notation can be found in the PLCS Technical Description online document (OASIS, 2010b). The only software supporting the extensions is a third-party freeware plug-in for an obsolete and no-longer-maintained version of a commercial diagramming software package.

The IP describes how to use and instantiate this information model. The IP uses a procedural language similar to that of ISO10303 SC4 reference paths and specifies:

- Input parameters of a template which correspond to the user input
- Reference parameters of a template which correspond to the instances created by the template
- Assignment of a value to an attribute of an entity
- Invocation of other templates
- Instantiation of entities
- The ordering of assignments, templates invocations and entity instantiations

Due to their nature, templates should not be seen as a mean of customizing an information model but as a way of customizing its use. Instead of instantiating an information model by instantiating its concepts, we create patterns of instantiations that correspond to recurrent domain-specific instantiations. One does not directly instantiate concepts anymore; one creates instances of templates, which will instantiate the information model using the templates definitions.

2.2.3.2 Summary

The use of DEX and templates combined with RDLs facilitates simplification and customization of AP239. Simplification is achieved through DEXs, which are domain-specific subsets of the AP239 and built using templates. Templates do not customize the information model itself, but the way it is used. They do this by providing patterns of instantiations to represent domain-specific information. The real customization is achieved by using an RDL, which defines the domain-specific concepts that will customize the information model. Regarding our classification of customization, DEXs, templates and RDLs compose a specialization of AP239 since the information model is not affected.

2.2.4 OAGIS

One notable customization effort has been done by the Open Application Group Inc (OAGi) for its Open Applications Group Integration Specification (OAGIS). This OAGIS standard facilitates interoperability between disparate business systems by defining a standardized formalism and architecture for representing the different messages that can be exchanged. In the context of the OAGIS, these messages are called Business Object Documents (BODs) and they are defined using XML Schemas. Figure 2-17 shows the different elements that compose a BOD.

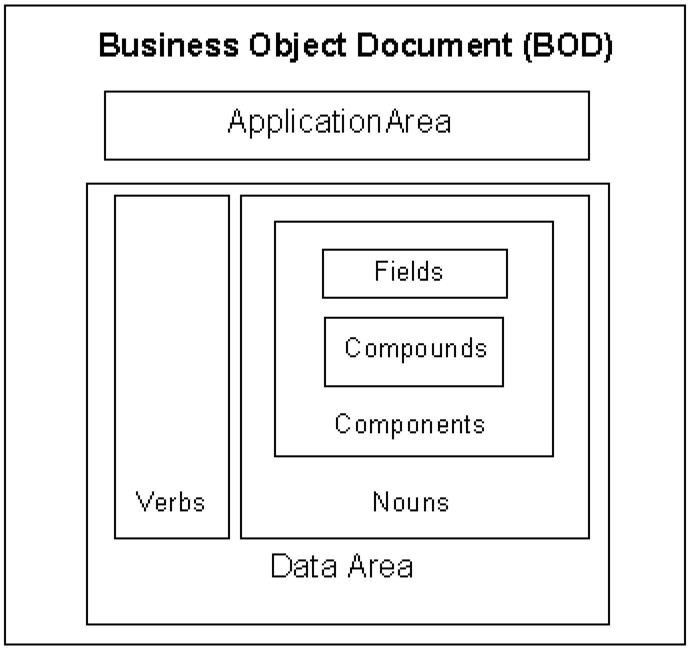


Figure 2-17 OAGIS BOD architecture

A BOD has two main areas: Application and Data. The Application Area contains all the information required to exchange the message such as creation date or the Globally Unique Identifier. The Data Area contains the content of the message, represented as a VerbNoun pair. The Verb identifies an action performed (Get, List...) on a Noun. The Noun identifies the business-specific information (PurchaseOrder, Shipment...) that is exchanged. Nouns are made of extensible building blocks called Components. Components contain Compounds and Fields. Compounds are basic building blocks (Quantity, Amount...) used by all BODs. They can be customized through contextual use (OrderedQuantity...) but cannot be extended with additional fields. Fields are the lowest elements defined in OAGIS and are fundamental elements (Description, Name...) used to create both Compounds and Components.

Figure 2-18 shows a high-level view of the ShowItemMaster BOD and its Application Area and Data Area.

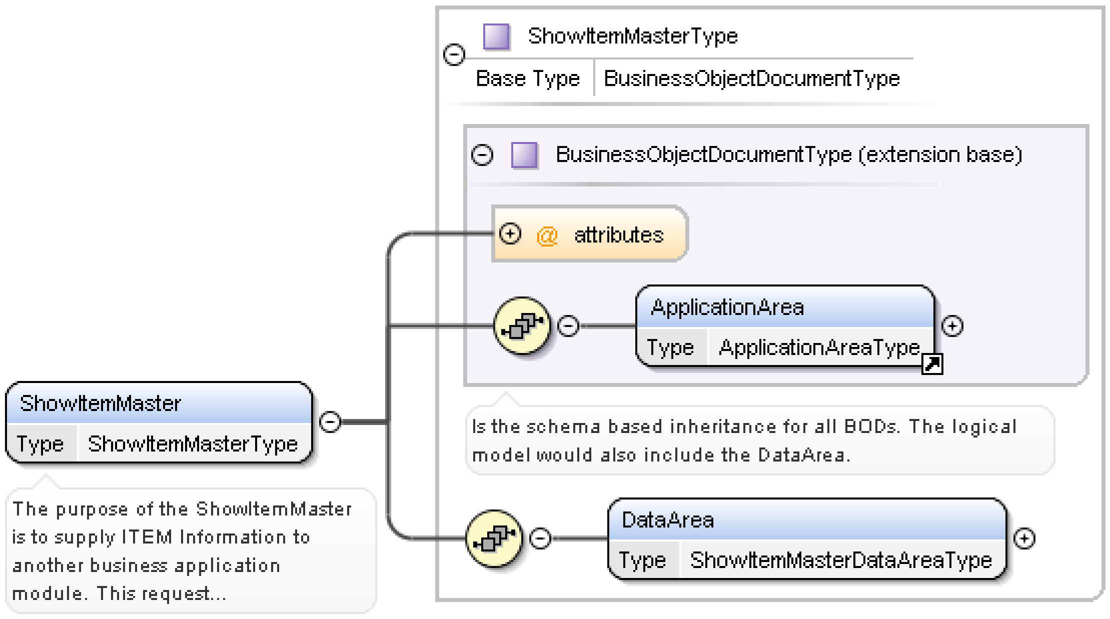

ShowItemMasterType
Base Type BusinessObjectDocumentType
BusinessObjectDocumentType (extension base)
@ attributes
ApplicationArea
Type ApplicationAreaType
ShowItemMaster
Type ShowItemMasterType
Is the schema based inheritance for all BODs. The logical model would also include the DataArea.
The purpose of the ShowItemMaster is to supply ITEM Information to another business application module. This request...
DataArea
Type ShowItemMasterDataAreaType

Figure 2-18 ShowItemMaster OAGIS BOD: ApplicationArea and DataArea

Figure 2-19 shows a high-level view of the Data Area for the ShowItemMaster BOD, composed of the Show verb and ItemMaster noun.

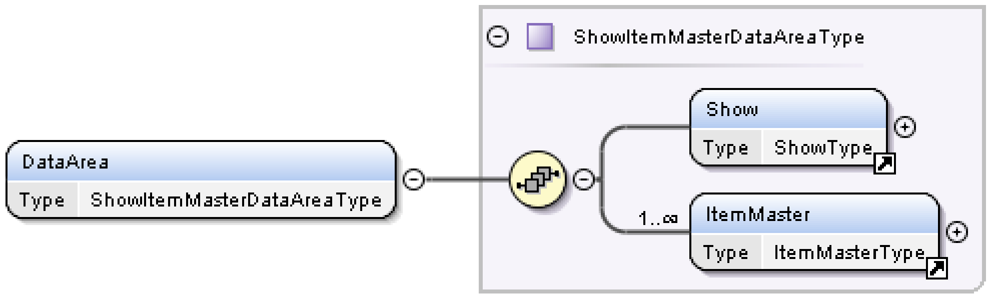


Figure 2-19 ShowItemMaster OAGIS BOD: verb and noun

OAGIS version 9.4.1 includes 13 verbs and its 79 nouns. Even so, it cannot identify and support all possible use cases. To address this, OAGIS can be extended in two different ways. First, the UserArea provides an extension mechanism that uses an optional UserArea field located at the end of each OAGIS component. It is described in more details below. Second, one can use Overlay extensions that allow users to either customize existing OAGIS components or create new ones from scratch.

2.2.4.1 UserArea

Despite the large number of nouns, components, compounds and fields, OAGIS cannot support every possible use case. As noted above, one way to support new use cases and related new information is to customize the UserArea. This type of customization is the easiest one because designing a UserArea extension does not require anything other than the information we want to include. Data added in the UserArea can be either 1) existing OAGIS elements that were not present in the initial element, or 2) any external source of information as long as it is XML. To illustrate the second situation suppose we want to customize the OAGIS element Facility (see Figure 2-20 for the schema of the element), which identifies a location, by adding data regarding the weather condition in the area of the facility.

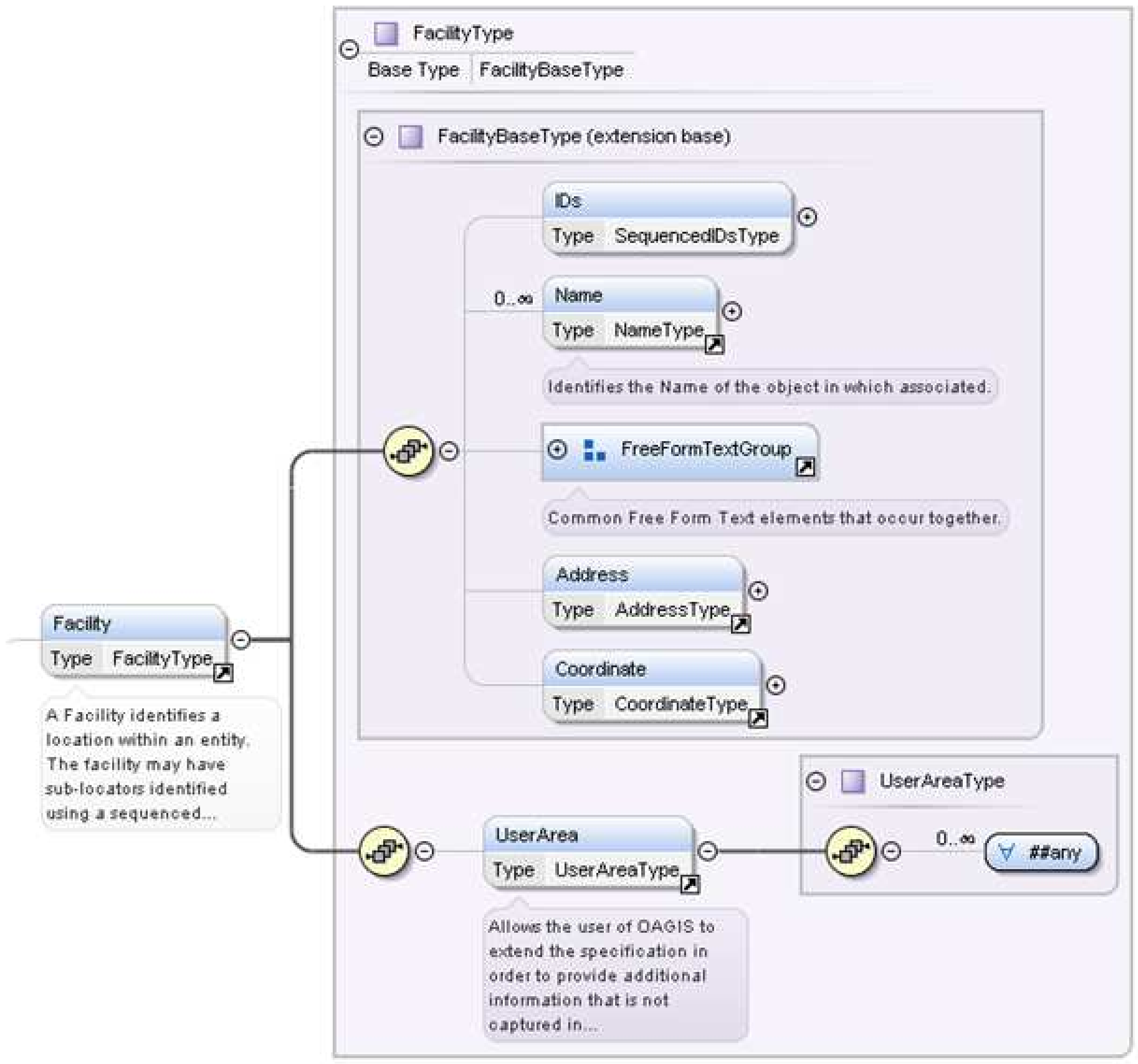
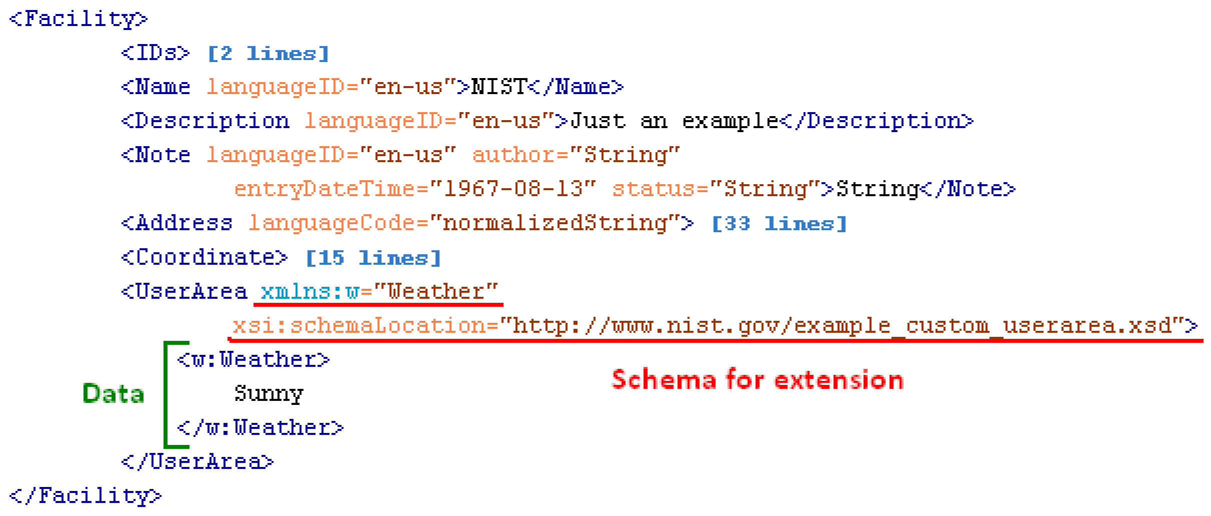

FacilityType
Base Type FacilityBaseType
FacilityBaseType (extension base)
IDs
Type SequencedIDsType
0..∞
Name
Type NameType
Identifies the Name of the object in which associated.
FreeFormTextGroup
Common Free Form Text elements that occur together.
Address
Type AddressType
Coordinate
Type CoordinateType
Facility
Type FacilityType
A Facility identifies a location within an entity. The facility may have sub-locators identified using a sequenced...
UserAreaType
UserArea
Type UserAreaType
0..∞
##any
Allows the user of OAGIS to extend the specification in order to provide additional information that is not captured in...

Figure 2-20 Initial OAGIS Facility

What we do first is to define an XML schema describing the structure of the information we need to represent. This schema defines an element called 'weather' whose value can be Sunny/Cold/Windy and looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="weather" type="weatherType"/>

    <xs:simpleType name="weatherType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Sunny"/>
            <xs:enumeration value="Cold"/>
            <xs:enumeration value="Windy"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

After we have designed an XML schema for the extension, we need to reference it in the UserArea element of the Facility and add the new data as content of the UserArea. This is shown below where we define a Facility called NIST located in a Sunny area.



```
<Facility>
        <IDs> [2 lines]
        <Name languageID="en-us">NIST</Name>
        <Description languageID="en-us">Just an example</Description>
        <Note languageID="en-us" author="String"
                entryDateTime="1967-08-13" status="String">String</Note>
        <Address languageCode="normalizedString"> [33 lines]
        <Coordinate> [15 lines]
        <UserArea xmlns:w="Weather"
                xsi:schemaLocation="http://www.nist.gov/example_custom_userarea.xsd">
            <w:Weather>
                Sunny
            </w:Weather>
        </UserArea>
</Facility>
```

Since the UserArea is generic and unstructured, OAGI recommends (OAGi, 2003) that it be used only by end users when they need to add a few additional fields not included in the OAGIS content. When adding a large number of additional and structured fields, OAGI provides a second mechanism of customization, the Overlay Extension.

2.2.4.2 Overlay Extension

Sometimes neither OAGIS components nor simple UserArea extensions address user requirements. To address these situations, OAGI provides an extension mechanism called overlay. This mechanism can be used to extend existing elements or create new ones. OAGIS elements that can be subject to this overlay mechanism are BODs, Nouns, Components, Fields and Compounds.

As an illustration of this overlay mechanism let us consider a company A that uses the ProcessInvoice BOD but needs to extend its DataArea that initially looks like Figure 2-21.

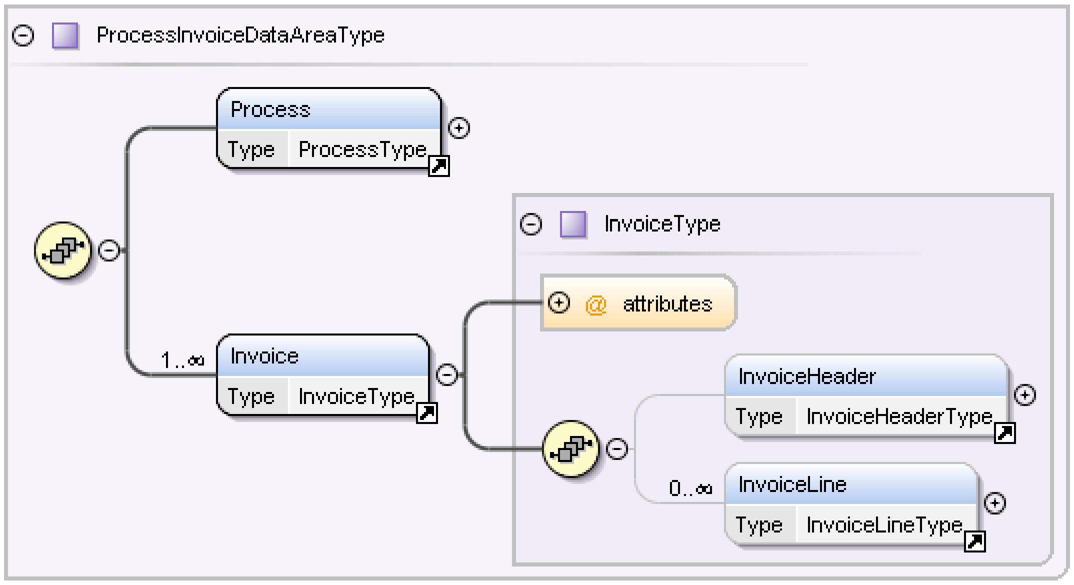


Figure 2-21 OAGIS BOD ProcessInvoice

This company A wants to add a new element corresponding to the sum of all the lines in the invoice. To do so we need to extend the Invoice element used in the BOD and append it to a new element that we call GrandTotal (cf. below XML code)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="Invoice">
        <xs:complexContent>
            <xs:extension base="oa:Invoice">
                <xs:sequence>
                    <xs:element ref="GrandTotal" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

In this situation we created a structurally identical BOD, since it keeps the Process verb and Invoice noun. Since the Invoice noun is extended with a new element, it is no longer the OAGIS Invoice noun. Hence, its namespace must change to trace its ownership to company A's Invoice noun. Consequently, the namespace of the BOD has to change as well. This operation is not required for the Process verb since we keep using the original OAGIS verb. After extension, the DataArea of our new company A ProcessInvoice BOD looks like Figure 2-22.

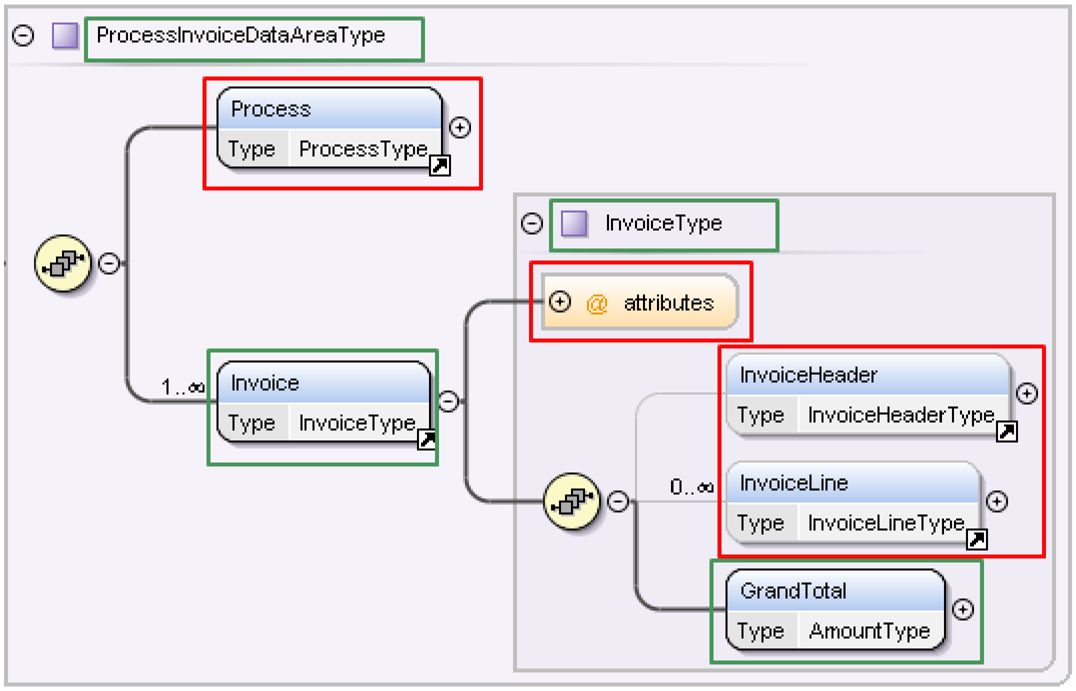


Figure 2-22 Extended OAGIS BOD ProcessInvoice

In the above figure, green elements are elements whose namespace is owned by the company A; red elements are elements whose namespace is still owned by OAGIS. Green elements are components of the extension.

In this example we have seen how to extend a BOD by extending a noun. The mechanism to extend other components is very similar. As noted above, the overlay mechanism also allows users to create new elements when none of the existing ones satisfy the user requirements. Since the procedure is really similar to extending existing components, we do not explain it here. More information on this procedure can be found in (OAGi, 2003).

2.2.4.3 Summary

We have seen in this section the two mechanisms provided by OAGI to customize OAGIS. First we introduced the UserArea field in which any type of XML information can be carried. While this mechanism has no impact on the data schema, it considers the data in the UserArea as plain data with no interest in its structure. When the information needs to be structured, OAGIS provides an overlay mechanism that can be used either to extend existing OAGIS elements or create new ones. While this mechanism offers more flexibility and power in structuring the information, it has a direct impact on the underlying schema.

2.2.5 Implementations

While all the previously described frameworks provide mechanisms for dynamic customization, their use and implementations have not all met the same success.

The Core Product Model has been extended by different research project to represent product centric information that was not initially in its conceptual model, and proves the efficiency of the CPM extension mechanism. Such extensions include: 1) the Design-Analysis Integration Model (DAIM) (Fenves, Choi, Gurumoorthy, Mocko, & Sriram, 2003), 2) the Product Family Evolution Model (PFEM) (Wang, Fenves, Rachuri, & Sriram, 2003) to support representation of families of product and 3) an extension for reverse engineering that enables generating Beginning of Life data from Middle of Life data (Troussier, Bricogne, Belkadi, Durupt, & Ducellier, 2010) 4) representation of heterogeneous material properties (Biswas, Fenves, Shapiro, & Sriram, 2007).

STEP is, as now, the biggest standard for product data. It is involved in a tremendous number of projects where interoperability and sustainability of data are critical aspects. Unfortunately, STEP seems to be underused by its implementations. While ISO 10303-203 & 10303-214 (ISO, 1994c, 2010) are the most implemented STEP APs and respond to a large variety of needs (requirements, geometry, assembly structure, Geometric Dimensioning & Tolerancing …), they are mainly used/implemented to exchange geometry. One sign of a possible use of the UDA mechanism is the release of recommended practices by the CAx-IF, an industrial forum involving some of the biggest ISO 10303-203 & 10303-214 implementers. Another STEP standard with available implementations is ISO 10303-233, which has been developed to support exchange of system engineering data, is often a choice for exchanging SysML models. In this process, an ISO 10303-233/SysML mapping[3] has been developed, where the STEP external classification is widely used. Regarding ISO 10303-209/10303-210 (ISO, 2001, 2011), we are not aware of any use of the customization mechanisms. ISO 10303-239, because of its complexity, has not been implemented as a whole but on demand following the OASIS DEX mechanism. A number of DEXs have been developed and make an intensive usage of the STEP external classification mechanism combined with external libraries developed in OWL.

Unlike OASIS, OAGi does not provide a common and publicly available repository for sharing business specific implementation of customization. The lack of such a resource makes it difficult to discover and evaluate customizations.

2.3 Conclusion

In this chapter, we defined the notion of product life-cycle management, which is a way of managing the different components involved in the product life cycle as well as enabling collaboration and exchange of information. Since information is digital, collaboration is based on data exchange and controlled by software applications. We introduced information models as one way to capture and exchange this digital information in a formal manner. We have seen the complexity of building such models to exchange data in large heterogeneous networks. We

[3] The SysML and AP233 mapping is described at http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-ap233:mapping_between_sysml_and_ap233

argued that information standards can play an important role, as common platform, in reducing complexity and facilitating collaboration. Unfortunately, due to the dynamic aspect of PLM and the static nature of these standards, information standards are not ready to support such collaboration. Methods are required, therefore, to extend these standards as needed. This chapter described several examples of such methods, their origins, and their uses. In the next chapter, we describe the strengths and weaknesses of these methods when they are applied in the context of PLM.

3 Evaluation of customizable frameworks

3.1 Introduction

In the previous chapter we discussed four frameworks that provide mechanisms for dynamic customization of information models. These frameworks were chosen because of the diversity of those mechanisms and the domains in which they can be applied. Diversity in mechanisms makes it a challenge to identify which of the four is the ideal framework for PLM, if one exists. Also, since these frameworks solve customization problems for specific domains, we must determine whether they can be applied to customization problems in another domain, PLM.

To address both of these issues, we evaluate the different frameworks based on the technical requirements associated with PLM. We first define those requirements, and then derive assessment criteria.   These criteria will help us to determine whether any of the frameworks is adequate for our purpose or if a new one is needed.

3.2 Requirements and assessment criteria

In this section, we identify the set of PLM-related requirements and assessment criteria that we will use to evaluate the frameworks described in the preceding chapter.

3.2.1 Customization

PLM collaboration is based on digital data exchange. We have seen that the heterogeneity of the environment is an important factor in determining the complexity of any information exchange. Heterogeneity means that the different applications involved in that exchange have diverse representations of the information exchanged - similar to people talking in different languages. Its impacts are supposed to be minimized by the use of international standards. Unfortunately, for large, dynamic domains such as PLM, it is hard to get complete agreement on what needs to be in the standards. The ability, therefore, to customize the information models in standards enables them to serve as a foundation that everyone build upon to meet specific needs. However, since we are trying to reach a certain level of homogeneity of information representation, that customization must be done without destroying that foundation - without having to rewrite the standard. This means that the result of any customization mechanism should not impact the data schemas in the standard.   In the preceding chapter, we identified two such mechanisms: extension and specialization.

Extension provides a means of introducing new concepts. Extensions not only widen the scope, they can also refine it. Extension is an easy form of customization since (1) it involves only new concepts and relationships and (2) it can be done with any modeling language. Conceptually, if m1 is an initial information model, m2 is an extension of m1 if there are at least 2 concepts, C1 in m1 and C2 in m2, and there is a relationship r1 between them. While it is easy to determine if r1 exists, it is not always easy to determine if the representation of r1 impacts the original data schemas in m1. Let us consider the UML class diagram (Figure 3-1) as our initial information model.

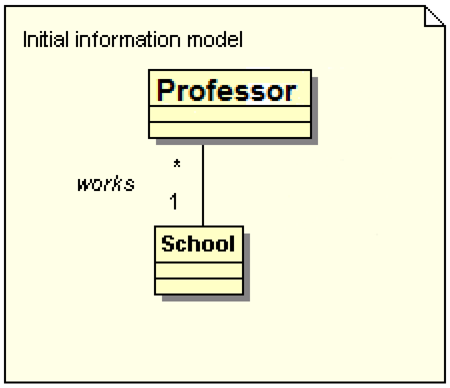


Figure 3-1 Professor-School UML class diagram

One underlying XML schema could look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Professor" type="ProfessorType"/>

    <xs:complexType name="ProfessorType">
        <xs:sequence>
            <xs:element ref="School" default="1”/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="School" type="SchoolType"/>

    <xs:complexType name="SchoolType">
        <xs:sequence>
            <xs:element         ref="Professor"         minOccurs="1"
maxOccurs=”unbounded”/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Now suppose we want to extend this initial information model to support information about the PhD students that a professor can supervise. The relationships we want to represent are (1) a professor can supervise many students simultaneously, but a student can be supervised by only one professor; and (2) a student has to write one thesis and each thesis can be written by only one student. We can capture these relationships in a UML class diagram like in Figure 3-2.

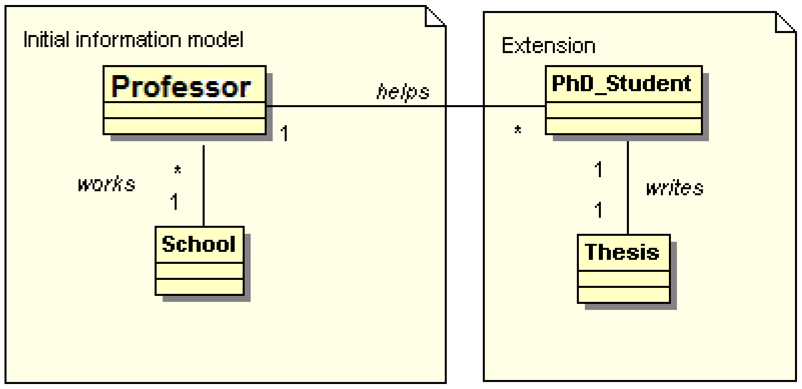


Figure 3-2 Extended Professor-School UML class diagram

From this information model, we can derive the following XML schema. Clearly, the new XML schema is substantially different than the previous one.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Teacher -->
    <xs:element name="Teacher" type="ProfessorType"/>
    <xs:complexType name="ProfessorType">
        <xs:sequence>
            <xs:element ref="School" default="1" />
        </xs:sequence>
    </xs:complexType>
    <!-- School -->
    <xs:element name="School" type="SchoolType"/>
    <xs:complexType name="SchoolType">
        <xs:sequence>
            <xs:element ref="Professor" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <!-- Student -->
    <xs:element name="Student" type="StudentType"/>
    <xs:complexType name="StudentType">
```

```
        <xs:sequence>
            <xs:element ref="Professor" default="1"/>
            <xs:element ref="Thesis" default="1"/>
        </xs:sequence>
    </xs:complexType>
    <!-- Thesis -->
    <xs:element name="Thesis" type="ThesisType"/>
    <xs:complexType name="ThesisType">
        <xs:sequence>
            <xs:element ref="Student" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Using this new XML schema for data exchange requires modifications to the software involved in the data exchange. This adds a layer of complexity and increases the cost of implementation.

The other customization mechanism is called specialization. Specialization refines the semantics of existing concepts in the original information model based on an external source of classification. Chapter 2 described two different implementations of this mechanism. The first uses an attribute to save the type of the concept - see 2.2.1.2 for an example. This attribute is meant to contain a string value that identifies the type of the concept. The second is a more rigorous and semantically meaningful implementation because a dedicated structure is designed to support the specialization - see 2.2.2.2 for an example.

Specialization is a more complex mechanism than extension because it must be accounted for during the initial design of the information model. It is also less expressive since it defines only sub-concepts. What can be seen as a lack of expressiveness, however, might also be considered as a first step toward a control mechanism for extensions. Moreover, depending on the technology used for defining the specialization, different levels of expressiveness are offered to the users. Additionally, using specialization does not impact the underlying data schema; no modifications to existing software, therefore, are needed.

The relative merits of these two mechanisms depend on the context in which the customizations are needed. For PLM, a number of expensive software applications use and modify information; consequently, frequent customizations are required. Extensive and recurrent modifications to that software are usually not possible. In this context, then, ease of implementation ranks higher than level of expressiveness. Therefore, in general, we consider specialization a better customization mechanism than extension for PLM. So, the first requirement is that the framework must have a mechanism for specialization.

3.2.2 Semantics

In the previous section we argued that ease of implementation is a more important factor than expressiveness for PLM. But, regardless of the expressiveness, we must ensure that the semantics of any new concepts are represented in a way that is computer interpretable. This is necessary for two reasons. First, the PLM environment is quickly becoming a digital, collaborative, and global environment. This means that the software applications in that environment must be able to exchange and process all necessary information electronically, seamlessly, correctly, and, with the minimal possible human intervention. Second, any human intervention will be costly, time consuming, and, prone to errors. The only way to make this happen is for all of those applications to have a consistent view of that information. Formally defined semantics is the best way to provide such a consistent view. It had an added advantage because it provides a basis for automated consistency checking at two levels: the conceptual level and the data level.

Conceptual-level consistency checking verifies consistency between concepts. This kind of checking is performed to ensure that newly added concepts are meaningful and can be instantiated. An example of the value of this type of checking is given below.

- Car and Boat are 2 concepts
- Car and Boat are disjoints
- Boat is a sub-concept of Car

To check the consistency of this system we first create the corresponding ontology with the Protégé editor:

- Car and Boat are 2 OWL class
- Car and Boat are disjoint classes
- Boat is a subclass of Car

Second, we use the reasoner FaCT++ (Manchester, 2004) to do the consistancy check that fails. That is, the OWL class Boat is inferred as subclass of owl:Nothing, which represents an empty set (W3C, 2004). Being inferred as a subclass of an empty set means that no instantiation of this class is possible.

Even though the inference failed, the result is useful because it identifies concepts that cannot be instantiated during customization design. This reduces development time by detecting errors before implementation.

Data-level consistency checks ensure that data exchanges are meaningful. These checks provide a way of identifying errors automatically before data are exchanged. Such mistakes mostly happen when data are instantiated in a way they should not be.

Let us consider the following simplistic information model:

- Car is a Vehicle
- Boat is a Vehicle
- Car and Boat are disjoint concepts

We then instantiate this information model and create the following vehicles:

- V1 is a Car
- V2 is a Boat
- V3 is a Car and a Boat

It is easy to see that the instantiation V3 is an inconsistency because Car and Boat are disjoint concepts. Detecting such inconsistencies is not always easy. With an ontology and a reasoner, however, these types of inconsistencies can be detected automatically. But, this kind of checking can only be done after the customization has been designed and implemented. A more effective approach would be to control somehow the customization and the new concepts it introduces. We discuss an approach to doing this in the next section.

3.2.3 Controlled customization

To limit the amount of consistency checking that must be done after design and implementation, we propose an approach called controlled customization. The goal of this approach is to limit the set of customizable concepts from the initial information model in such a way that number of inconsistencies is minimized. Since PLM is cross-domain, its generic initial information model is very large and never used as a whole. Consequently, customization is a frequent, complex process. To reach our goal, we define a subset of the original concepts that can be customized. To define this subset, we need to understand the information needs from the downstream processes that will use the customization.

For illustration, let us consider the information model in Figure 3-3 as initial one:

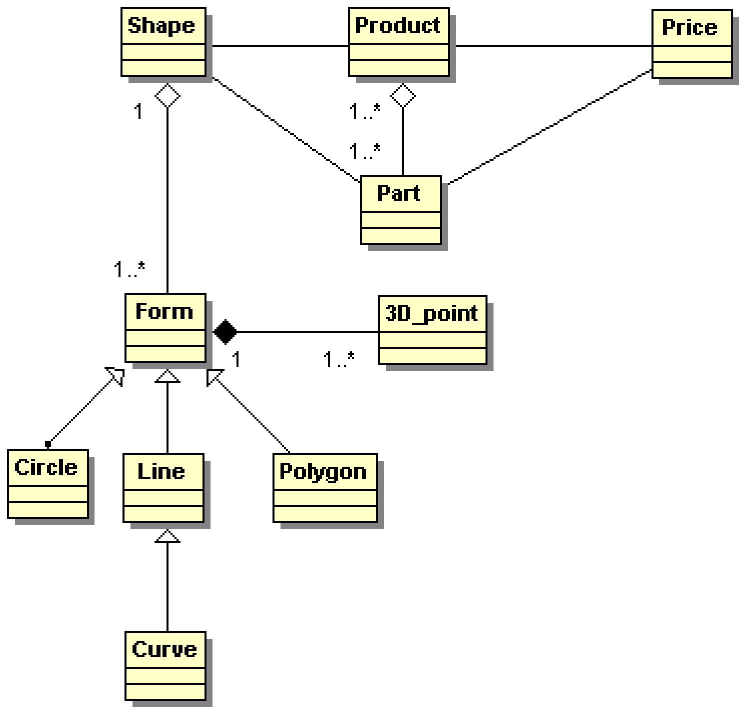


Figure 3-3 Controlled customization information model

In this model, we represent a Product as a set of Parts. A Product and a Part have a Shape that is described by a set of Forms where a Form can be a Line, a Curve, a Polygon or a Circle. We can control the number and kinds of customizations by limiting them to only additional Forms. So, for example, creating a customization to introduce additional Forms such as Square and Rectangle would be permissible. Creating a customization to introduce Shapes, other than Form, would not be allowed.

So, one of our requirement is that the framework must allow, and have a mechanism for, controlled customizations.

3.2.4 Business objects

Previously, we focused on two customization mechanisms, their validation using high-level semantics, and a possible control approach to limit their inconsistencies. We indicated that specializations are easier for implementers than extensions because the software requires no major modifications to read and write the data. But, we did not consider the impacts on the end users, who must be able to understand both the initial information model and all of its customizations. The initial model should include enough generic information so that customizations will provide most of the information needed to drive all product life-cycle processes. Since users are typically domain experts in only one of those processes, the terminology they use is usually process dependent, which means that the terminology will differ slightly from user to user; yet, the way that terminology is represented should stay the same from process to process. Furthermore, that representation is determined by the initial information model. Thus there is often a need for a mapping between the domain-specific terminology in the customized model and the generic terminology in the initial information model.

To illustrate this, consider the generic information model in Figure 3-4 as the initial one.

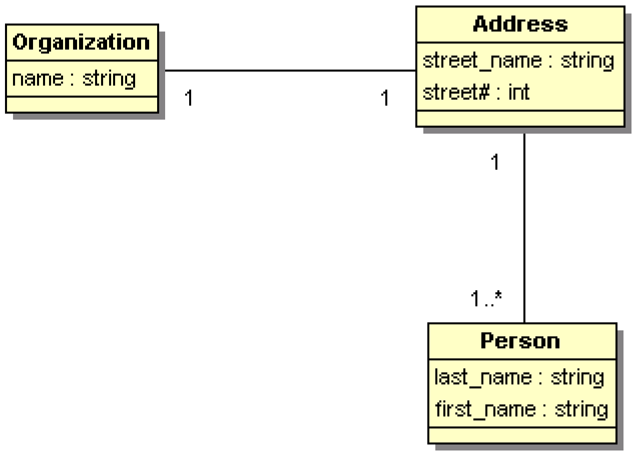


Figure 3-4 Generic information model for business objects

This model represents the concepts Organization, Address and Person. The relations between them are as follows: an Organization has a unique Address and so does a Person. These concepts are generic. Now consider a customization designed to support logistics process called shipment. In this process, an expediter ships a product somewhere and this somewhere is represented by a shipping address and a recipient. The customization of the original information model is as follows: the organization is the expediter, the person is the recipient, and the address is the shipping address. The underlined words are business-specific terms needed for the customization. The associated information model, which is shown in Figure 3-5, is called a business objects model (Barnard-Feeney & Hunten, 2011).

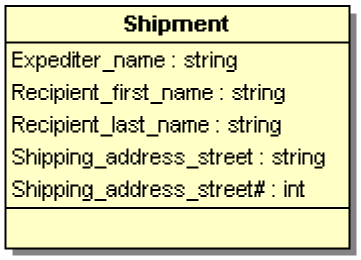

Shipment
Expediter_name : string
Recipient_first_name : string
Recipient_last_name : string
Shipping_address_street : string
Shipping_address_street# : int

Figure 3-5 Shipment business object

A business objects model not only uses domain-specific vocabulary but it also hides the complexity of the original generic information model. In other words, it is not necessary to replicate all of the relationships from the generic model in customizations of that model. At the implementation level, when the Shipment model is instantiated, the generic concepts - Organization, Person and Address - and their relationships will be instantiated automatically. These instantiations are made possible with a mapping (see Figure 3-6) between the business objects model and the initial information model. This mapping needs to be defined formally so it can be computer interpretable.

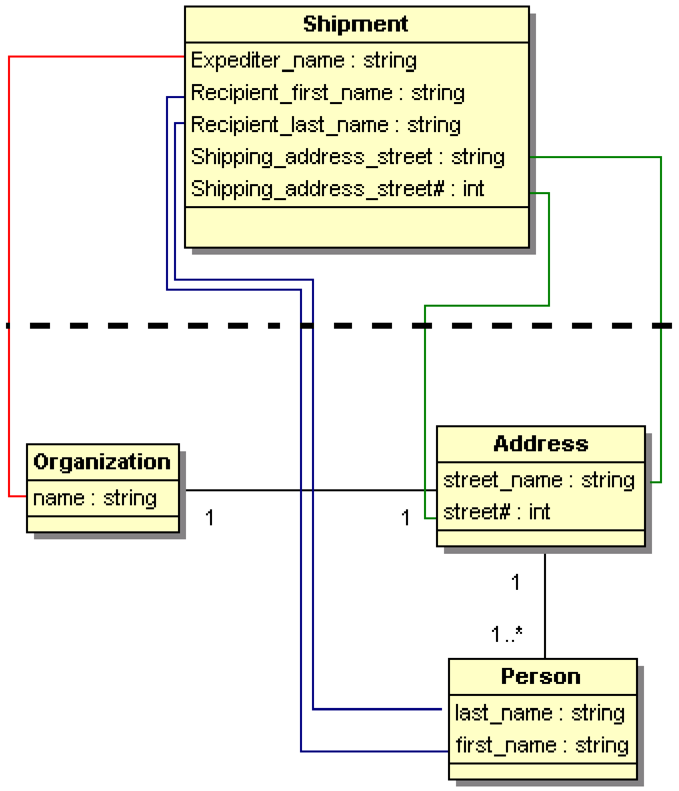


Figure 3-6 Mapping information model - business object model

In (Barnard-Feeney & Hunten, 2011), the authors discuss the role of a business objects model in reducing the complexity of Application Programming Interfaces (APIs), which are used commonly to manipulate information. Complexity of the API is reduced because it becomes more intuitive and less confusing for its users. Implementing an API based on the initial information model requires CRUD (Creation, Retrieval, Update and Deletion) methods for the 3 concepts: Organization, Address, and Person. Implementing an API based on the business objects mode, however, requires methods only for the business object Shipment. This results in an easier-to-use API.

Business objects show how meaningful aggregations of concepts can be. These aggregations of concepts are done to ease domain-specific information needs. They can be structurally similar but semantically different. This difference comes from the context of operation, which defines the semantics of the aggregation. The same generic trio (Organization, Address, Person) can be used to represent several semantically different concepts with no changes to the structure of the initial model. In these cases, consistency checking of the generic concepts, as defined in the previous section, is not enough anymore. There is now a need of additional consistency checking related to the domain-specific concepts.

3.2.5 Business rules

In the context of data exchange we have seen that data quality is important. We have shown how formally defining semantics helps to identify inconsistencies from a generic perspective. But PLM involves numerous domain-specific, context-dependent perspectives. Checking the data associated with these context-dependent perspectives cannot be done using approaches described above. Those approaches were based on generic semantics. We propose another approach based on context-dependent rules, commonly known as business rules.

Consider the following example. From the information model in Figure 3-7, we see that an Operator can perform many Operations and a Machine can be involved in many Operations.

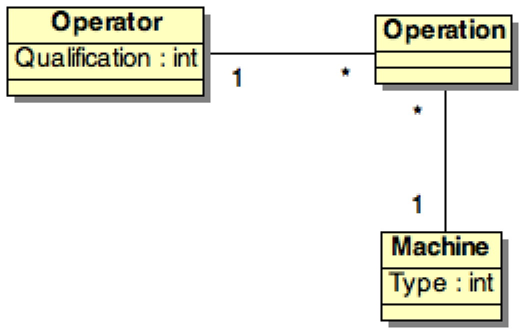


Figure 3-7 information model for business rules

After instantiating this information model for two specific operations - painting and cleaning - we come up with the UML object diagram in Figure 3-8.

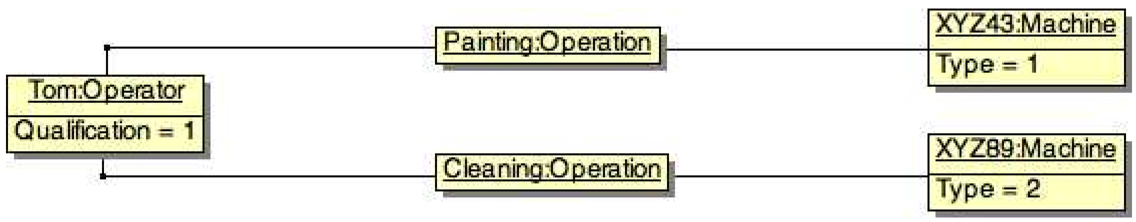


Figure 3-8 instantiation model

These instances are semantically correct and consistent with the information model defined in Figure 3-7. Operator Tom is involved in two Operations and each operation involves one machine. Now let us consider that a company A has a special policy (a rule) that says “an

Operator can only use a Machine, if the Operator has a valid Qualification on that machine". Unless Tom has been qualified to use machines XYZ43 and XYZ89, these same instances in Figure 3-8 will not be valid against this business- specific rule of company A.

Since business rules play a major role in the validity of PLM data, we must be able to express them formally so that automatic validation can be performed. So, another requirement is that the framework must provide the capability to enhance customizations with business rules.

3.2.6 Summary

In this section we presented different requirements for evaluation of customization mechanisms. Those criteria include mechanisms for a specializations, formal semantics, controlled customizations, and, business rules.

3.3 Evaluation

In the previous section we have defined evaluation criteria, in the form of requirements, regarding customization frameworks. This section reports the evaluation of the different frameworks described in Chapter 2.

3.3.1 Specialization

The first criterion we developed relates to the customization mechanism itself. We have identified and defined two mechanisms: extension and specialization. Since the ideal PLM framework should be a driver for efficient collaboration within the product life cycle, specialization fits the need. We now evaluate each framework based on its ability to enable customization through specialization. To do so we will represent, when possible, a car assembly composed of four (4) wheels and one (1) engine, as a specialization.

NIST CPM

In section 2.2.1 we presented the NIST Core Product Model (CPM) as a framework that provides both specialization and extension. An example of CPM specialization is the NIST Design Repository (Szykman & Sriram, 2006), which provides mechanisms for editing and browsing product models stored in any repository. An example of extension is the Open Assembly Model (OAM), which describes the nature and information requirements for part features and assembly relationships. In section 2.2.1.2, Figure 2-10 we have demonstrated how to use the CPM specialization mechanism for representing a car assembly as a composition of wheels and engine. Since it provides both mechanisms, the NIST CPM meets the expectation regarding this criterion.

ISO STEP

In section 2.2.2 we presented the ISO 10303 standard, informally known as STEP. Customization in STEP is based on a two mechanisms: external classification and User Defined Attributes (UDA). The first mechanism allows classification of existing concepts using an external source. The UDA mechanism allows domain-specific attributes to be added to certain concepts. Both mechanisms, while different in scope, are identical to specialization.

In section 2.2.2.2, Figure 2-13 we have seen how to specialize a product as a car. The car assembly itself is represented in the following Part21 code. In this code we define a Product (#1), specialized as a Car (#15) from an external source (#21). We also specialize a Product (#13) as an Engine (#19), and a Product (#14) as a Wheel (#20). The car is an assembly of the engine (#4) and four (4) wheels (#5, #6, #7, #8).

```
DATA;
#15 = EXTERNALLY_DEFINED_CLASS('Car',$,$,#21);
#14 = PRODUCT($,'WheelModel',$,());
#13 = PRODUCT($,'EngineModel',$,());
#12 = PRODUCT_DEFINITION_FORMATION($,$,#14);
#11 = PRODUCT_DEFINITION_FORMATION($,$,#13);
#10 = PRODUCT_DEFINITION($,$,#12,$);
#9 = PRODUCT_DEFINITION($,$,#11,$);
#8 = NEXT_ASSEMBLY_USAGE_OCCURRENCE($,$,'back right wheel',#3,#10,$);
#7 = NEXT_ASSEMBLY_USAGE_OCCURRENCE($,$,'back left wheel',#3,#10,$);
#6 = NEXT_ASSEMBLY_USAGE_OCCURRENCE($,$,'front right wheel',#3,#10,$);
#5 = NEXT_ASSEMBLY_USAGE_OCCURRENCE($,$,'front left wheel',#3,#10,$);
#4 = NEXT_ASSEMBLY_USAGE_OCCURRENCE($,$,'engine',#3,#9,$);
#3 = PRODUCT_DEFINITION($,$,#2,$);
#2 = PRODUCT_DEFINITION_FORMATION($,$,#1);
#1 = PRODUCT($,'CarModel',$,());
#21 = EXTERNAL_CLASS_LIBRARY(IDENTIFIER('http://my_vocabulary/source.owl/'));
#20 = EXTERNALLY_DEFINED_CLASS('Wheel',$,$,#21);
#19 = EXTERNALLY_DEFINED_CLASS('Engine',$,$,#21);
#18 = APPLIED_CLASSIFICATION_ASSIGNMENT(#20,$,(#14));
#17 = APPLIED_CLASSIFICATION_ASSIGNMENT(#19,$,(#13));
#16 = APPLIED_CLASSIFICATION_ASSIGNMENT(#15,$,(#1));
ENDSEC;
```

OASIS DEX

In section 2.2.3, we presented the OASIS recommendations and framework for implementing ISO 10303-239, which provides a different approach for implementing both external classification and UDA. That approach uses OWL to represent formally the external source of classification, called the Reference Data Library (RDL). Since the mechanisms are identical, OASIS DEX does meet the expectation regarding the customization criterion.

OAGi specification

In section 2.2.4, we presented the framework developed by the Open Application Group Inc (OAGi), the Open Applications Group Integration Specification (OAGIS). This framework provides two customization mechanisms: UserArea and Overlay. As shown, in Section 2.2.4.1 and 2.2.4.2, these two mechanisms are extension. Hence, they do not meet our specialization requirement.

3.3.2 Semantics

The second requirement is related to the formal semantics of new concepts. Recall that a formal semantics is important for information processing, understanding, and consistency checking. Not only the framework must provide for new concepts to be formally defined, the level of expressiveness matters.

NIST CPM

The NIST CPM provides two customization mechanisms: specialization and extension. Based on the preceding discussion, we need to consider specialization only. In the CPM specialization mechanism, a special string value is given to an attribute of a class. This string value acts as a classifier. It has no formal semantics, therefore, direct computer interpretation of the attribute value cannot be done. Automatic processing is then reduced to string comparison; automated consistency checking is not possible. So, NIST CPM does not meet this requirement.

ISO STEP

ISO 10303 provides two customization mechanisms: external classification and User Defined Attributes. We consider them separately since they deal with formal semantics differently.

When implementing the external classification mechanism, ISO does not provide recommendations on the format of the external source and its concepts. The semantics of the external source is tied to the technology used to express it. Therefore, it is impossible for us to identify a priori whether or not any particular external classification meets our semantics requirement.

The User Defined Attributes mechanism is very similar to the NIST CPM specialization since the semantics of a UDA resides in its name. Moreover ISO 10303-41 provides specific information structures for representing values of attributes depending on their type. Those structures include:

- string
- integer
- real
- boolean
- measure (combination of a unit name, measure value, measure unit)

These specific structures have almost no semantic expressiveness. Hence, they cannot be processed automatically or understood by computer software applications. We conclude that the ISO framework does not meet the formal semantics requirement.

OASIS DEX

OASIS provides an external classification mechanism that differs from the STEP approach in one important way: it uses OWL for representing the external source of classification, called the Reference Data Library (RDL). Although OWL is a strongly expressive language, most RDLs are built as taxonomies. Consequently, the semantics is limited to hierarchical relationships between the different concepts in the taxonomy. Nevertheless, when building an RDL, one is free to add as much semantics and OWL constructs as desired. So, an OASIS DEX meets the formal semantics requirement.

OAGi specification

The OAGi specification provides two customization mechanisms, which are based on XML schemas. XML schemas are similar to UML they do not have a formal semantics (Arenas & Libkin, 2005). Therefore, the OAGi framework, in its form, does not satisfy our semantics requirement.

3.3.3 Controlled customization

The third requirement is that the framework must have a mechanism to control the customization by limiting the set of customizable concepts.

NIST CPM

The two CPM customization mechanisms do not control customization the same way. So, we address them separately.

The CPM extension mechanism does not provide any control on customization. Furthermore, CPM authors do not provide recommendations on how to write extensions, consequently, any class from the CPM conceptual model can be subject to extension.

The CPM specialization mechanism does provide a control mechanism: to specialize a concept, the attribute type is needed(cf. section 2.2.1.2). Unfortunately, this attribute is inherited from the **CoreProductModel** class that is the root of the CPM conceptual model, meaning that any class can be specialized.

Thus, the NIST CPM framework does not meet the customization control requirement.

ISO STEP

The STEP external classification mechanism is based on a special information structure (cf. section 2.2.2.2). In this structure, the EXPRESS entity **Classification_assignment** is a connector between a classifier and the concept to classify. **Classification_assignment** can be connected only to certain types of EXPRESS entities. Only the concepts associated with these entities, therefore, can be specialized. As explained in section 2.2.2.1 the UDA mechanism works in a similar fashion, since certain concepts/EXPRESS entities can benefit from it. Therefore, the ISO STEP framework does meet this customization control requirement.

OASIS DEX

OASIS DEX is an implementation framework for ISO 10303-239. As such, its specialization mechanism is similar to ISO STEP. OASIS also provides rigorous control over the external classification, which is enforced by a standard Reference Data Library. This RDL is composed of the only concepts that can be specialized. New RDLs should be based on this standard RDL and only specialize it. So, OASIS DEX meets the controlled customization requirement.

OAGi specification

The UserArea (cf Section 2.2.4.1) field is embedded within all OAGIS Components. Since a Component is composed of Fields and, possibly other, Components, it cannot be customized. OAGIS Nouns are composed of Components, so when extending a Component, one also extends the associated Nouns. Since Nouns are the building blocks of an OAGIS BOD, the extension of a Noun extends the BOD itself. In summary, the UserArea can only be used to extend Components, Nouns and BODs, while Verbs, Compounds and Fields cannot be extended. So, the UserArea mechanism meets the customization requirement.

A similar situation happens with the Overlay extension since BODs, Nouns and Components can be extended. But the Overlay extension also enables creation of new BODs, Nouns and Components. So, the Overlay mechanism does not meet this requirement.

3.3.4 Business objects

The fourth requirement is that the frameworks must provide a mechanism for representing business objects explicitly in a formal way, where business objects are aggregation of lower level data.

NIST CPM

The NIST CPM doesn't provide a mechanism for explicitly representing business objects. It is possible to do so, but the representation has two problems: it is not explicit and it is not at the same level as the rest of the data. Considering the example introduced in Section 3.2.4 of this chapter, representing a Shipment using the NIST CPM extension mechanism would result in Figure 3-9.

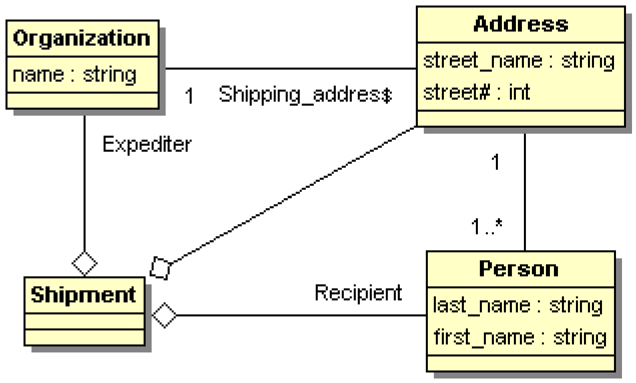


Figure 3-9 CPM business object

The class Shipment is an aggregation of Organization, Address and Person as it is supposed to be. Unfortunately, there is no way of recognizing Shipment as a business object. Moreover, with this representation, a business object is defined at the same level as the low level data that composes it. So, NIST CPM does not meet the business object representation requirement.

ISO STEP

In its current version, STEP does not provide a mechanism for explicitly representing business objects. Business objects can be represented using the right set of instantiations but they do not explicitly appear as such. Recently, (Barnard-Feeney & Hunten, 2011) introduced a possible approach for representing a business objects model for STEP in a formal and explicit fashion. So, ISO STEP might meet the business object representation requirement in the future.

OASIS DEX

OASIS provides a mechanism called templates for explicitly representing business objects, which are partially formalized (cf. section 2.2.3.1). Templates were initially designed to represent patterns of instantiations and to enable reusability. So, OASIS DEX does meet the business object representation requirement.

OAGi specification

OAGIS has a native mechanism for representing business objects, the OAGIS BODs. They are composed of Verbs and Nouns where Nouns represent business objects and Verbs are actions performed on them. So, OAGi meets the business object representation requirement.

3.3.5 Business rules

The last requirement is that the framework must provide recommendations for applying business rules to both the initial information model and its customization(s). When the framework does, and provides specialization, we will provide an example.

NIST CPM

The NIST CPM is based on UML and its authors also provide an implementation model in XML (CPXS). CPM does not provide recommendations on business rules or how to apply them. Nevertheless, options are available for users, the Object Management Group has a standard known as Object Constraint Language (OCL), which is “a formal language used to describe expressions on UML models” (OMG, 2010). OCL is currently being used as a mechanism for expressing business rules to UML models (Korthaus, 1998)(van Engers, Gerrits, Boekenoogen, Glassée, & Kordelaar, 2001). The ISO standard *19757-3:2006 Information technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-based validation -- Schematron* (ISO, 2006) is another option for expressing business rules and is described as “a language for making assertions about patterns found in XML documents”. Moreover many free, open-source and commercial implementations are available. Unfortunately the lack of official recommendations from the CPM authors is a drawback to this framework. So, at best, NIST CPM partially meets the business rules requirement.

Based on the UML model in Figure 2-10 we could write a rule to validate that: 1) an Artifact specialized as a car has 5 assembly components, including four (4) wheels and one (1) engine, 2) an Artifact is specialized as a wheel if its attribute type is Wheel and is part of the assembly of only one car, 3) an Artifact is specialized as an engine if its attribute type is Engine and is part of the assembly of only one car. The 3 rules can be represented in OCL as follows:

```
context Artifact
inv isCar: if (self.type = 'Car')
        then self.subartifacts-> size = 5
        and self.subartifacts->select(a | a.type = 'Wheel')->size = 4
        and self.subartifacts->select(a | a.type = 'Engine')->size = 1
endif

inv isWheel: if (self.type = 'Wheel')
        then self.subartifactof->size = 1
        and self.subartifacts.type = 'Car'
endif

inv isEngine: if (self.type = 'Engine')
        then self.subartifactof->size = 1
        and self.subartifacts.type = 'Car'
endif
```

ISO STEP

STEP data modeling is based on another ISO language, EXPRESS. EXPRESS is used to represent both information models and data instances. The EXPRESS language has been developed for STEP and is used almost exclusively within this community. EXPRESS has a native support for expressing rules. Unfortunately since EXPRESS has been developed for STEP only, few implementations are available - and none of them supports user-defined (business) rules. Fortunately, STEP also has an option to represent data using XML. This makes it easier to write and run business rules. But once again, as we saw for the CPM, rules are supported by the modeling language(s), but ISO does not provide specific recommendations on writing them. So, at best, ISO STEP partially meets the business rules requirement.

OASIS DEX for PLCS

OASIS does not provide any recommendations on writing and executing business rules. DEXs developers have access to the same resources as STEP users and are facing the same issues related to the language they use.

OAGi specification

OAGIS architecture features a "BOD constraints" mechanism for writing and executing rules on individual BODs. OAGIS is delivered with pre-existing BOD constraints that can be modified/extended for answering specific needs. This mechanism also enables users to write new BOD constraints for extensions. OAGIS architecture is based on XML technologies so it does benefit from solutions we mentioned before. Additionally, Schematron is the one chosen by OAGi to specify BOD constraints. This architecture fully meets our requirements since users can write business rules on both the initial model and their customizations, which are extensions.

3.4 Conclusion

In this chapter, we have identified specific requirements regarding customization of information models for PLM. From these requirements we derived 5 evaluation criteria: 1) the type of customization for its impact on the implementable data schema, 2) the level of semantics used by the framework for its impact on the consistency checking, 3) the control mechanism on customization for its faculty of reducing inconsistencies and complexity of development, 4) the representation of business objects, which are high level aggregations of information for reducing complexity of implementation, and 5) the recommendations on applying business rules, that are implementable form of business constraints, for enabling more data validation.

We performed an evaluation of the existing frameworks using these 5 criteria; the results are summarized in Table 3-1. From the evaluation it appears that no single framework meets all the requirements. Therefore, a new framework is needed. This new framework is presented in the following chapter.

Table 3-1 Summary of frameworks evaluation

	CPM extension	CPM specialization	STEP UDA	STEP external classification	OASIS DEX	OAGIS UserArea	OAGIS Overlay
Customization	Extension, expressed in UML.	Specialization of concepts with a string attribute	Specializations of attributes with a string attribute.	Specialization using an external source of classification	Similar to STEP + recommendation on using OWL format for the external source of classification	Extension: generic field that can store any information	Extension: extension/creation of BODs, Nouns and Components
Semantics	UML has no formal semantics but consistency checking can be performed and results are tied to the implementation	Consistency checking cannot be performed using string values	Type of a UDA is known because of the information structure. Its semantics is encoded as a string value.	Due to the freedom given to users on the format of the external source, we cannot make a general statement	OWL has a formal semantics which allows native consistency checking but the classification is not complete enough to do it	XML has no formal semantics but consistency checking can be performed and results are tied to the implementation	XML has no formal semantics but consistency checking can be performed and results are tied to the implementation
Controlled customization	Conceptual model can be customized without any restrictions.	Only concepts with the "type" attribute can be customized, which corresponds to ALL the concepts.	Only for attributes, and can be applied only to certain types of concepts.	Only a restricted set of entities can be customized	Only a restricted set of entities can be customized	Only Components can be customized	Only Components, Nouns and BODs can be customized.
Business objects	There is no explicit representation of business objects	There is no explicit representation of business objects	There is no explicit representation of business objects	There is no explicit representation of business objects	There is an explicit representation of business objects	There is no explicit representation for new business objects in the customization	Native support for business objects.

Business rules	There are no recommendations but it is technically feasible.	There are no recommendations but it is technically feasible.	There are no recommendations and it is technically challenging with EXPRESS. STEP-XML is easier to implement.	There are no recommendations and it is technically challenging with EXPRESS. STEP-XML is easier to implement.	There are no recommendations and it is technically challenging with EXPRESS. STEP-XML is easier to implement.	Special architecture has been developed to support business rules.	Special architecture has been developed to support business rules.

4 New framework components for PLM customizations

4.1 Introduction

In the previous chapter, we evaluated several of the existing frameworks. Based on the analysis of the evaluation results, we concluded that (1) individually, no single framework fully met our requirements, and (2) even collectively, all of our requirements could not be fully met(see Appendix A for more details). It is still not possible to fully meet our requirements by combining components from all of the frameworks. The challenge here is to identify those components that partially or fully meet our requirements, modify them if necessary, add any new components, and combine them into a single, composite framework that fully meets all the requirements. When making any modifications or developing new components, we will use existing, standard technologies.

4.2 Customization

We identified in 3.2.1 the need for a customizable PLM framework to support dynamic specialization. For our framework to fulfill this requirement we developed a customized version of the STEP entities that enable dynamic specialization.

As with ISO and OASIS, our new framework also implements specialization using a controlled vocabulary approach but in a new way that combines the best parts of each. We introduce a new, minimalist conceptual model, called the *embryo.* When extended, the *embryo* enables development of a customizable information model by specialization. This specialization reuses what ISO does but requires the external classification to be expressed in OWL - as mandated by OASIS. To emphasize the use of OWL and ontologies for classification, we do not use the STEP naming rules for the different concepts in the *embryo.*

The *embryo* also includes and modifies the UDA mechanism from STEP. This is a major benefit to users who can define and append specific attributes/properties to some concepts. Here the *unit* of a property is not defined as a string anymore; it is classified using the same mechanism as previously defined with an ontology. The same approach is applied to the property itself.

In Figure 4-1, we give a version of the *embryo* that does not show all the data types it supports (for ease of reading). This *embryo* is represented with a UML class diagram for clarity and documentation purposes only. UML is not required; users are free to choose the implementation method with which they feel most comfortable.

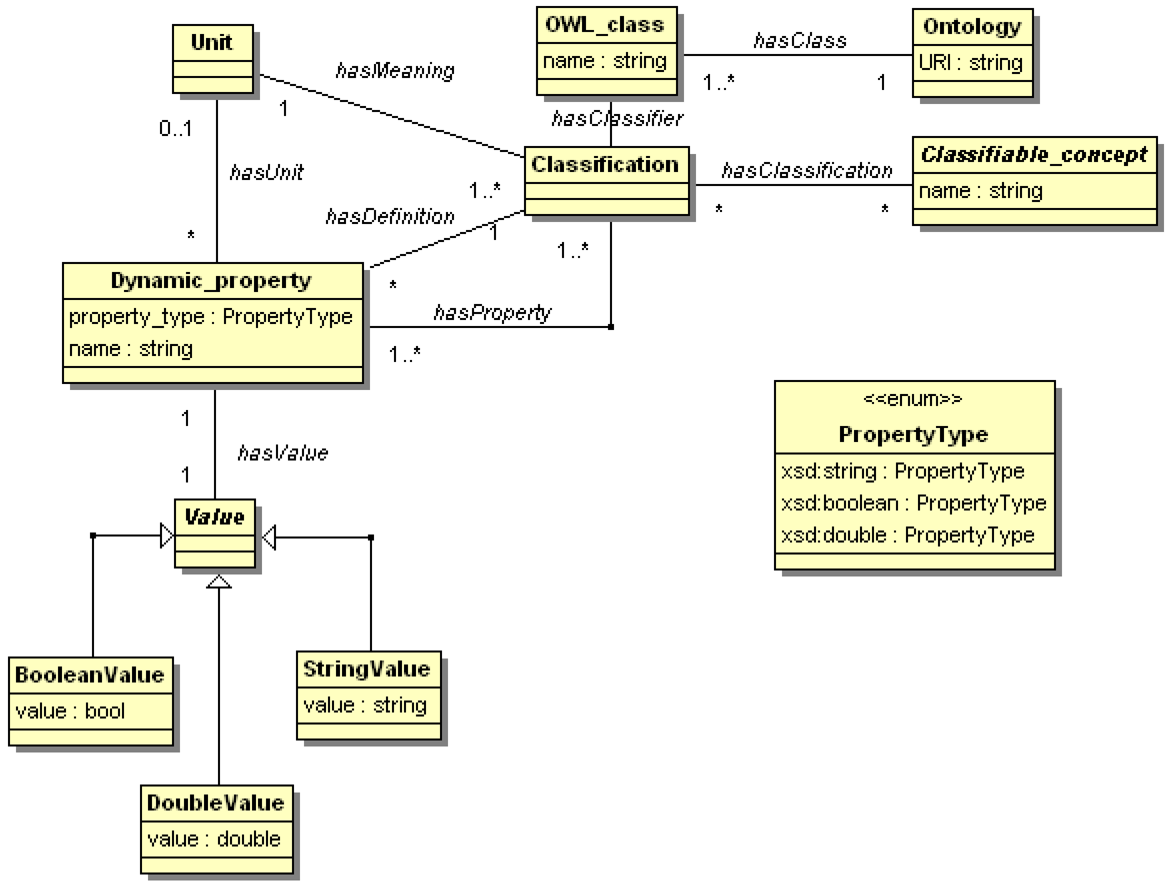


Figure 4-1 Embryo UML class diagram

This model is designed to support specialization of concepts as defined below.

Classification is meant to connect a classifiable concept to its specialization type defined in an ontology. The specialization type is represented with an **OWL_class** whose attribute name is the name of the class used to specialize the classifiable concept. The source ontology where the specialization type is defined is represented with **Ontology** whose attribute URI represents the location of the ontology. The abstract **Classifiable_concept** has to be seen as the root of all the concepts that will be specialized. It is meant to be sub-classed by these concepts during definition of the information model. **Classifiable_concept** is the link between the *embryo* and the user's information model.

Dynamic_property is a mean of representing properties for specialization types, where the property has a name and a property_type. The property_type attribute of a **Dynamic_property**

is of **PropertyType -** that is an enumeration type built from the XML Schema built-in datatypes[4]. A property defined this way needs to use a **Classification** to identify the ontology where it is defined (cf. ***hasDefinition*** relationship). The value of the property is represented using one of the subclasses of **Value** (for ease of reading we do not show all the subclasses in **Error! Reference source not found.** but the embryo uses any XML Schema built-in datatypes). A **Unit** for a property might be defined, and the **Unit** definition is expressed through a **Classification** (cf. ***hasMeaning*** relationship).

A specialization type and a dynamic property are connected through the ***hasProperty*** relationship. The property is linked to the classification because the property is owned by the type of specialization, not by the classifiable concept.

Figure 4-2 shows an example where we use the *embryo* to create a simplistic information model in which a Product is an aggregation of Part. In this example we define Part and Product as concepts that can be specialized.

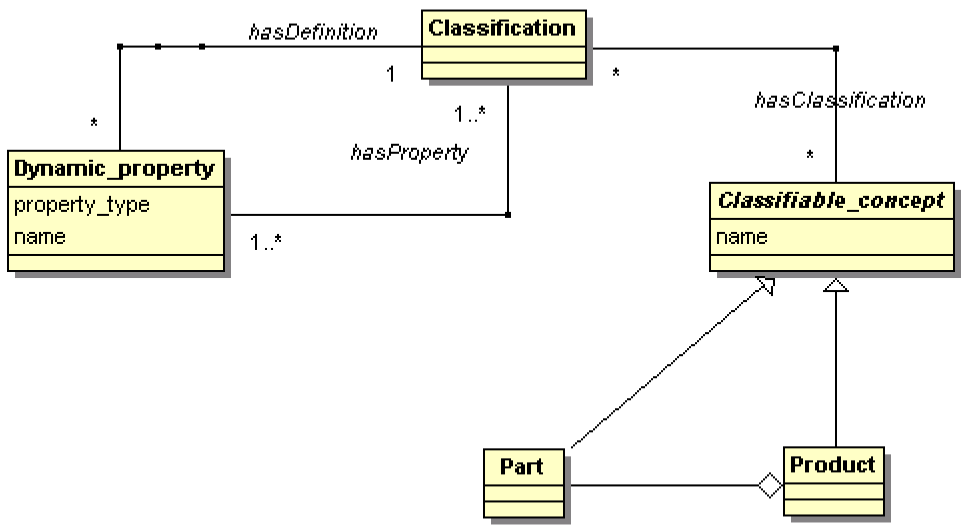


Figure 4-2 Embryo minimalist extension example

This model can then be instantiated and used to represent a vehicle as in Figure 4-3. In this UML object diagram, we represent a Car by an aggregation of four Doors. Car and Door are concepts that are defined in a fictive ontology (http://sylvere.org/ontology.owl). We retrieve those concepts using instances of **OWL_class** that we use to classify instances of Part and Product. The four parts should be seen as Door due to the classification Door, and the Product should be seen as a Car due to its classification. The four parts share the same classification, since we want them to represent the same type of part. We also define a **Dynamic_property** that represents the weight of the car. To do so, this property is classified using (1) the same ontology where Car and Product are defined and (2) an **OWL_class,** which defines the property itself, from the ontology. This property is linked to the classification of the car. This weight

[4] XML Schema Data Types http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

property has a xsd:double value represented by a **DoubleValue**. We also define its **Unit** using the Quantities, units, Dimensions and Data Types in OWL and XML (QUDT) (Hodgson & Keller, 2011) ontology and the Kilogram **OWL_class** from it.

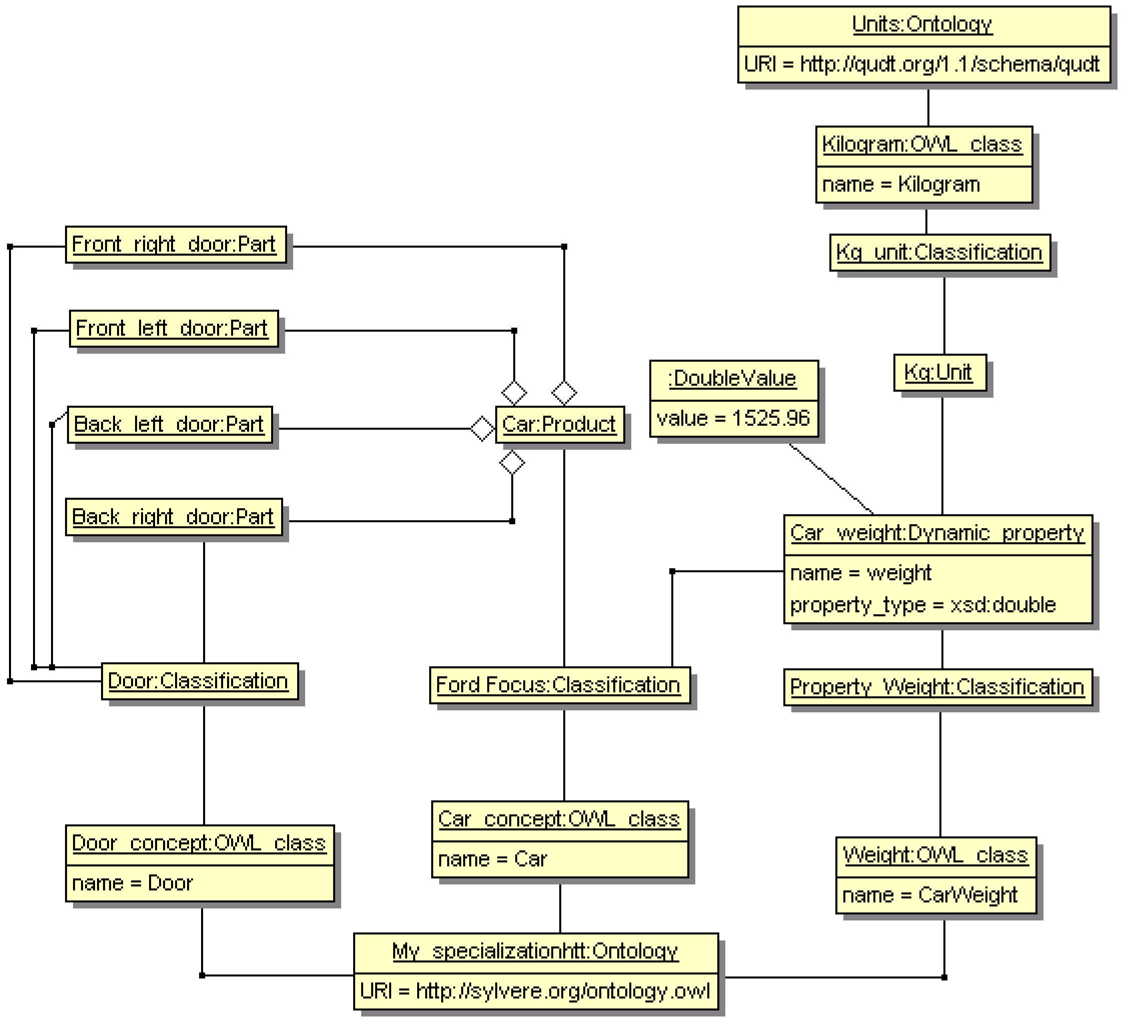


Figure 4-3 Embryo instantiation representing a car model with classification

In this state, the *embryo* provides a mechanism for specializing concepts. It also provides a way to define and attach properties to a specialization type. This section has shown implementation of specialization using OWL. In the next section we will justify the use of OWL to meet another requirements, semantics.

4.3 Semantics

In Chapter 3 we saw the importance of defining formally the semantics of the concepts introduced by the specialization. Semantics enable consistency checking of the specialization and its instantiation. We also identified a lack of formal semantics for XML and UML. Because of this, models created in these languages can have possible inconsistencies, though it has been overcome by different research efforts (Balaban & Maraee, 2006; Usman, Nadeem, Kim, & Cho, 2008)(Berardi, Calvanese, & De Giacomo, 2005). Because of its formal semantics based on description logic, OWL allows unambiguous understanding and definition of ontologies, for both human and machine. Consequently, consistency checking of ontologies created in OWL can be performed automatically. In our framework, therefore, we mandate the use of OWL for defining the classifiers of the specialization.

In addition to consistency checking, validation of integrity constraints with instance data is also important for our framework. For an example of an integrity constraint let's consider a car is defined as a product with four wheels (the constraint); any instance of car that does not have four wheels should be seen as an error. Using OWL with the absence of Unique Name Assumption (UNA), we are essentially dealing with the Open World Assumption (OWA)(Sirin & Tao, 2009)(Motik, Horrocks, & Sattler, 2007) . In the open world, a car with three wheels cannot be seen as inconsistent with our constraint. This can happen because it is possible that this car has four wheels, but the information about the fourth wheel has not been discovered yet. In other words, open world means that we cannot assume that our knowledge base is complete. As a result, it is quite complex to use native OWL mechanisms for integrity constraint validation. We need an approach that simulates a closed world

Research efforts (Motik et al., 2007; Reiter, 1998; Sirin & Tao, 2009) in this domain have yielded some approaches, implementations, and software (Knublauch, Hendler, & Idehen, 2011; Parsia, 2011) that provide solutions for validation of integrity constraints when using OWL. At this point in time, SPIN (SPARQL Inferencing Notation) is the only implementation publicly available, known to work, and meets our needs[5]. SPIN is a SPARQL-based rules and constraints language with an object-oriented approach. With SPIN users can define rules and constraints at the class definition level, and then apply them to instances. More importantly for our purpose, implementations of SPIN can simulate a closed world.

We will now show how to resolve the three-wheels problem above using SPIN. First, we define a simple ontology with four classes. We define classes Part and Product as subclasses of owl:Thing, Wheel a subclass of Part, and Car a subclass of Product. We also define an object property hasWheel whose domain is Car and range is Wheel. We now want to make sure that any instance of Car is connected to four wheels - we could not do this before - using the hasWheel object property. Then, we use the built-in SPIN function called Attribute. Attribute takes an object property, an object type, a minimum cardinality and a maximum cardinality as input parameters. When attached to a class definition through the spin:constraint mechanism, this functions triggers an error when an instance of this class does not respect the cardinalities with the given object property and object type. In our case, we use it to represent the fact that an instance of car has **a maximum and minimum** of four instances of hasWheel with objects of type Wheel. Figure 4-4 shows this using TopBraid Composer[6], a SPIN editor.

[5] SPIN has been submitted as a proposal for W3C standardization (Knublauch et al., 2011).

[6] Available at http://www.topquadrant.com/products/TB_Composer.html

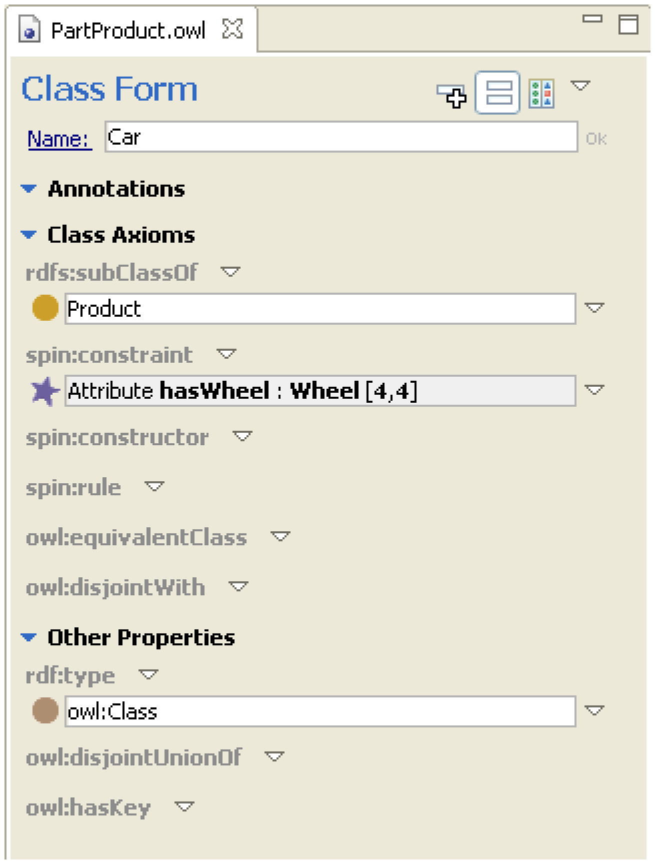


Figure 4-4 SPIN constraint using Topbraid

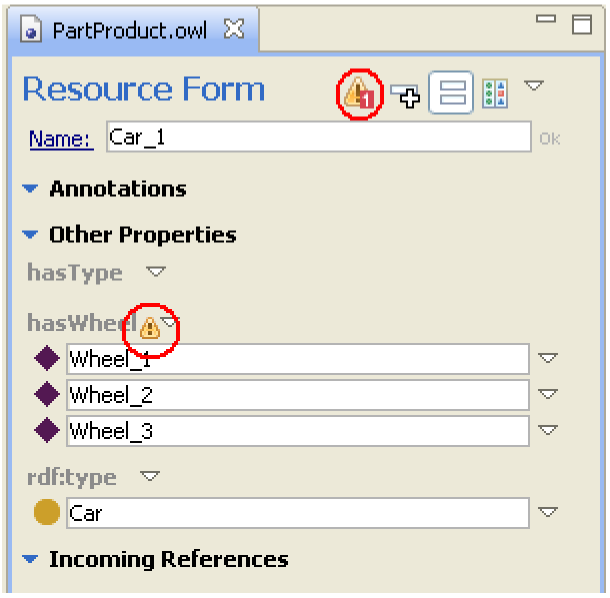
Using the n3[7] notation from the W3C, the same constraint can be expressed as follows:

```
:Car
      a        rdfs:Class ;
      rdfs:label "Car"^^xsd:string ;
      spin:constraint
              [ a        spl:Attribute ;
                rdfs:comment "A Car must have 4 Wheels"^^xsd:string ;
                spl:maxCount 4 ;
                spl:minCount 4 ;
                spl:predicate :hasWheel ;
                spl:valueType Wheel ] .
```

After creating an instance of Car, called Car_1, we connect it to only three instances of Wheel to see if the SPIN constraint triggers an error. Figure 4-5 shows (1) how that the inconsistency has been identified by SPIN and (2) how it is shown to the user within the TopBraid tool. The warning signals we highlighted in red are representations of the existence of an inconsistency. The warning in the top corner shows that this individual (Car_1) violates SPIN constraints and

[7] http://www.w3.org/TeamSubmission/n3/

the number in red indicates how many constraints are violated. The other warning signal next to the hasWheel object property indicates the source of the violation.



Car_1
Wheel_1
Wheel_2
Wheel_3
Car

Figure 4-5 SPIN constraint validation with Topbraid

In this section, we argued that XML and UML are not suitable for our framework. They have important limitations because of their lack of formally defined semantics. We saw that OWL overcomes this limitation. However, OWL has another limitation. It does not provide native mechanisms for integrity constraints because it is based on OWA. We then showed how to use SPIN to overcome this limitation.

4.4 Controlled customization

In 3.2.3 we explained the importance of controlling the customization. Depending on the real-world context, it is not always necessary to provide specialization for every concept. Reducing the set of specializable concepts is a way to decrease the complexity of writing customizations. It also reduces the likelihood of inconsistencies.

In 2.2.3 we saw the mechanism in use by OASIS for its PLCS framework. This mechanism, based on ontology and STEP, allows users to customize only a specific set of concepts. The set allowed by STEP includes all the concepts that have a link to the **Classification_assignment** EXPRESS entity. The OASIS PLCS approach has derived a (flat) proxy ontology from this set of classifiable concepts. We consider a proxy ontology as an ontology that contains only classes and no object/data properties. It requires users to create reference data for specialization from that set.

In this framework, we use the same approach and recommend users to generate a proxy ontology based on the concepts that need to be classified. This proxy ontology has to be the starting point for writing specializations. Also, any specialization must be derived from the proxy ontology. Implementation of this approach will be demonstrated in Chapter 5.

4.5 Business objects

In 3.2.4 we explained the two potential benefits of aggregating low-level information objects from the original information model into high-level business objects. The first is better human readability. This happens because business objects use business specific terms with which end users are already familiar. The second is simpler computer implementation. This is achieved by providing a high-level API for information management (create, read, update and delete operations).

To realize these potential benefits, business object models must be both human and computer interpretable. In 3.3.4 we showed that only two frameworks provide a mechanism for explicitly and formally representing business objects: OAGIS and OASIS DEX for PLCS. OAGIS business objects are defined formally with an XML schema that makes them easily computer interpretable. Unfortunately, XML schemas can be understood only by people with special training, such software engineers. The OASIS approach is based on templates as defined in 2.2.3.1. In (Krima et al., 2011) we described drawbacks of this approach and provided a solution based on the UML activity diagram, called DEXML. From its beginnings as a graphical language, UML has always been and remains human interpretable. A computer interpretable representation of UML, XML Metadata Interchange (XMI) (OMG, 2011), was developed as an OMG standard. A number of tools exist to generate XMI from UML and it is widely used in different domains. To address interoperability issues within PLM, the OMG recommends the canonical XMI schema (OMG, 2009), which grew out of an agreement between several major UML software vendors.

DEXML uses the UML activity diagram to describe formally, in an ordered fashion, the different data instantiations and assignments associated with a business object. Instantiation of a concept (a UML class in this context) is represented by an instance of *Create Entity Action*. *Create Entity Action* overrides the UML *Create Object Action* and allows it to have *input pins* that represent attributes of the concept (Krima et al., 2011). Users are then able to bind values to these *input pins* to represent data assignments. DEXML was designed initially for PLCS; but its approach can be applied to any information model that has a UML representation as a class diagram. Because the result of DEXML is an activity diagram, it provides both a graphical and an XMI representation that are respectively human and computer readable.

Based on the example we introduced in 3.2.4, Figure 4-6 shows a detailed DEXML representation of the **Shipment** business object. With DEXML we are able to represent formally the mapping(s) between a business object and the information elements that compose it. Figure 4-6 shows the mappings between the attributes of the **Shipment** business object and the different attributes of **Person**, **Address** and **Organization**. The XMI representation of this model is located in Appendix C. The canonical XMI representation of this model, generated using the NIST tool (NIST, 2006), is located in Appendix D.

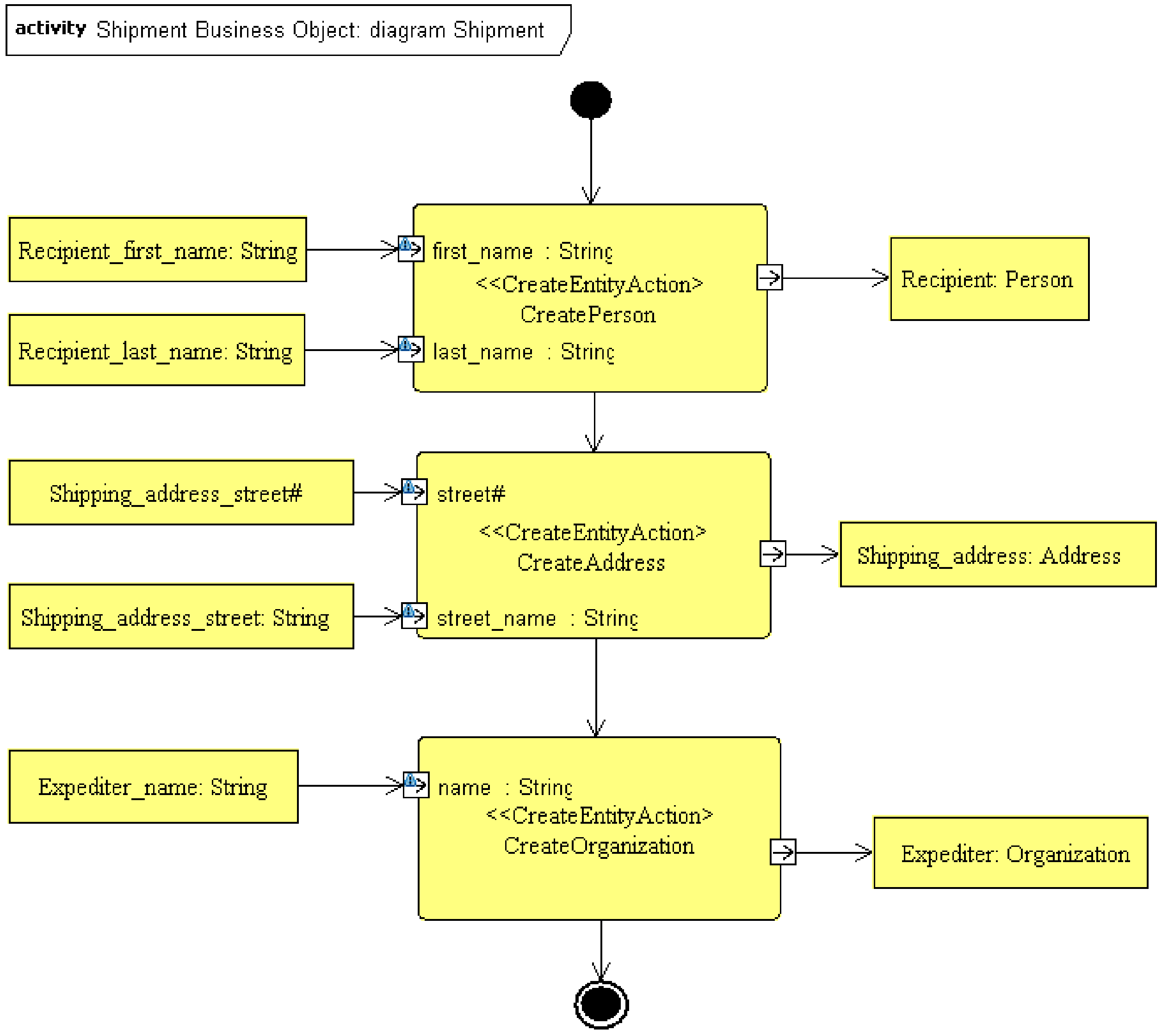


Figure 4-6 DEXML representation of the Shipment business object

Note, the attributes derived from the relationships between the classes are missing and not represented as input pins. This happens because DEXML was based on the physical model of ISO 10303-239, which was implemented as a 10303-11 (Part 11) file. In this model, all attributes of an entity are embedded within the entity; and, relationships between entities are represented as attributes of entities. Consider the following example: the initial logical model in Figure 4-7 is presented in EXPRESS-G, External_source expresses a relationship between two classes:

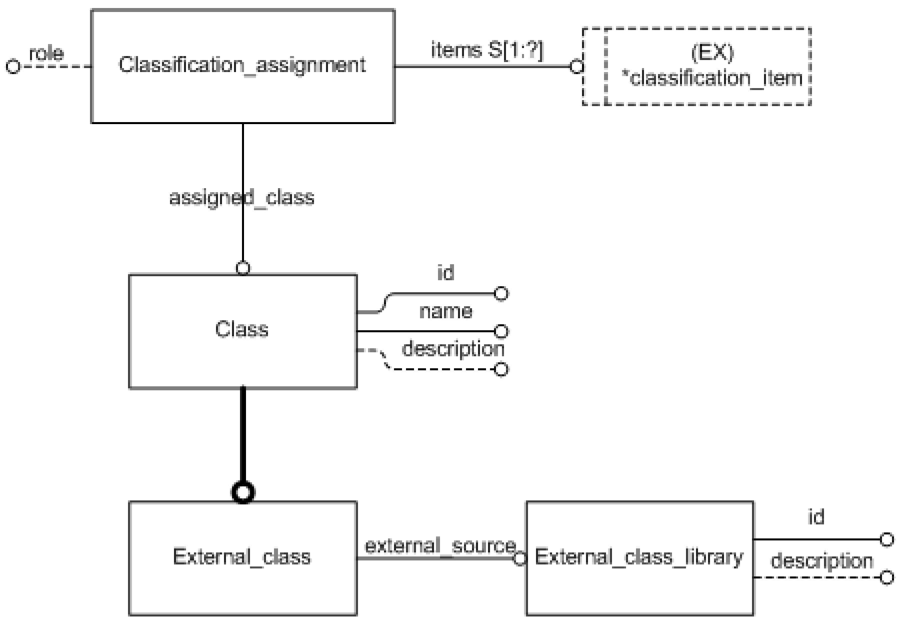


Figure 4-7 An EXPRESS information model

External_class and External_class_library. As shown in the following corresponding Part 11 physical file, that relationship is mapped to an attribute, external_source, of the ENTITY called External_class.

```
ENTITY External_class
SUBTYPE OF (Class);
external_source : External_class_library;
END_ENTITY;

ENTITY External_class_library;
id : STRING;
description : OPTIONAL STRING;
END_ENTITY;
```

From the Part 11 file, DEXML uses Reeper (Barnard-Feeney & Price, 2011) to generate a UML-based physical model that keeps the same representation: the class **External_class** has an attribute external source of type **External_class_library**. As a result, DEXML reads only attributes that are embedded within a class; it does not know, and cannot infer, if one of these attributes was derived from a relationship. Since we need this capability in our framework, we developed the following algorithm:

Input:
C: C is a class that will be instantiated
Output:
LIST_OF_ATTRIBUTES: LIST_OF_ATTRIBUTES is a list of attributes that will be given to the DEXML method that creates the input pins for the DEXML Create Entity Action of C.

```
Create LIST_OF_ATTRIBUTES
For each Relationship R from C to END_CLASS
/*
*   If the class C has an attribute that has the name of the relationship we consider
*   that this attribute is derived from the relationship
*/
If there is no attribute  in C with a name identical to the name of R Then
Create an attribute DERIVED_ATTRIBUTE
DERIVED_ATTRIBUTE.name = R.name
DERIVED_ATTRIBUTE.cardinality = R.cardinality
DERIVED_ATTRIBUTE.type = END_CLASS.type
Add DERIVED_ATTRIBUTES to LIST_OF_ATTRIBUTES
EndIf
EndFor
Add all original attributes of C to LIST_OF_ATTRIBUTES
```

Consider the class diagram in Figure 4-8 as initial information model. Now we apply this new algorithm to create a business object that instantiates these two classes. The trace of the execution of the algorithm follows.

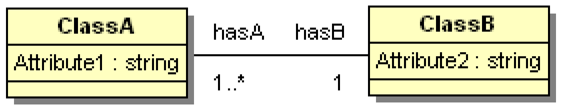


Figure 4-8 An information model before transformation seen by DEXML before modifications

```
STEP 1
C = ClassA
LIST_OF_ATTRIBUTES = EMPTY
R = hasB
END_CLASS = ClassB
DERIVED_ATTRIBUTE = EMPTY
DERIVED_ATTRIBUTE . name = hasB
DERIVED_ATTRIBUTE .cardinality = 1
DERIVED_ATTRIBUTE .type = ClassB
LIST_OF_ATTRIBUTES = [hasB ]
LIST_OF_ATTRIBUTES  =  [hasB , Attribute1]

STEP 2
C = ClassB
LIST_OF_ATTRIBUTES = EMPTY
R = hasA
```

```
END_CLASS = ClassA
DERIVED_ATTRIBUTE = EMPTY
DERIVED_ATTRIBUTE . name = hasA
DERIVED_ATTRIBUTE .cardinality = [1,*]
DERIVED_ATTRIBUTE .type = ClassA
LIST_OF_ATTRIBUTES = [hasA]
LIST_OF_ATTRIBUTES  =  [hasA , Attribute2]
```

This makes DEXML see the information model not the way it was originally designed but like Figure 4-9 where all attributes are embedded.

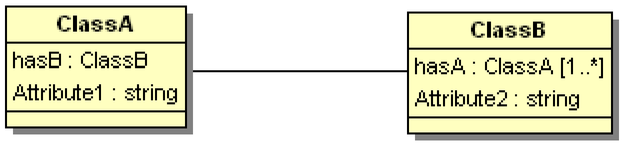


Figure 4-9 Information model 4-8 after modifications

When applying this algorithm on the Shipment example we previously described in this section, the resulting DEXML Activity diagram for instantiation looks like Figure 4-10.

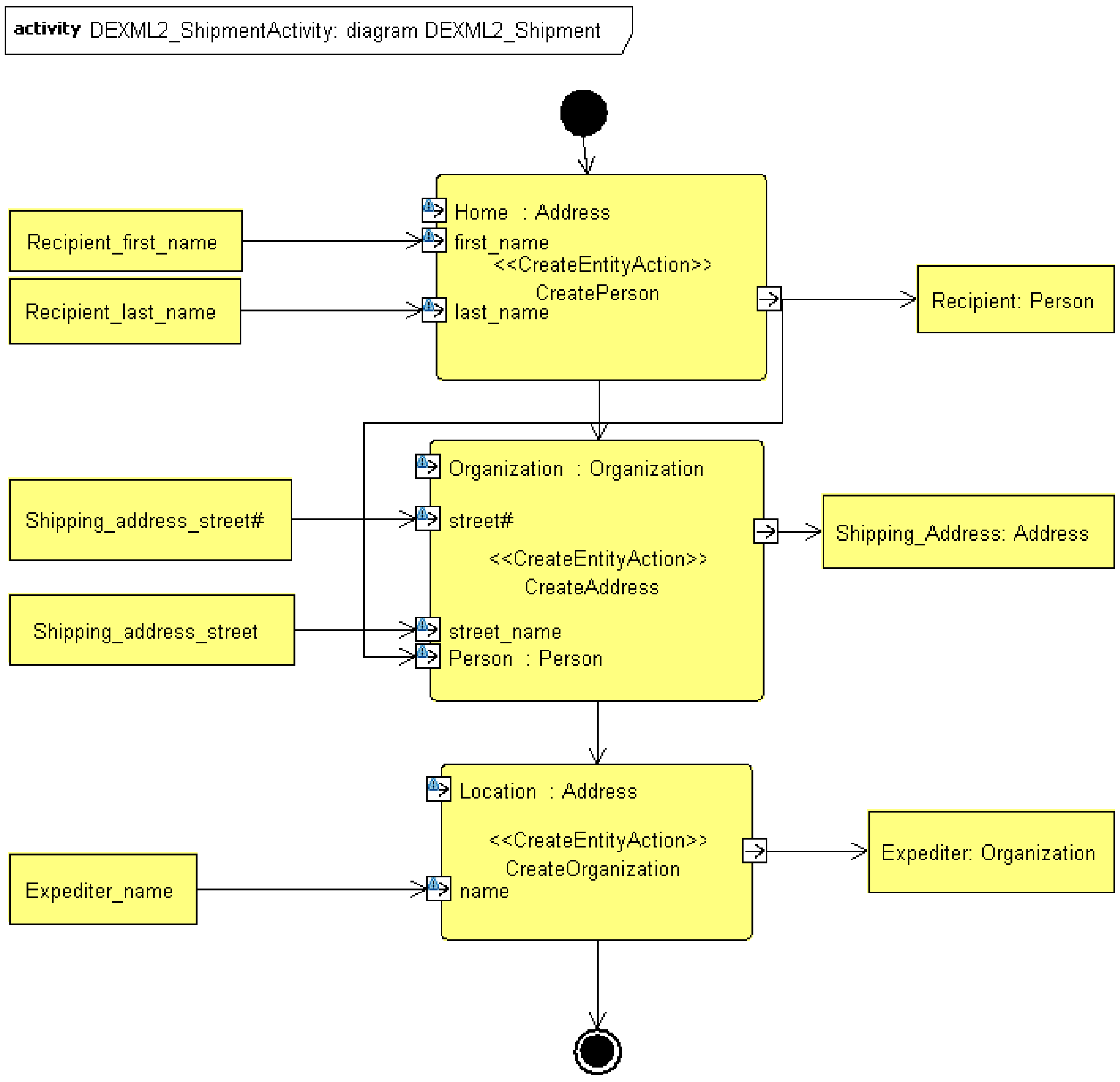
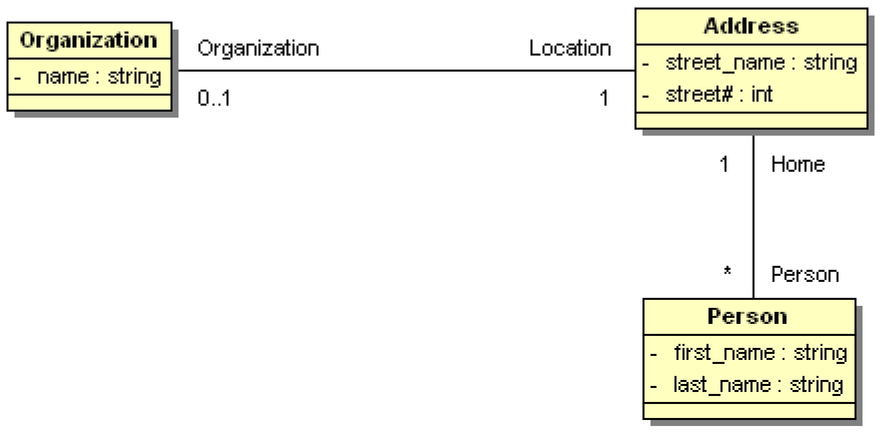


Figure 4-10 Diagram 4-6 after DEXML modifications

In this diagram we can see new *input pins* such as Home, Organization, Person and Location. These *input pins* are inferred from the Shipment information model, which has been modified from the one used in Chapter 3 and is as follows:



In this section we discussed the importance of a formalized representation of business objects so they can be understood by both people and machines. UML satisfies only the former. OMG has standardized a version of UML, canonical XMI, which is computer interpretable and supports our need for interoperability across diverse PLM applications. But, business object interoperability is not enough for PLM. We must be able to validate instantiations of business objects against domain-specific constraints. We discuss this in the next section.

4.6 Rules

In 3.2.5 we discussed the need to represent domain-specific constraints. We suggested business rules as a way of expressing the semantics of those constraints in PLM. This is technically feasible with all frameworks. In general, the mechanism for defining business rules must be compatible with the one chosen for representing the information and all of its customizations. Only OAGIS provides recommendations and an approach on how to implement such a mechanism. But, that approach is based on XML representations. It cannot, therefore, be used within our new framework, which uses OWL. We need a mechanism to implement business rules in OWL.

To develop this new mechanism, we need to understand better how business rules work. Business rules provide constraints based on a domain-specific context. Those constraints usually apply to a number of instances. Therefore, instances are not checked individually but as a group. This implies that instances are checked multiple times, individually against a single constraint and collectively against multiple constraints. Earlier, we used the SPIN mechanism for constraint checking of individual concepts or instances of those concepts. Defining and executing business rules requires a mechanism that is able to query data based on multiple constraints - sometimes called collective validation. Since SPIN is based on SPARQL, which is a query language for RDF that allows querying and filtering data, it can be that mechanism.

In 4.3, we demonstrated how to use SPIN to validate an individual constraint. The example we gave was that any instance of a car must have four wheels. An example of business rule, with multiple constraints, for this example would be that cars must have four wheels and these wheels must have the same diameter.

There are three ways to express this constraint, all based on SPARQL. A first way is to write a SPARQL query and execute it. Such a query could be written as follows:

```
SELECT DISTINCT “invalid car”?c
WHERE {
?c rdf:type :Car;
      :hasWheel ?1;
      :hasWheel ?2;
      :hasWheel ?3;
      :hasWheel ?4.
?1 owl:differentFrom ?2;
      owl:differentFrom ?3;
      owl:differentFrom ?4.
?2 owl:differentFrom ?3;
      owl:differentFrom ?4.
?3 owl:differentFrom ?4.
?1 :hasDiameter ?d1.
?2 :hasDiameter ?d2.
?3 :hasDiameter ?d3.
?4 :hasDiameter ?d4.
FILTER (!(?d1 = ?d2 && ?d2 = ?d3 && ?d3 = ?d4))
}
```

This query identifies all the wheels connected to a car and retrieves all cars that have all the wheels with the same diameter. As noted above, this query, which is independent of SPIN, can only be run after the individual validation. That is it can only be run on those cars that meet the constraint of having four wheels.

The second way of defining and executing such a business rule is by using the SPIN constraints introduced in 4.3. Unfortunately, this mechanism needs to be tied to the class definition language, which is OWL in our case. It also diverges from the notion of collective validation because the validation is performed within the context of the class itself. Though it is not recommended, such a constraint would be attached to the class definition of Car and defined as follows:

```
CONSTRUCT {
    _:b0 a spin:ConstraintViolation .
    _:b0 spin:violationRoot ?this .
    _:b0 spin:violationPath :hasWheel .
    _:b0 rdfs:label "All wheels should have the same diameter" .
}
```

```
WHERE {
    ?this :hasWheel ?1 .
    ?this :hasWheel ?2 .
    ?this :hasWheel ?3 .
    ?this :hasWheel ?4 .
    ?1 owl:differentFrom ?2 .
    ?1 owl:differentFrom ?3 .
    ?1 owl:differentFrom ?4 .
    ?2 owl:differentFrom ?3 .
    ?2 owl:differentFrom ?4 .
    ?3 owl:differentFrom ?4 .
    ?1 :hasDiameter ?d1 .
    ?2 :hasDiameter ?d2 .
    ?3 :hasDiameter ?d3 .
    ?4 :hasDiameter ?d4 .
    FILTER (!(((?d1 = ?d2) && (?d2 = ?d3)) && (?d3 = ?d4))) .
}
```

A SPIN engine would then flag all instances of Car that violate the constraints. Within the TopBraid editor, constraint violation will appear as a warning sign as shown in Figure 4-5. This validation can be performed during the individual validation, unlike the previously described mechanism in which validation is a separate step.

A third way of defining and executing business rules is to query the model and enrich it with the result so that invalid instances would be classified as invalid. This means that the ontology is enriched with concepts representing invalid data. Any concept should have its opposite. In the context of the car example, the concept Car requires the concept InvalidCar. This can be done within SPARQL using the CONSTRUCT mechanism as follows:

```
CONSTRUCT { ?c rdf:type :InvalidCar}
WHERE {
?c rdf:type :Car;
      :hasWheel ?1;
      :hasWheel ?2;
      :hasWheel ?3;
      :hasWheel ?4.
?1 owl:differentFrom ?2;
      owl:differentFrom ?3;
      owl:differentFrom ?4.
?2 owl:differentFrom ?3;
      owl:differentFrom ?4.
?3 owl:differentFrom ?4.
?1 :hasDiameter ?d1.
?2 :hasDiameter ?d2.
?3 :hasDiameter ?d3.
?4 :hasDiameter ?d4.
FILTER (!(?d1 = ?d2 && ?d2 = ?d3 && ?d3 = ?d4))
}
```

This CONSTRUCT classifies any invalid instance of Car as an instance of InvalidCar. Unfortunately, this classification does not prohibit the use of this invalid data in other extensions or business objects. That is, other users may unknowingly use this data unless we can enrich the ontology enough to make them aware of the error. Since collaboration is an important aspect of PLM, it is crucial that all actors working on the same product have access to this kind of information. Consider the following example in which three engineers are designing different parts of the same car. Engineer A is designing wheels, engineer B is designing the front axle, and engineer C designs the assembly of the wheels with axle. C is designing a part that is an aggregation of other parts. Therefore, it is a business object on which business rules may apply. Our goal is to determine automatically if this aggregation is valid or invalid. If engineer A or engineer B makes an error, we need to make it possible for engineer C to have access to this knowledge.

To enable this access, our strategy is to use the ontology as the means of communication. In this way, other users of the same data cannot unintentionally introduce inconsistencies into the ontology. This strategy is realized in 3 steps: 1) the opposite of a concept should be disjoint from the concept itself 2) invalid instances are classified using the opposite concept 3) reasoning is performed again. The reasoner will then trigger an inconsistency since we declare an individual that is an instance of two disjoint concepts.

From the preceding discussion, we now realize the importance of the consistency between the language used to model the information and the language used to express the business rules. In earlier sections of this chapter, we chose OWL as our information modeling language because of its support for consistency and constraint checking of individual concepts or instances of those concepts. Here, we described three mechanisms for defining business rules using SPARQL queries, a SPIN constraint, and, an ontological inconsistency construct. The choice of the mechanism depends on the importance of the business rule and the consequence it has on others using the same data. The best choice for PLM is discussed more in the next chapter.

4.7 Product Family Management

4.7.1 Requirements

In the previous sections of this chapter we have seen how the framework we developed can be used to handle needs of customization for product data modeling and exchange throughout the entire product lifecycle. Since this framework was developed specifically to answer technical requirements (see Chapter 3) from PLM, it does not support directly customizations associated with business requirements such as addressing customer preferences.

As the world moves toward mass product customization, the diversity of customer preferences will pose significant challenges for both product designers and information modelers (Feng et al., 2003). To meet these challenges, companies need a way to customize both the variations of an initial physical product and all of the information associated with those variations (Zhang et al., 2006). The initial product with its variations is commonly called a product family. The customization of the associated information is called Product Family Management (Johnson et al., 2010). We refer to these information variations as business oriented customizations, since they directly impact a company's ability to satisfy its customers.

Variations of a product can be designed either by modifying its existing properties or by creating new properties. A parallel can be drawn between the two (2) product customization mechanisms and the two (2) information models customization we presented in Chapter 3: Extension and Specialization. We then define **product specialization** as the customization of a product by constraining values of its existing properties. We define **product extension** as the customization of a product by defining new properties.

In this section we highlight the limitations of our framework and DEXML for representing and managing product specialization and product extension. We then present an extension to the DEXML UML profile to support both. The last section describes the necessary components for integrating the DEXML extension with the rest of the framework.

4.7.2 DEXML limitations for Product Family Management

To develop models that represent and manage product family information, we must be able to (1) represent products as a complex association of information, (2) represent a hierarchical relationship among different products of the same family and (3) represent customization(s) among the different family members. A product model comprises a large variety of pieces of information of different levels of abstraction and complexity. The challenge when dealing with product data is to efficiently manage and represent the complex instantiations and aggregations of the different pieces of information. In Section 4.5 we described DEXML as a tool to "describe formally, in an ordered fashion, the different data instantiations and assignments associated with a business object". We used DEXML to represent the complex underlying information model of a single product. Despite enabling graphical and formal representation of complex information structure such as products, DEXML cannot be used directly to capture, represent, and manage all of the information requirements associated with a family of products. Recall that DEXML information structures/models, called business objects, are not semantically categorized. This means that they cannot be stereotyped as products. This makes it impossible to create and control the hierarchical relationships between models needed to represent the members of a product family.

Consequently, DEXML does not provide a mechanism for representing the necessary customizations among members of a product family. On the other hand, DEXML already has a mechanism to represent complex aggregations and instantiations of information, which makes it a good candidate as a foundation for building a solution. As seen in Section 4.5 DEXML consists of a UML profile that extends initial UML information modeling mechanism to support new requirements. Because DEXML provides a partial solution to representing product families we intend in extending it with a new UML profile to fulfill our requirements in terms of information modeling capabilities.

4.7.3 A DEXML-based UML profile for Product Family Management

In the previous section, we have stated that DEXML is limited in its ability to represent product family information. We have identified four such limitations. First, we need an ability to classify DEXML business objects as products. This is necessary to ensure (1) that all members of the family are indeed products and (2) that all products are indeed members of the same family. This provides a basis for developing the classification needed to control the hierarchical relationships among the members of a product family. Second, we need a mechanism for defining these hierarchical relationships. Third, we need to be able to define customization points in a product. Last, users should have the possibility to choose what part of a product can be customized, whether by extension or specialization.

To remove these limitations, we developed a new UML profile that extends DEXML (see Figure 4-11). In the following sections, we describe this new UML profile and show how it removes those limitations.

PFM::ProductFamilyMember
In DEXML, a business object is represented as a UML::Activity. To offer the possibility to classify a business object as a product that can be used in a family, we developed the PFM::ProductFamilyMember stereotype, a specialization of UML::Activity, which has a list of PFM::Specialization_Entry and Option. PFM::ProductFamilyMember also has an attribute *Parent* of type PFM::ProductFamilyMember that must be used to represent the product family structure. This attribute represents the PFM::ProductFamilyMember of which the owner is a customization.

PFM::Specialization_Entry
A UML::Activity may have UML::ActivityParameter(s) representing its input and output. To enable specialization of a product property(ies) it is necessary to be able to have new input data used during the property(ies) instantiation(s). This data will be assigned to the property, thus overriding the initially defined instantiation. PFM::Specialization_Entry is a stereotype to classify input UML::ActivityParameter and represents a way to “inject” data for specializing an existing property of a product.

PFM::Option

Option is an abstract stereotype representing customization methods. It has a property ***definition*** of type string that stores the definition of the customization, such as “A new part of type X is added to meet the requirement R.1”.

PFM::Extension

We defined extension as one way of customization, meaning that new property(ies) are added to a product. Adding a new property can be achieved through the use of either the DEXML::CreateEntityAction or UML::CallBehaviorAction. These two mechanisms are available to us to represent creation and instantiation of new properties in a DEXML business object definition.

PFM::Specialization

Specialization is a way of customization where existing properties are constrained or replaced. In DEXML, assignments of property values are represented with a UML::ObjectFlow between parameters. The *definition* property will be used to store any constraint.

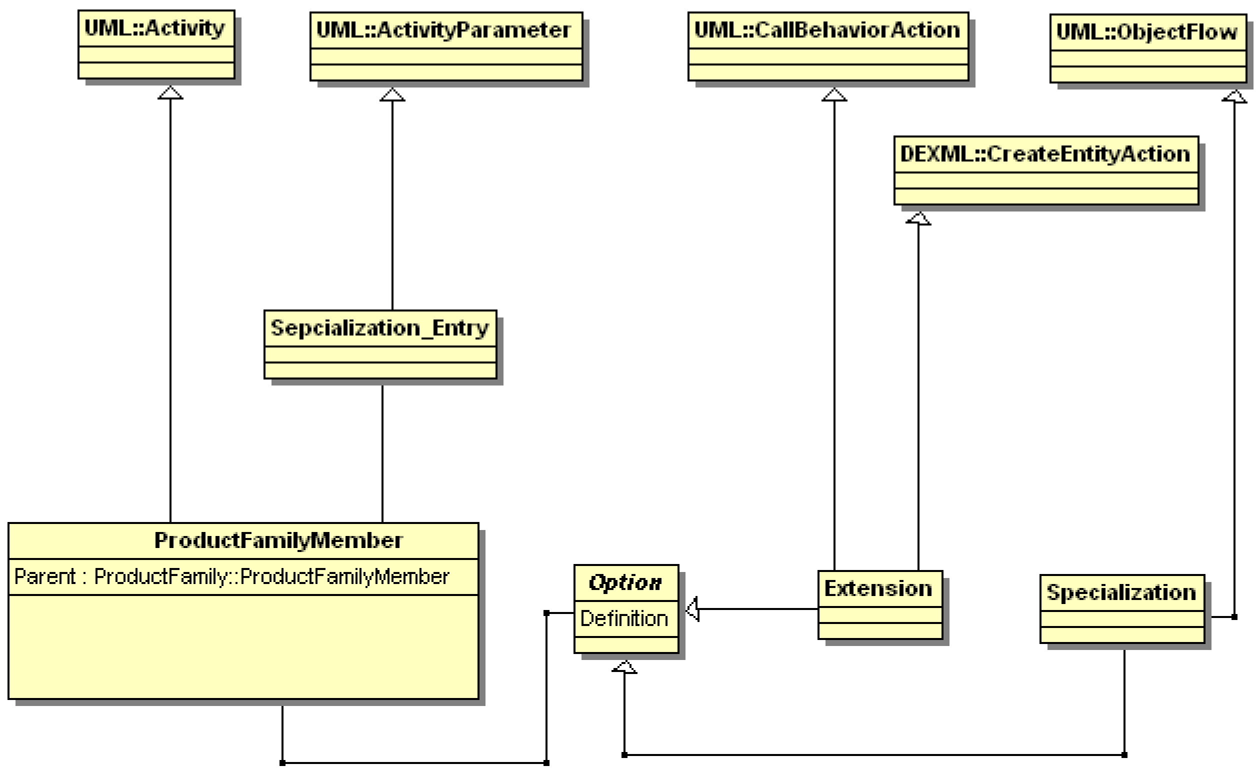


Figure 4-11 Product Family Management (PFM) profile metamodel

4.7.4 Car example

In this section we describe how a 3-member product family can be implemented using the UML profile introduced in the previous section. We intentionally do not present the DEXML graphical representation of this family because it is not our focus and does not reflect the semantics of the family model we developed. We will, however, refer to DEXML components introduced in Section 4.5. This example shows how to use the previous product family profile through its instantiation to represent a family. Our initial assumptions are:

- we start with an initial product, a Car, which we need to customize into 2 new products, a sportive version and a luxury version
- the difference between the initial car and its sportive variation is the power of the engine
- the difference between the initial car and its luxury variation is a set of options
- we already have a DEXML representation and diagram of the Car and its variations
- both variations use a UML::CallBehaviorAction to represent the instantiation of a Car, a UML::Activity from DEXML, which is customized

In the rest of this section we will describe what semantic connections from the product family profile we need to create the semantic relationship between the car and its two variations.

Family members hierarchy

The first step is to create the hierarchy between the different members of the family in a way that Car represents the root and the sport car and luxury one its variations. Because all 3 products are DEXML business objects they are instances of UML::Activity and can be specialized as PFM::ProductFamilyMember. We then use the attribute ***Parent*** of the PFM::ProductFamilyMember to represent the hierarchy between the 3 cars. The sport and luxury variations ***Parent*** attribute will refer to the initial car product.

Variation by extension

One way of customizing a product family member is by creating an extension of an existing product. Extension is achieved by adding new properties, parts or options to a product. DEXML uses the UML::CallBehaviorAction and DEXML::CreateEntityAction modeling elements to add new information components (properties, parts, …) to an object. In our case, we need to classify those new information components as extensions of the parent product of the current product. This is the role of the PFM::Extension stereotype.

In our car example, we decided to consider the luxury car as an extension because it adds several options. Initially the LuxuryCar UML::Activity has an Options DEXML::CreateEntityAction to represent the instantiations of its different options. This DEXML::CreateEntityAction has to be stereotyped to be understood as an extension of the LuxuryCar PFM::ProductFamilyMember ***Parent***.

Variation by specialization

The second customization mechanism, specialization, works by constraining existing properties of a product, at the variation level. To do so, we need to (1) specify constrainable properties at the product level, which is achieved by classifying them with the PFM::Specialization_Entry stereotype, and (2) override properties at the product variation level, which is achieved by classifying them with the PFM::Specialization stereotype.

In the car example, we decided to consider the sport car as a specialization. It requires a certain horsepower, which can be viewed as a constraint implemented with a PFM::Specialization. First we use the PFM::Specizaliton_Entry to classify the HorsePower property of the car as a constrainable property. The SportCar HorsePower property is classified as a PFM::Specialization. Both the specialization and the specialization entry are connected together through an UML::ObjectFlow to represent one overriding the other.

The following Figure 4-12 and Figure 4-13 show the Car family as DEXML metamodel instantiation, lacking of semantics, and as a PFM metamodel with hierarchical relationships and semantics of customization.

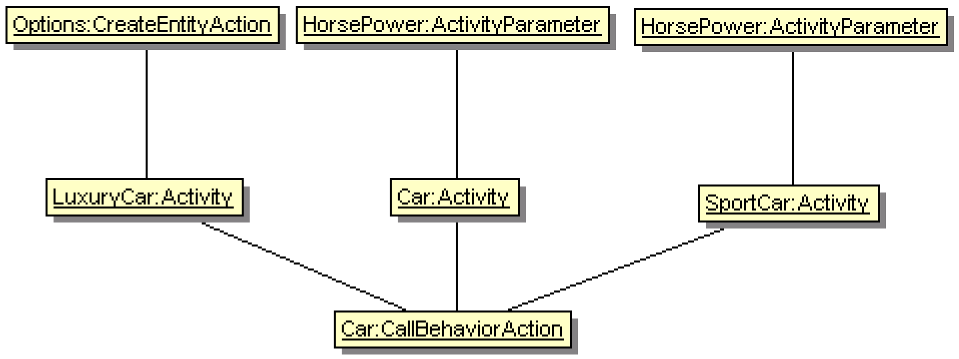

Options:CreateEntityAction
HorsePower:ActivityParameter
HorsePower:ActivityParameter
LuxuryCar:Activity
Car:Activity
SportCar:Activity
Car:CallBehaviorAction

Figure 4-12 Car Family DEXML metamodel instantiation

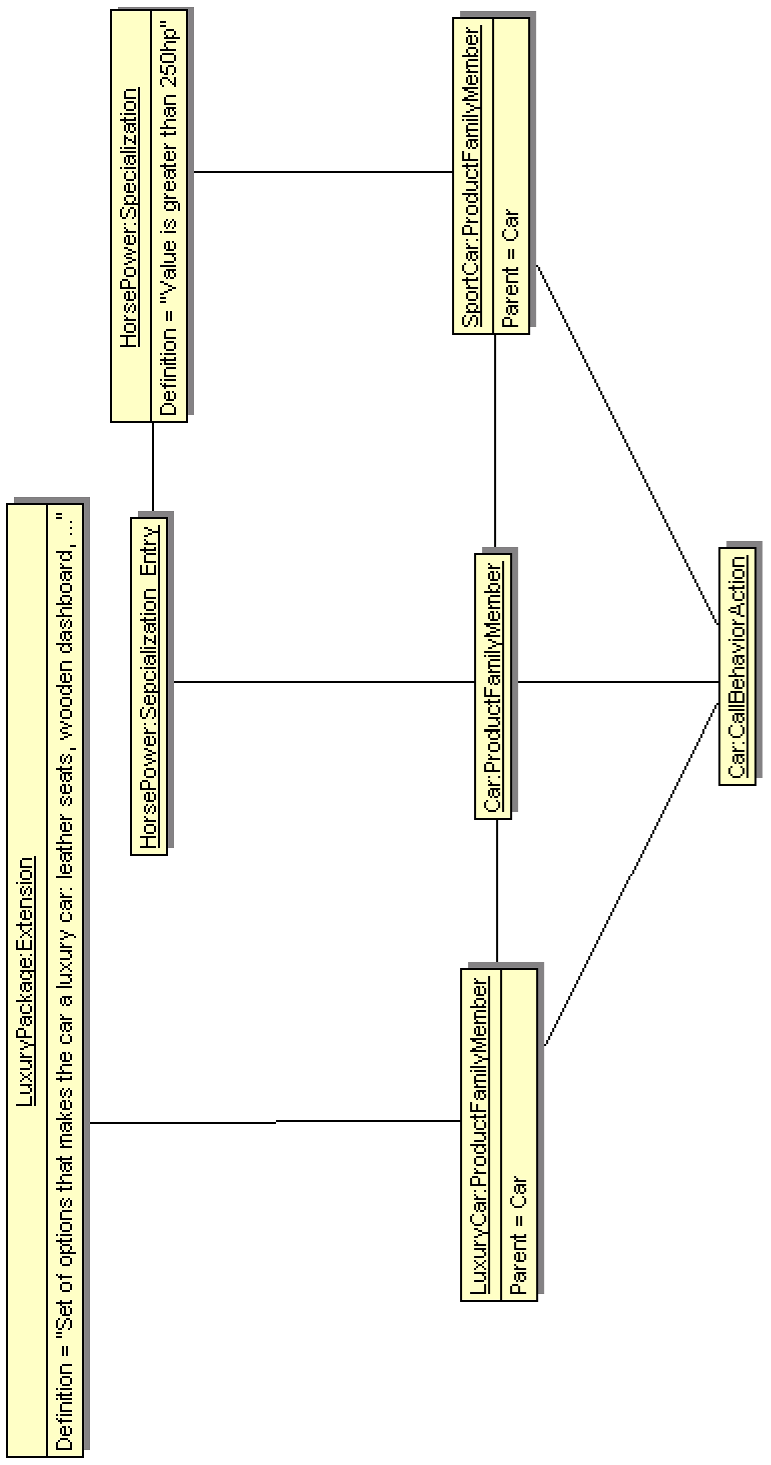

LuxuryPackage:Extension
Definition = "Set of options that makes the car a luxury car: leather seats, wooden dashboard, ..."
HorsePower:Specialization
Definition = "Value is greater than 250hp"
HorsePower:Sepcialization_Entry
SportCar:ProductFamilyMember
Parent = Car
Car:ProductFamilyMember
LuxuryCar:ProductFamilyMember
Parent = Car
Car:CallBehaviorAction

Figure 4-13 Car family PFM metamodel instantiation

4.7.5 Summary

In this section, we highlighted that specialization and extension are not information model customization mechanisms but very generic concepts that are also present in product customization. Product customization modeling requires a very specific set of modeling features that are crucial in a product-centric framework such as a PLM framework. Information modeling languages typically do not have such features. To overcome this, we benefited from the DEXML work previously done and the extensibility mechanism offered by UML, the profile. We created a UML profile, which extends both UML and DEXML, to enable the semantic representation of a product and its variations, known as a product family. This profile supports the semantic representation of hierarchical links between products, specialization and extension of products.

4.8 Conclusion

Chapter 3 demonstrated that no possible combination of components from the different existing frameworks could satisfy all our PLM-related requirements. Consequently, in this chapter we described straightforward modifications to components from those frameworks or entirely new components that fully meet those requirements. Section 2 described an implementation mechanism to design a specializable information model using an external controlled vocabulary. This mechanism is an information model itself, called *the embryo*, that needs to be extended with specific needs. Section 3 discussed the motivation for using OWL and ontologies to represent the vocabulary used by *the embryo.* It also identified validation - consistency and constraint checking - as a major benefit of using OWL. Section 4 explained the reason why we are able to reuse an existing approach from the OASIS PLCS to control customizations. Section 5 discussed how to improve our previous UML-based approach for representing business objects to support derived attributes. Section 6 provided three possible solutions for expressing and executing business rules. Section 7 defined a new UML profile, as a DEXML extension, to classify DEXML objects as products and aggregate them into product families with customizations.

Even though this chapter provides a solution to fully meet requirements from 3.2, we only responded individually to all the requirements. Section 6 raised the importance of the different components to be technologically aligned to enable them to collaborate. The next chapter will discuss this issue and provide a new solution to bring homogeneity within the framework and to integrate the information definition and information representation, respectively UML and OWL.

5 The integrated PLM framework

Chapter 4 described a set of components that meet individual requirements from Chapter 3. In 4.6 we raised a key concern related to the integration of these components into a single PLM framework. Components described in Chapter 4 all share a common purpose, they are about: information definition (see 4.2 and 4.5), specialization definition (see 4.4) and constraint definition (see 4.3 and 4.6). To achieve the necessary integration, the definitions built from individual components must be visible to all of them in one, single view. Achieving this integration can be difficult because each component has a number of available modeling techniques and languages, which includes UML class diagrams, UML activity diagrams, XML Schemas, OWL ontologies, SPARQL queries, and SPIN constraints. Each of these languages has its own representation and sense of meaning, leading to possible inconsistencies during integration. The easiest way to overcome these inconsistencies and simplify the integration is to select a unique modeling language for every component within the framework.

In Chapters 3 and 4 we implicitly compared the available modeling languages and decided that OWL is the best technical candidate because of the different levels of validation it supports. Unfortunately, OWL is not the best candidate, for implementing our framework, from the human perspective. We say this because, OWL is the least mature of the commonly used modeling languages. Consequently, it has fewer development tools, less market penetration, and less use than the other two (UML and XML). This is important since information modelers must have confidence in and experience with the selected tools in order for them to accept this framework. In this chapter we propose a trade-off between the maturity of the languages and the technical requirements of our integrated framework.

5.1 Information model and data representation

In 4.2 we introduced the *embryo*, a first step toward developing an extensible information model. This embryo was shown using a UML class diagram. UML is probably one of the easiest and most used languages for information modeling and software design. It provides an object diagram designed to represent instances of class diagrams graphically; but this graphical diagram can be used to support data exchange. XML schema, on the other hand, is widely used both for data representation and data exchange. It also enables users to maintain a certain level of consistency.

Another commonly used, PLM-related information modeling language is EXPRESS. EXPRESS was designed by the STEP community for representing product models and exchanging product data. The use of EXPRESS, however, has declined dramatically since its introduction more than

20 years ago. Even at the height of its popularity, EXPRESS was rarely used outside of the product-modeling domain.

In summary, each of the available modeling languages has strengths and weaknesses. Our challenge is to find the best trade-off among them - a trade-off between the validation strengths, which are crucial to our work, and popularity strengths. This trade-off can be achieved by giving users the freedom to choose the languages they want and provide them a transformation mechanism from the chosen language to OWL. In (Ferdinand et al., 2004; Gasevic et al., 2004), the authors present methods for transforming (sometimes called mapping) UML and XML models into OWL ontologies. Such mechanisms could be used whenever a model validation is needed. Until recently, such transformation mechanisms did not exist for EXPRESS.

5.1.1 EXPRESS to OWL mapping efforts

Two major efforts were initiated to develop such a mechanism (Klein et al., 2008; Zhao & Liu, 2008). The first is the intelligent Self- describing Technical and Environmental Networks (S-TEN) project. S-TEN was funded by the European Community to develop bi-directional translations between EXPRESS and OWL. S-TEN focuses on translating modules that can be used within several STEP APs. Hence, no AP is covered in full. The STEP modules are also modified either to take advantage of OWL or as an improvement. Moreover new capabilities were added, such as a better management of the product identifiers. A manual check is performed after the translation to ensure that the meaning of the data models has not changed. The final ontology is stored in a database.

Zhao and Liu proposed a method to represent EXPRESS models in OWL and SWRL, a rule-based language for OWL. Their method translates procedural code contained in the EXPRESS schemas. That code specifies algorithms that 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[8] rules to represent EXPRESS procedures and functions. However, it is not clear whether this mapping between procedures and Jess rules will work for all the procedures. Moreover, some aspects of the EXPRESS language are not properly translated. For instance, the translation of EXPRESS ordered lists was not proposed. Automated translation tools are planned, but we are not aware of any software releases of those tools.

5.1.2 Our Mapping

5.1.2.1 Mapping the main concepts

[8] http://www.jessrules.com/

In our translation, which we call OntoSTEP, we map EXPRESS entities and instances respectively to OWL classes and individuals. EXPRESS 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 express explicitly 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 (Fiorentini et al., 2008). The following table summarizes our proposed translation of the basic concepts from EXPRESS to OWL.

Table 5-1 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	The range of the property is restricted to have, for that

an aggregation type	property, minimum and maximum cardinalities corresponding to the aggregation size.

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

5.1.2.2 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. In the following section we refer to these data files as "Part 21 files."

The translation to OWL is similar to the process described in the previous section and summarized in Table 5-1. 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.

While STEP considers all instances to be different, OWL does not have the unique name assumption. This means that in OWL the same object can be identified with two different names. One way to capture the EXPRESS semantics, then, is to declare all the created OWL individuals as different.

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. As we discussed earlier, 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 absence 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, an explicit declaration that this instance of product has no id must be made.

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

5.1.2.3 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. OWL, since 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. Some EXPRESS concepts, such as SELECT, ENUMERATION and UNIQUE are all translated into OWL and the details of the translation are provided in (Krima et al., 2009).

EXPRESS defines simple data types to capture all common product information. OWL inherits the data types defined in the XML Schema Definition (XSD) language. In EXPRESS, some types, like Boolean and string, have the exact equivalent in OWL. Other types, like number and real, are represented in a slightly different way in OWL. For example, we map the real data type in EXPRESS into 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. The translation of the logical and binary data types is outside the scope of this research, since we have no interest in these data types.

EXPRESS allows the creation of data types derived from the simple types previously presented. In order to deal with these derived types in OWL, we build a type hierarchy and we apply the concept of data wrapping. For 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 map all of the user-defined data types related to string (Label in this case). This concept organization allows 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, analysis of correctness is required.

EXPRESS provides four different types of aggregations: 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 in the aggregation are defined. The detailed mapping of these types are explained in (Krima et al., 2009). 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. Because object properties in OWL do not allow duplications, we create the concept structure shown in the below OWL code. 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.

```
:Container a owl:Class.
:Bag a owl:Class.
:Content a owl:Class

:hasBag a rdf:Property;
     rdfs:domain :Contaner;
     rdfs:range :Bag.

:hasContent a rdf:Property;
     rdfs:domain :Bag;
     rdfs:range :Content.
```

Figure 5-1 Bag implementation with OntoSTEP

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. 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 OWA-related ontology, such as OWL, 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 subtype 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 is declared as an instance of the supertype class and not an instance of the subtype class, 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. For these reasons, we ignore the ABSTRACT keyword. It is then impossible for the reasoner to check that abstract entities are not directly instantiated.

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 as parameter a list of entities 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 as all disjoint. Another solution could be to use the logical definition of XOR: we could use in OWL the intersection, the union, and the complement to translate and, or, and not. However, this increases the complexity of the ontology, since 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: the instance can belong to more than one subclass. In OWL a set of entities joined by an ANDOR is translated by a union of the corresponding classes in OWL. First, we represent the union of the subclasses by using the ObjectUnionOf construct; we then declare this union to be equivalent to the parent class.

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

5.1.3 Embryo in EXPRESS

In this section we descried our new EXPRESS to OWL mapping. This mapping is required to help the EXPRESS modelers who would use the embryo. We also developed an EXPRESS schema-based implementation of the embryo.

```
(*
      Author: KRIMA Sylvere
      Schema name: Embryo
      Version: 0.1

*)
SCHEMA embryo_v_0.1;

(* for readability we do not include all the possible types *)

TYPE PropertyType = ENUMERATION OF(
      xsd:double,
      xsd:string,
      ...
);
END TYPE;

ENTITY Unit
      UnitOf: SET OF Dynamic_Property := [];
      hasMeaning: SET OF Classification := [];
END_ENTITY;

ENTITY Dynamic_Property
      name: STRING;
      property_type: PropertyType;
```

```
     hasDefinition: Classification;
     propertyOf: SET OF Classification := [];
     hasValue: Value;
END_ENTITY;

ENTITY Classification
     hasProperty: SET OF Dynamic_Property := [];
     defines: SET OF Dynamic_Property := [];
     meaningOf: Unit;
     classificationOf: SET OF Classifiable_Concept := [];
     hasClassifier: OWL_Class;
END_ENTITY;

ENTITY OWL_Class
     name:STRING;
     classOf: Ontology;
     classifies: Classification;
END_ENTITY;

ENTITY Ontology
     URI:STRING;
     hasClass: SET OF OWL_Class := [];
END_ENTITY;

ENTITY Classifiable_Concept
     ABSTRACT SUPERTYPE;
     hasClassification: SET OF Classification := [];
END_ENTITY;

ENTITY Value
     ABSTRACT SUPERTYPE;
     valueOf: Dynamic_Property;
END_ENTITY;

ENTITY BooleanValue
     SUBTYPE OF (Value);
     value: BOOLEAN;
END_ENTITY;

ENTITY StringValue
     SUBTYPE OF (Value);
     value: STRING;
END_ENTITY;

(* for readability we do not include all the possible value type*)
```

```
END_SCHEMA;
```

5.2 Business objects representation

The previous section was about representing information models and data so that OWL-based validation mechanisms can be applied. One goal of this chapter is to enable these validation mechanisms to be applied to everything that is not initially designed using OWL. This challenge first appeared in 4.6 where business rules were defined. Recall that business rules and business objects must be designed using compatible or identical technologies. The solutions in 4.5 and 4.6 respectively were based on DEXML (specifically UML activity diagrams) and OWL/SPARQL/SPIN. These two solutions are neither compatible nor identical. While this might look like an inconsistency, technologies are not used at the same level. Business objects are conceptually defined with UML, but must be implemented physically with OWL. The missing link, then is a mapping that generates the implementation from the definition.

5.2.1 A DEXM-to-OWL mapping for business objects

We noted in 5.1 that mechanisms exist to map UML class diagrams into OWL ontologies. That is not our goal here. Our goal is to map the DEXML representation of business objects, which is based on the UML activity diagram, to an OWL-based representation.

DEXML builds conceptual models of business objects to define how atomic concepts are aggregated together to represent more complex concepts. In the object-oriented paradigm, these complex concepts are objects and the atomic concepts used to build them are their attributes. Since OWL is based on description logic it is not object oriented. Hence, it is not a native solution for representing business objects (Koide, Aasman, & Haflich, 2005); but, it has enough features to represent objects structurally. Moreover, there is the SPIN technology that brings an object-oriented flavor to OWL (Knublauch et al., 2011) and enables us to represent the behavioral aspects of objects. Our solution is to combine OWL and SPIN. We describe how we implement that solution below.

The structural aspect of an object is defined in DEXML by the different CreateEntityAction(s) involved in the conceptual model. The output parameters of the Activity representing a business object are its attributes. OWL by itself has enough mechanisms to describe the structural aspect of an object because it allows us to define classes and attributes using object properties and data properties.

The limitations appear when describing the behavioral aspect of an object. Regarding the behavioral aspect, DEXML defines how an instance of a business object is built, including the instantiations of its attributes and the values that they have (Krima et al., 2011). OWL does not allow us to enrich class definitions with operations executed when instantiating an object. Fortunately, SPIN provides a workaround with its rule mechanism that enables users to attach rules, written with SPARQL, to a class definition as seen in 4.6. This solution can validate the different instantiations and assignments required during the creation of a business object and be executed by the SPIN engine.

The mapping between these 2 languages, DEXML and OWL, needs to (1) map the business object itself and its attributes and (2) validate the different instantiations and assignments of these attributes. The mapping is summarized in Table 5-2 DEXML to OWL mapping.

Table 5-2 DEXML to OWL mapping

Aspect	DEXML	OWL/SPARQL/SPIN
Structural	UML Activity representing a business object	OWL class
Structural	Output parameter	OWL property
Behaviroal	Object flow from CreateEntityAction to another	SPIN:rule using SPARQL and attached to the class

Figure 5-2 and the following OWL code (in n3 notation) show an example where a DEXML business object representing a bike is mapped to its OWL representation. The figure is an extract of the DEXML business object representing the bike.

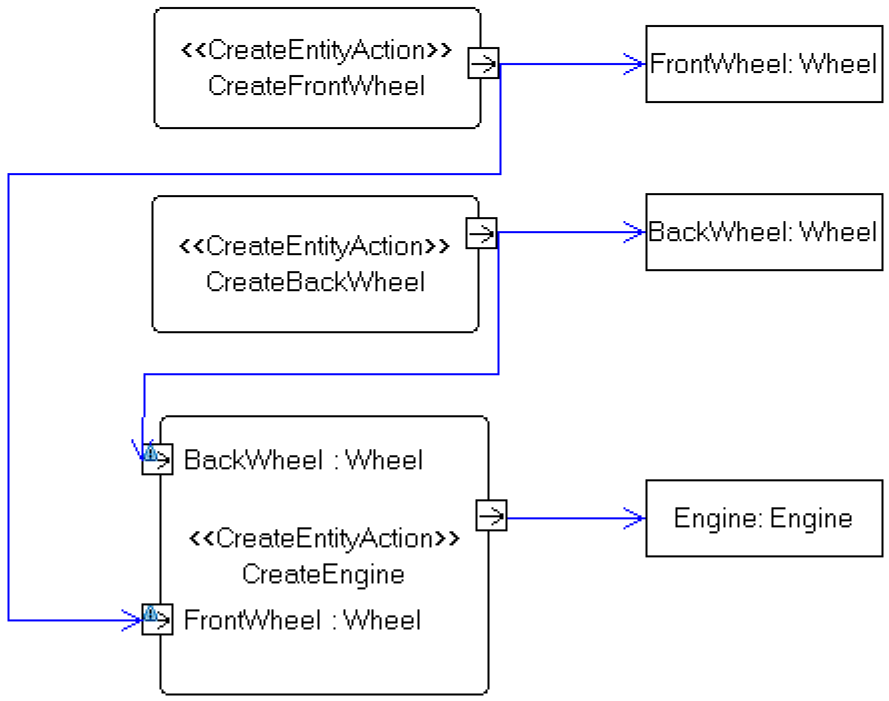


Figure 5-2 DEXML Bike business object

```
:Bike a rdfs:Class.
:Wheel a rdfs:Class.
:Engine a rdfs:Class.

:hasFrontWheel a rdf:Property;
     rdfs:domain :Bike;
     rdfs:range :Wheel.

:hasBackWheel a rdf:Property;
     rdfs:domain :Bike;
     rdfs:range :Wheel.

:hasEngine a rdf:Property;
     rdfs:domain :Bike;
     rdfs:range :Engine.
```

The bike has two wheels and one engine, and the engine is connected to the two wheels once they are created. After manually performing the mapping we obtain (1) the same business object as an OWL class with a spin:rule that connects the two wheels to the engine, and (2) three object properties that enable users to assign the wheels and an engine to the bike. When running the SPIN engine on the ontology that contains this business object and its instances, assignment between wheels and engine will be checked together.

The following rule expressed in SPIN is an example of a rule that checks if the engine is correctly connected to the wheels.

```
CONSTRUCT {
    _:b0 a spin:ConstraintViolation .
    _:b0 spin:violationRoot ?this .
    _:b0 spin:violationPath :hasEngine .
    _:b0 rdfs:label "Wheels of the engine should be the wheels of the
Bike" .
}
WHERE {
      ?this :hasFrontWheel ?w1.
      ?this :hasFrontWheel ?w2.
      ?this :hasEngine ?e1.
      ?e1 :hasWheel ?ew1;
           :hasWheel ?ew2.
      ?ew1 owl:differentFrom ?ew2.
}
```

We define business objects as domain-specific concepts. These concepts are initially introduced in the external controlled vocabulary used for specialization. Therefore, the DEXM-to-OWL mapping enriches the specialization with semantics of the concepts that are mapped. Any business object instance should then be seen as an instance of a concept from the controlled vocabulary. For this to happen, we need to make sure that there is one and only one output parameter in each DEXML definition whose name is identical to the name of the business object. For example, when we defined a business object Wheel, we needed to make sure that there is an output parameter whose name is Wheel. This output parameter should be ignored during the DEXML-to-OWL mapping because it represents a generic concept. That generic concept will be classified later as a domain-specific concept that will be the reference to the instance of the business object. An example is given in the following section.

5.3 Use case

In this section we will go step-by-step through an entire use case. In doing so, we will demonstrate the highlights and the benefits of our framework. The use case involves a manufacturing company that needs (1) an information model to represent information about all the parts it designs, engineers, and manufactures, and (2) to validate this information model. In general, it is impossible for such companies to create an information model that supports all the parts they might be able to make and sell. How would such a company proceed?

5.3.1 Using the embryo to create an information model

The easiest way to proceed is to use our *embryo* and build customizable information models for new parts as needed. The first step is to create a generic information model that extends the *embryo* and build the following model (see Figure 5-3 for the equivalent UML class diagram) in EXPRESS.

```
 (*
      Author: Manufacturing_Company
      Schema name: ManufacturedPart
*)
SCHEMA ManufacturedPart;

USE FROM embryo_v_0.1;

ENTITY Part
      SUBTYPE OF (Classifiable_concept);
      partOf: Part;
      hasAssembly: SET OF Part := [];
END_ENTITY;

END_SCHEMA;
```

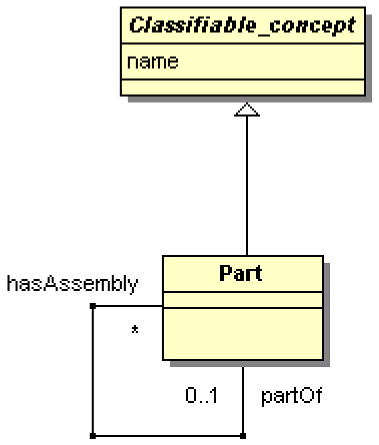


Figure 5-3 UML class diagram of embryo extension for manufactured parts

In this model, the Part concept is a subtype of Classifiable_concept because Part is generic and this relation will enable the company to specialize any instance of Part. The company also added a self-relationship hasAssembly/partOf for representing assembly structures.

The second step is to create the external controlled vocabulary where we define the different types of parts that can be created and used to specialize instances of Part. We (1) use OntoSTEP to create a new ontology from the *embryo* schema, (2) use OntoSTEP to create a new ontology from the ManufacturedPart schema, (3) manually create a new ontology representing the external controlled vocabulary (or Reference Data Library) and specialization types that can be applied to instances of Part. This process is summarized in Figure 5-4. In this context, the company decides to enrich the RDL with concepts that they will use when manufacturing a car. Figure 5-5 shows how the company represents a car: a Car is an assembly of an Engine and a Body, this Engine is built with two Axles and each Axle is built with two Wheels. The RDL is then enriched with these domain- specific concepts: Car, Body, Engine, Axle and Wheel. These five concepts are business objects. The next step is to define, using DEXML, how these business objects are instantiated.

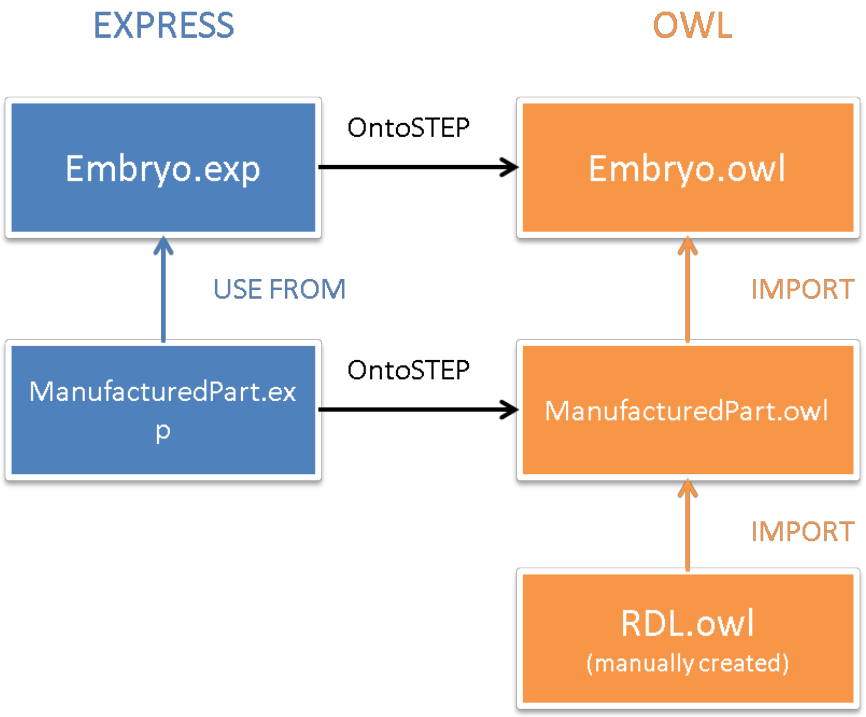


Figure 5-4 Role of OntoSTEP in our framework

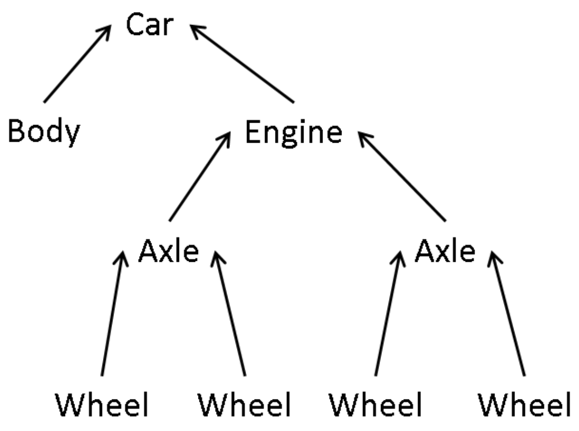


Figure 5-5 Car assembly structure

5.3.2 Instantiating the Business objects

Now that the business objects have been identified we must define their instantiations formally using DEXML, with atomic and generic concepts from the *embryo* and the ManufacturedPart schema. To use DEXML we need to provide a UML representation of the schemes that contain atomic concepts. To do this, we used Reeper[9], a tool that transforms an EXPRESS schema to a UML class diagram (see Figure 5-6). The next step is to create a new DEXML definition, and import the newly created UML class diagrams so that we can formalize the business objects instantiation.

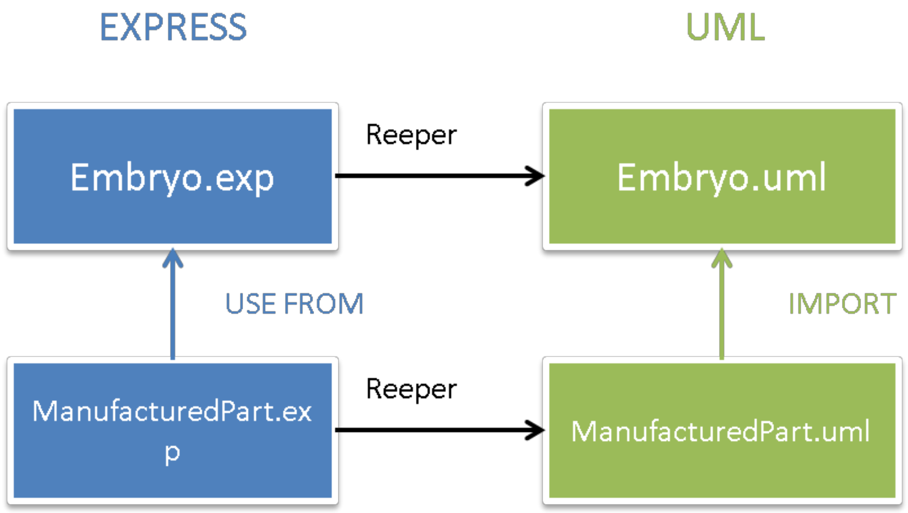


Figure 5-6 Role of Reeper in our framework

Here we will describe how to instantiate the business object Wheel. According to 4.2, to be classified as an instance of Wheel, an instance of Part needs to be connected to an instance of Classification. This instance of Classification is then connected to an instance of an OWL_class whose name attribute is “Wheel”. The instance of OWL_class must be connected to an instance of Ontology whose URI attribute is the URI of the RDL where the Wheel concept is defined. From this definition, an instantiation of Wheel should be as follows:

```
#1 = Ontology('http://mycompany/part/RDL.owl',);
#2 = OWL_class('Wheel',#1,);
#3 = Part('FrontLeftWheel',,);
#4 = Classification(,,,(#3),#2);
```

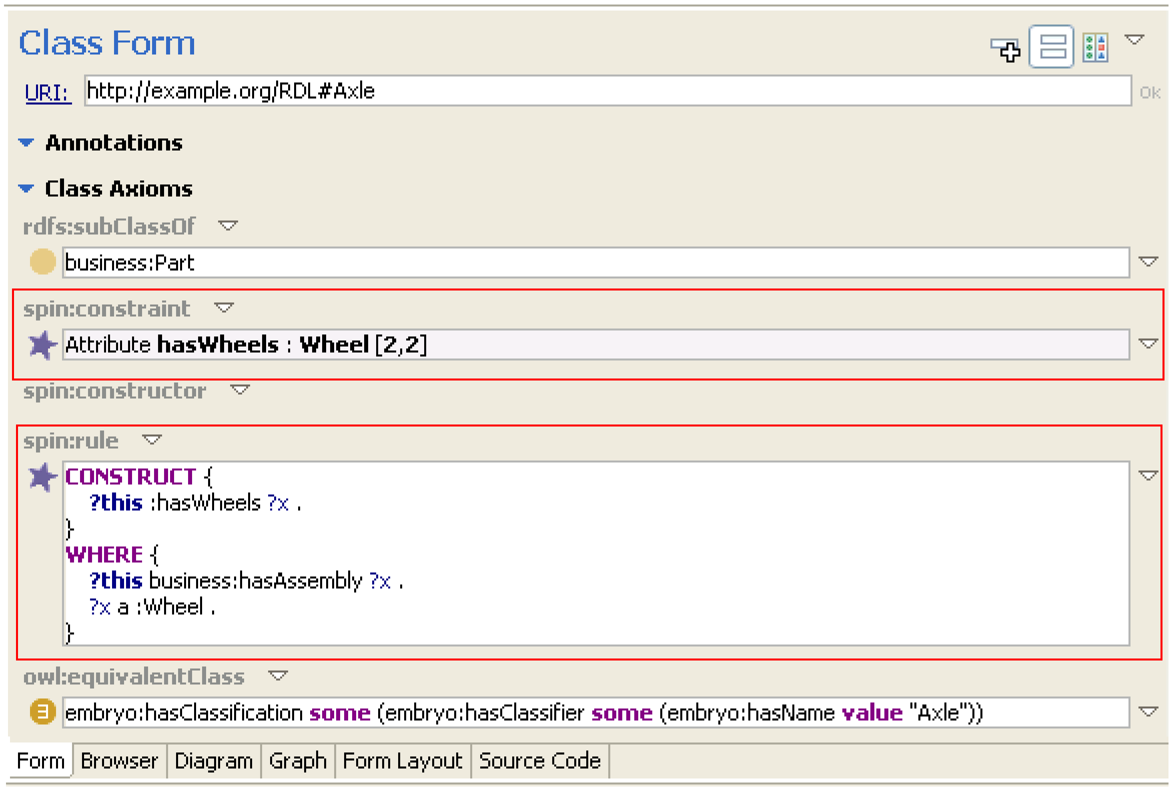
In 5.3 we also learned how to use the DEXML definitions to enrich the RDL. DEXML definitions help to derive attributes of the business objects automatically. From these definitions we can also manually enrich the RDL by adding business rules when necessary. One rule that is **absolutely necessary** is the rule that enables us to automatically and dynamically classify instances of Classifiable_concept, and its subclasses, as instances of business objects. In OWL, we can say if x1 is an instance of X, we can define conditions that, if met, classify x1 as

[9] http://www.nist.gov/el/msid/reeper.cfm

an instance of Y. This means we are able to define business rules such as "an instance of Part could be classified as an instance of Wheel" using the following OWL code:

```
@prefix business <http://mycompany.org/concepts#>.
@prefix embryo <http://sylvere/framework#>.

:Wheel a rdfs:Class;
      rdfs:subClassOf business:Part;
      rdfs:subClassOf
      [a owl:Restriction;
            owl:onProperty embryo:hasClassification;
            owl:someValuesFrom
            [a owl:Restriction;
                  owl:onProperty embryo:hasClassifier;
                  owl:someValuesFrom
                  [a owl:Restriction;
                        owl:onProperty embryo:hasName;
                        owl:hasValue
Wheel^^<http://www.w3.org/2001/XMLSchema#string>
                  ].
            ].
      ].
```

This dynamic classification can be performed by a reasoner, if the instances are represented in OWL. In this example, the transformation of the instances from EXPRESS to OWL is done using OntoSTEP. The manufacturing company also adds a business rule to instantiate the new attribute Wheels of Axle. This attribute represents the two wheels to which the axle is connected. These wheels are instances of Part classified as Wheel and they are already connected to the Axle through the Assembly attribute. One more constraint is added to make sure that an instance of Axle has two and only two Wheels. Figure 5-7 shows how to represent this rule.



```
URI: http://example.org/RDL#Axle
rdfs:subClassOf
    business:Part
spin:constraint
    Attribute hasWheels : Wheel [2,2]
spin:constructor
spin:rule
    CONSTRUCT {
        ?this :hasWheels ?x .
    }
    WHERE {
        ?this business:hasAssembly ?x .
        ?x a :Wheel .
    }
owl:equivalentClass
    embryo:hasClassification some (embryo:hasClassifier some (embryo:hasName value "Axle"))
```

Figure 5-7 Adding OWL and SPIN business rules to a business object

Figure 5-8 summarizes the different steps we went through since the beginning of this use case. Domain experts use the product data modeling language EXPRESS to represent their information models, which then convert to: 1) UML for graphical representation and benefit from advanced tooling 2) OWL for computation power and reasoning capabilities that we use for customization.

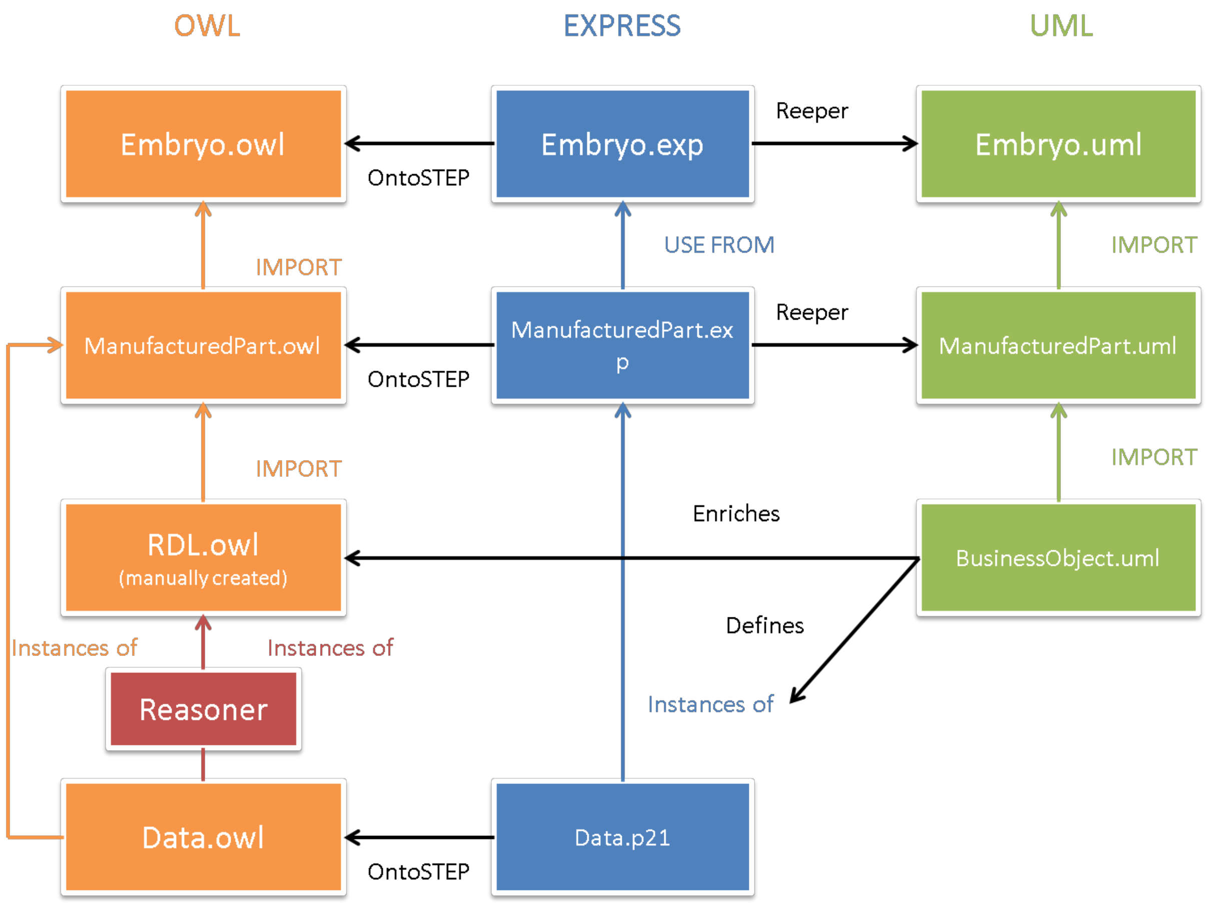
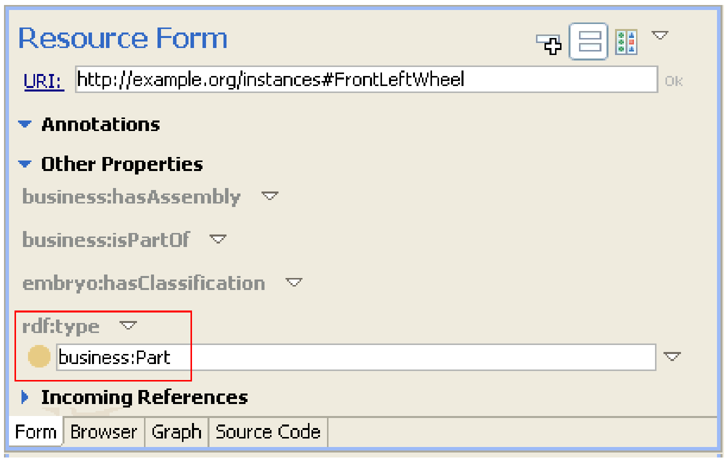


Figure 5-8 Full steps to benefit from the embryo

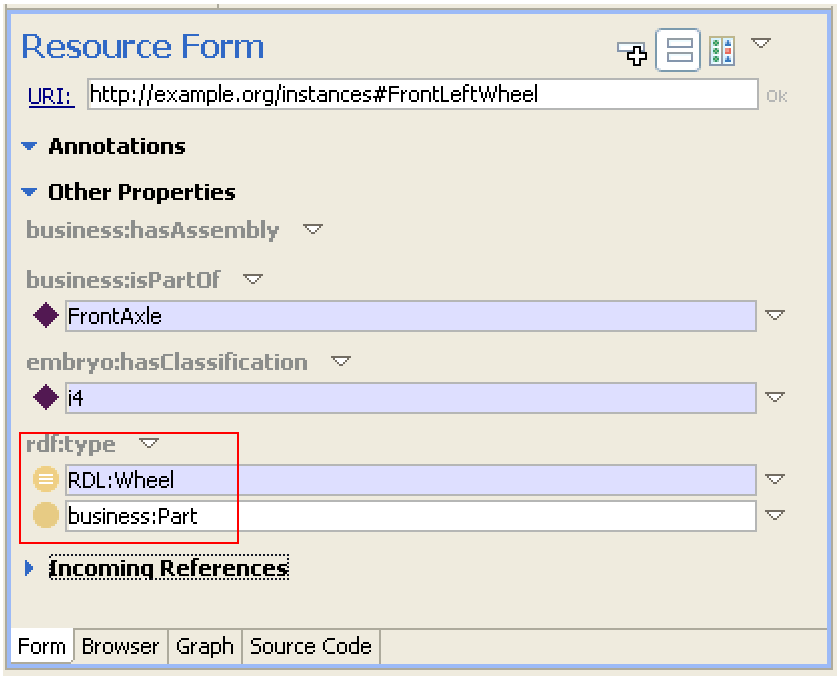
Using a reasoner such as Pellet and the rule defined in Figure 5-7 we can see that the instance of Part we defined previously in EXPRESS is first mapped to OWL. Then, still as an instance of Part (see Figure 5-9), but after reasoning, this same instance is classified as a Wheel by the reasoner (see Figure 5-10).



URI: http://example.org/instances#FrontLeftWheel

- Annotations
- Other Properties
  - business:hasAssembly
  - business:isPartOf
  - embryo:hasClassification
  - rdf:type
    - business:Part
- Incoming References

Figure 5-9 OWL instance before classification


Resource Form

URI: http://example.org/instances#FrontLeftWheel

Annotations

Other Properties

business:hasAssembly

business:isPartOf

FrontAxle

embryo:hasClassification

i4

rdf:type

RDL:Wheel

business:Part

Incoming References

Figure 5-10 OWL instance after classification

The reasoner has been able to infer that this instance of Part is also an instance of Wheel. The next step is to make sure that we are able to execute all required business rules on inferred instances of business objects.

5.3.3 Data validation

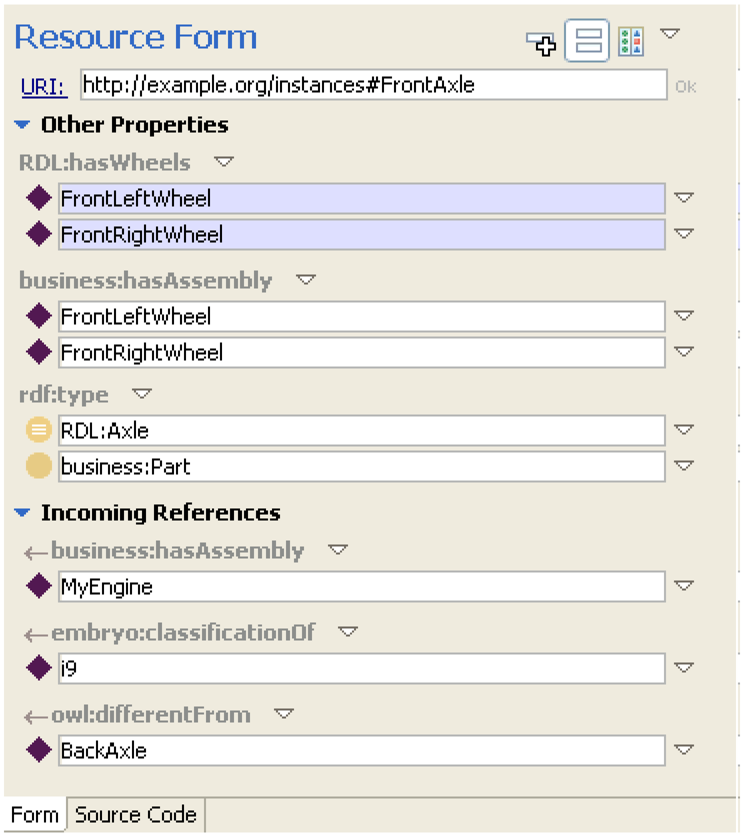
We previously saw how to create instances of business objects from instances of atomic concepts. Since we have business rules attached to business objects we need to make sure that these new instances are conform to the business rules. We will use the following EXPRESS data as input;

```
#1  = Ontology('http://mycompany/part',);
#2  = OWL_class('Wheel',#1,);

#3  = Part('FrontLeftWheel',,);
#4  = Classification(,,,(#3),#2);
#5  = Part('FrontRightWheel',,);
#6  = Classification(,,,(#5),#2);
#7  = OWL_class('Axle',#1);
#8  = Part('FrontAxle',,(#3,#5));
#9  = Classification(,,,(#8),#7);

#10  = Part('BackLeftWheel',,);
```

```
#11 = Classification(,,,(#10),#2);
#12 = Part('BackRightWheel',,);
#13 = Classification(,,,(#12),#2);
#14 = Part('BackAxle',,(#12));
#15 = Classification(,,,(#14),#7);

#16 = OWL_class('Engine,#1);
#17 = Part('MyEngine',,(#8,#14));
#18 = Classification(,,,(#16),#17);

#19 = OWL_class('Body',#1);
#20 = Part('BodyPart',,)
#21 = Classification(,,,(#19),#20);

#22 = OWL_class('Car',#1);
#23 = Part('FordFocus',,(#17,#20));
#24 = Classification(,,,(#23),#22);
```

Figure 5-7 showed the two constraints we defined on the Axle business object: one as a spin:constraint, one as a spin:rule. After transformation of the test data to OWL, classification of that data, and execution of the spin:rule we have two axles defined as shown in Figure 5-11 and Figure 5-12. We can see that the FrontAxle instance has two values of type Wheel for its hasWheels property: FrontLeftWheel and FrontRightWheel. This assignment has been made following the spin:rule. The BackAxle instance has only one value of type Wheel for its hasWheels property: BackRightWheel. This assignment has been made following the spin:rule. The next step during validation is to apply the spin:constraint that expects every Axle to have two values of type Wheel for the property hasWheels. We expect here the BackAxle instance to raise a warning.

Figure 5-13 shows what appears in the ontology editor when we turn on the constraint validation mechanism. The SPIN engine is able to identify the number of violated constraints (blue box), the instances that violates the constraint(s) (red box) and the property(ies) that violates the constraint(s) (green box).



Resource Form

URI: http://example.org/instances#FrontAxle

Other Properties

RDL:hasWheels
- FrontLeftWheel
- FrontRightWheel

business:hasAssembly
- FrontLeftWheel
- FrontRightWheel

rdf:type
- RDL:Axle
- business:Part

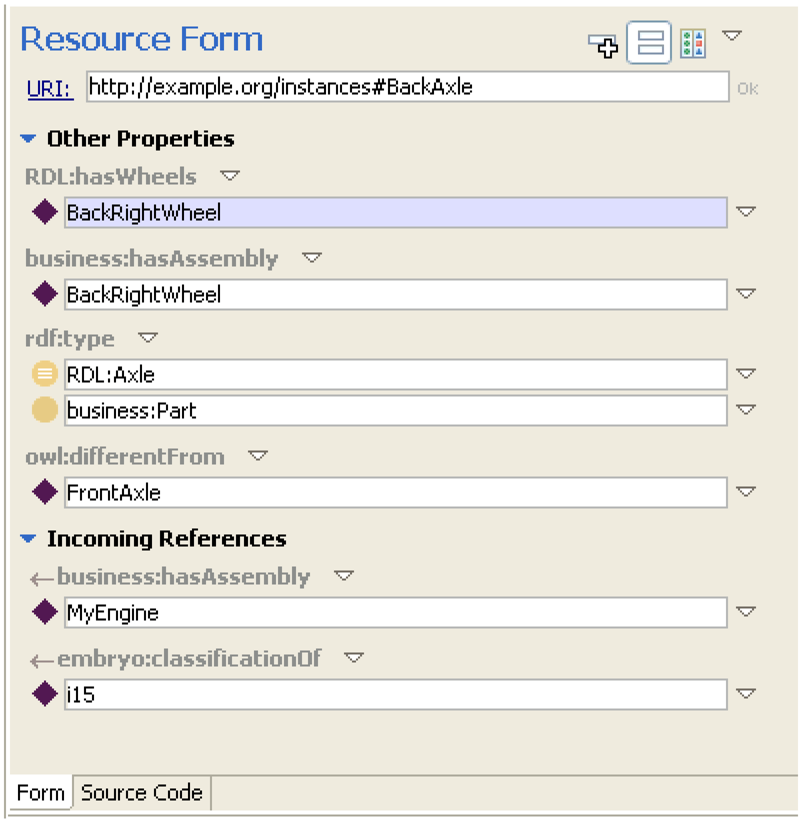
Incoming References

←business:hasAssembly
- MyEngine

←embryo:classificationOf
- i9

←owl:differentFrom
- BackAxle

Figure 5-11 Front axle instance



Resource Form

URI: http://example.org/instances#BackAxle

Other Properties

RDL:hasWheels
- BackRightWheel

business:hasAssembly
- BackRightWheel

rdf:type
- RDL:Axle
- business:Part

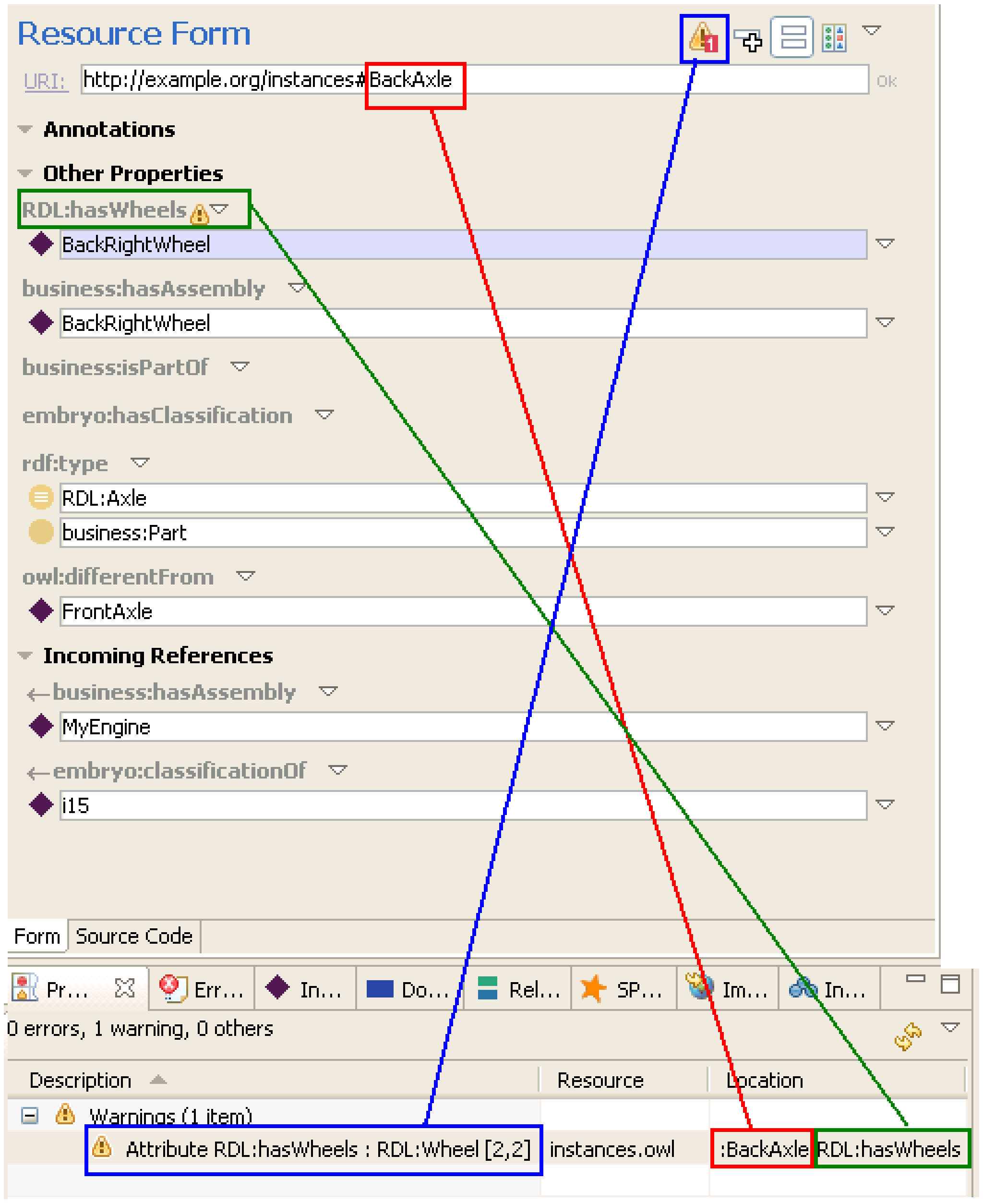
owl:differentFrom
- FrontAxle

Incoming References

←business:hasAssembly
- MyEngine

←embryo:classificationOf
- i15

Figure 5-12 BackAxle instance


Resource Form

URI: http://example.org/instances#BackAxle

Annotations

Other Properties

RDL:hasWheels
BackRightWheel

business:hasAssembly
BackRightWheel

business:isPartOf

embryo:hasClassification

rdf:type
RDL:Axle
business:Part

owl:differentFrom
FrontAxle

Incoming References

business:hasAssembly
MyEngine

embryo:classificationOf
i15

Form | Source Code

0 errors, 1 warning, 0 others

Description	Resource	Location
Warnings (1 item)		
Attribute RDL:hasWheels : RDL:Wheel [2,2]	instances.owl	:BackAxle RDL:hasWheels

Figure 5-13 Validation results and warnings for the back axle

In this section we saw two examples of business rules, implemented with two different mechanisms, and how to execute each of them using SPIN.

5.3.4 Integration of new requirements

After manufacturing parts, our company sells them to many customers. It must provide product data for each of these parts. One of these customers has a big concern about sustainability and asks our company to provide "the quantity of CO_2 produced during manufacturing" as part of the product data. This customer also wants the company to create a special category for parts whose CO_2 exceeds 10Kg. In the traditional approach, it would be necessary (1) to rewrite the original information model to include the quantity of CO_2 and (2) make major modifications to software that both the manufacturer and the customer use to input and output that quantity. Using the *embryo* approach, we do not need to rewrite the original information model and modifications to the software, if needed, will be negligible.

The first step is to be able to represent this new property that we call “QuantityOfCO2” using the dynamic property mechanism of the *embryo*. First we need to create a new concept called QuantityOfCO2 as a subconcept of Dynamic_property. This will later enable us to classify instances of Dynamic_property as instances of QuantityOfCO2, if needed. Then, to our previous input data we can add the following data to represent the quantity of CO_2 produced when manufacturing the wheels.

```
#25 = OWL_Class('QuantityOfCO2',#1);
#26 = Dynamic_Property('QuantityOfCO2FrontLeftWheel', double,,(#4), );
#27 = DoubleValue(#25, 9.81);
#28  =  Dynamic_Property('QuantityOfCO2FrontRightWheel',  double,,(#6),
);
#29 = DoubleValue(#25, 9.81);
#30  =  Dynamic_Property('QuantityOfCO2BackRightWheel',  double,,(#13),
);
#31 = DoubleValue(#25, 9.81);
#32 = Dynamic_Property('QuantityOfCO2BackLeftWheel', double,,(#11), );
#33 = DoubleValue(#25, 9.81);
#34 = Classification(,(#26,#28,#30,#32),,,#25);
```

The process is used to define the QuantityOfCO2 property of other parts of our test data. The first challenge has been met and we are able to add information to our data with no change on the information model, this means that any software that was able to read/write data conforming to the *embryo* does not need modifications at all.

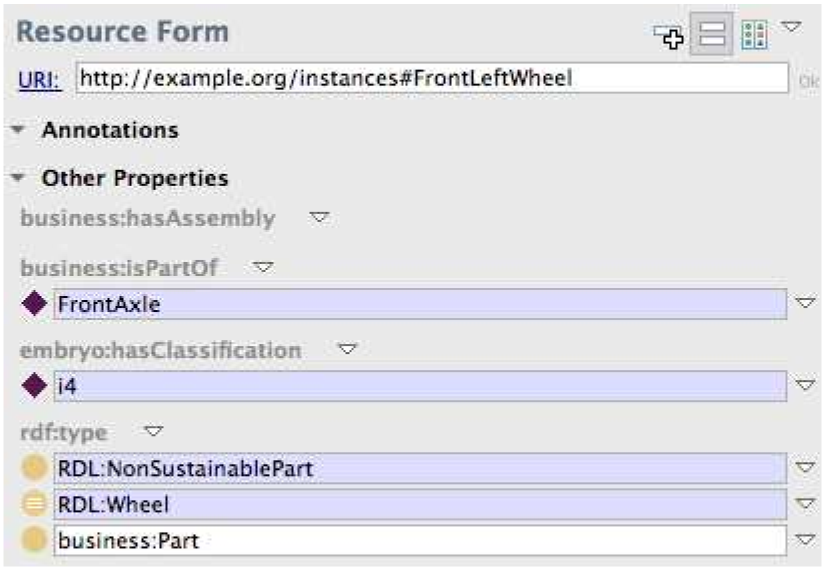
The second challenge is to be able to process the data and categorize parts whose CO_2 exceeds 10Kg. To meet this requirement we need to create a new OWL class in the RDL that will “contain” all invalid instances: we call this class “NonSustainablePart”. Then we add a rule on the Part concept so all its instances whose CO2 exceeds 10Kg are classified as NonSustainablePart (see the following SPIN rule).

```
CONSTRUCT{
      ?this a :NonSustainablePart.
}
WHERE{
      ?this embryo:hasClassification ?x.
      ?x embryo:hasProperty ?y.
      ?y a :QuantityOfCO2.
      ?y embryo:hasValue ?v.
      ?v embryo:hasDoubleValue ?dv.
      FILTER(?dv > 10.00).
}
```

As demonstration, let us replace the instance #27 of our test data by the following:

```
#27 = DoubleValue(#25, 10.05);
```

This line means that the front left wheel produces 10.05Kg of CO2 and should be classified as a NonSustainablePart. After using OntoSTEP to transform the data to OWL, we run the reasoner and the SPIN engine and get the instance of Part that represents the front left wheel classified also as an instance of NonSustainablePart (see Figure 5-14).


Resource Form

URI: http://example.org/instances#FrontLeftWheel

Annotations

Other Properties

business:hasAssembly

business:isPartOf

FrontAxle

embryo:hasClassification

i4

rdf:type

RDL:NonSustainablePart

RDL:Wheel

business:Part

Figure 5-14 Instance classified after execution of a business rule

5.4 Conclusion

In this chapter, we discussed usability and integration of the different components described in Chapter 4, and provided a use case to demonstrate how to use our framework.

We first described the importance of the modeling language when developing information models and representing data. We highlighted the difficulty in finding a good compromise between existing technologies and our PLM framework requirements. We showed that with the proper translator, they can be used together. To achieve this, we developed a new ontology, OntoSTEP, that enables the design community to use a familiar technology for developing product models, EXPRESS. With OntoSTEP, they benefit from the validation and reasoning support offered by some of the new semantic technologies such as OWL, SPARQL and SPIN.

We also developed a mapping to help leverage both DEXML and OWL representations of the same information. The simplicity of UML enables modelers to define complex business objects easily; the validation mechanisms offered by OWL ensure that instances of these business objects are valid. This unique bridge is important for our framework to be widely accepted and used.

The second part of this chapter aimed at demonstrating usability and efficiency of our framework through a basic use case. The first part of the use case showed how generic concepts can be specialized and information validated using emerging semantic technologies. The second part of the use case was a demonstration of how the framework can easily handle new requirements. Because PLM involves many stages, actors and processes, it was necessary to show that the proposed framework can support evolving requirements.

6 Conclusion and future work

The focus of this dissertation is dynamic customization of information models in the context of PLM, Product Life-cycle management. We noted that an important requirement for PLM is the ability to dynamically extend information models. Our objective was to provide an alternative solution to the currently available static methods of extending information models. We wanted this solution to reduce complexity and cost of both development and implementation. To develop such an alternative, we had to overcome two important challenges. First, to find a technical solution, involving methods and tools, to dynamically customize information models. Second, to enable a smooth transition from the tools and methods currently used by PLM information modelers to the new ones that we needed.

In chapter 2 we identified and described four major extension approaches that could potentially be applied to PLM: the NIST Core Product Model, the ISO 10303 Product data representation and exchange, OASIS implementation framework for ISO 10303-239 Product life cycle support, and the OAGIS OAGi industrial standard. In Chapter 3 we defined five requirements derived from specific PLM needs and we concluded that (1) individually, no single framework fully meet our requirements, and (2) even collectively, all of our requirements could not be fully met.

To achieve our objective, tackle the challenges, and overcome the drawbacks and weaknesses of the aforementioned approaches, we developed a new framework composed of the following components:

1. The embryo: Using the work done in ISO 10303, we developed a new information model, which we called the embryo. This model, when properly extended, enables users to add new concepts and properties dynamically. This work improves what already exists in ISO 10303 because (1) it enables semantic specialization of properties with their units and (2) it is designed specifically to be used with OWL ontologies. This model has been developed in a way that it can be used with any existing modeling language. We also provided a UML class diagram representation as well as an EXPRESS schema for the embryo.
2. We proposed a novel method for defining, and enabling consistency checking of, new concepts and properties dynamically and formally using OWL. The OASIS implementation framework for ISO 10303-239 recommends using flat ontologies in OWL for defining dynamic concepts. We have shown in Chapter 3 that when enriching a flat ontology with new OWL axioms we can achieve consistency checking at the schema level. We have also shown in Chapter 4 that because of the Open World Assumption, it is not possible to enable data validation and consistency checking at the data level. To overcome this issue, we enriched the ontology using a recent technology, SPIN. We chose SPIN because it simulates a Closed World Assumption (CWA) for OWL ontologies. SPIN can be used to perform validation and consistency checking at the data level.

3. DEXML: We created a new method and a tool for defining instantiation patterns of the previously defined concepts graphically. The OASIS implementation framework for ISO 10303-239 presents a mechanism, called a template, that enables users to define patterns of concept instantiations to represent business objects. In (Krima et al., 2011) we have identified benefits, weakness and limitations of this approach. We overcame its limitations by developing a new method, DEXML, using the UML activity diagram.
4. OntoSTEP: The first three solutions tackled the first challenge and provided a mechanism for dynamically customizing an information model using OWL and SPIN. To tackle the second challenge we developed OntoSTEP, a new method and a tool to map EXPRESS information models and data into OWL. This enables PLM modelers to keep using familiar technologies while taking benefits from our new solution such as consistency checking and data validation using SPIN.
5. The PFM: Based on the DEXML work, we created a new UML profile, the PFM (Product Family Management), to support graphical and formal representation of product families. This profile enables formal definition of product customizations whether users deal with product specialization or product extension.

The emphasis on using recent and emerging information modeling techniques and tools such as ontology, OWL, SPARQL and SPIN has been driven by the formal semantics and validation mechanisms they offer. This emphasis on web semantic technologies makes this framework unique and a demonstration of the power of the semantic web.

Finally, in Chapter 5 we went step-by-step through a use case and did demonstrate how to use our framework and the benefits of doing so. The use case included creation of a generic information model with a finite number of concepts to represent an unlimited number of future, product-related, information requirements for a manufacturing company. We also demonstrated how semantic technologies can be used to classify data and apply business rules. The last part showed that this framework is able to support new requirements when properly using its specialization mechanism. In this particular case, no software modifications were needed.

This framework has required different tools that we developed in order to generate information automatically from formal definitions. The diversity and number of tools combined with the lack of an Integrated Development Environment is an inconvenient when working with big models. The principal reason is that many tools need to share and exchange information. Also some information are still to be manually written which is not convenient when working with big models.

Another issue related to big models is the scalability. Scalability is important mainly because of the use of reasoners for classification and validation of data. The number and length of operations required for classification and validation could potentially become an issue that would make our framework difficult to use. To do this, we would need to test the framework on a less trivial example than the one we introduced in Chapter 5.

Also, although we have been able to demonstrate how to build dynamic information models using our framework, we did not provide a solution for the integration of legacy data. Because of the strong influence of STEP on this framework, being able to handle STEP legacy data would bring important added value to this framework. To achieve such a support for legacy data, we aim at developing a STEP to *embryo* mapping.

Our framework has been driven by PLM requirements, which implies this framework is a domain-specific solution. As we said earlier in the introduction, the problem we solved for PLM is not specific to this domain but appears in many others. A next step could be to understand requirements in other domains such as finance and health care in order to develop a generic solution that would not be limited to the PLM area.

References

Arenas, M., & Libkin, L. (2005). XML data exchange: consistency and query answering. *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '05* (p. 13). New York, New York, USA: ACM Press. doi:10.1145/1065167.1065171

Balaban, M., & Maraee, A. (2006). Consistency of UML Class Diagrams with Hierarchy Constraints. *Lecture Notes in Computer Science Next Generation Information Technologies and Systems*, 71–82.

Barnard-Feeney, A., & Hunten, K. A. (2011). Business Objects for Industrial Data Standards. *Proceedings of the ASME 2011 International Design Engineering Techical Conferences & Computers and Information in Engineering Conference*.

Barnard-Feeney, A., & Price, D. (2011). Reeper. Retrieved from http://www.nist.gov/el/msid/reeper.cfm

Benson, T. (2010). Why Interoperability is Hard. *Principles of Health Interoperability HL7 and SNOMED* (pp. 25–34). Springer London. doi:10.1007/978-1-84882-803-2

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, *168*(1-2), 70–118.

Biswas, A., Fenves, S. J., Shapiro, V., & Sriram, R. (2007). Representation of heterogeneous material properties in the Core Product Model. *Engineering with Computers*, *24*(1), 43–58. doi:10.1007/s00366-007-0065-y

Brunnermeier, S., & Martin, S. A. (1999). *Interoperability Cost Analysis of the U . S . Automotive Supply Chain*. *Environmental Research*. National Institute of Standards & Technology (NIST),. Retrieved from http://www.rti.org/publications/abstract.cfm?pub=1390

Feng, T., Dan, B., Lan, L.-C., Zhang, X.-M., Tao, F., Bin, D., Lin-chun, L., et al. (2003). Product family structure and configuration management for mass customization. *Computer Integrated Manufacturing Systems*, *9*(3), 210–213.

Fenves, S. J. (2002). *NISTIR 6736: A core product model for representing design information*. *Technology*. National Institute of Standards and Technology. Retrieved from http://scholar.google.com/scholar?hl=fr&q=Core+Product+Model&btnG=Rechercher&lr=&as_ylo=&as_vis=0#0

Fenves, S. J., Choi, Y., Gurumoorthy, B., Mocko, G., & Sriram, R. D. (2003). *NISTIR 7004: Master product model for the support of tighter design-analysis integration*.

Ferdinand, M., Zirpins, C., & Trastour, D. (2004). Lifting XML Schema to OWL. *Proceedings of the 4th International Conference ICWE 2004* (pp. 354–358). Munich, Germany: Springer.

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

Foufou, S., Fenves, S. J., Bock, C., Rachuri, S., & Sriram, R. D. (2005). A CORE PRODUCT MODEL FOR PLM WITH AN ILLUSTRATIVE XML IMPLEMENTATION. *Proceedings of the PLM'05 International Conference on Product Lifecycle Management* (pp. 21–32). Lyon, France: Inderscience Publishers. Retrieved from http://130.203.133.150/viewdoc/summary;jsessionid=FFFABEE7C21121A616C7543FDFCBF3F2?doi=10.1.1.161.1107

Gallaher, M. P., O'Connor, A. C., Dettbarn, J. L., & Gilday, L. T. (2004). *Cost Analysis of Inadequate Interoperability in the U . S . Capital Facilities Industry*. *Technology*. NIST. Retrieved from http://fire.nist.gov/bfrlpubs/build04/art022.html

Gasevic, D., Djuric, D., Devedzic, V., & Damjanovi, V. (2004). Converting UML to OWL ontologies. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (pp. 488–489). ACM Press.

Hodgson, R., & Keller, P. J. (2011). QUDT - Quantities, Units, Dimensions and Data Types in OWL and XML. Retrieved from http://qudt.org/

ISO. (n.d.-a). 10303-239:2005 Industrial automation systems and integration -- Product data representation and exchange -- Part 239: Application protocol: Product life cycle support. International Organization for Standardization.

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

ISO. (1994a). Industrial automation systems and integration -- Product data representation and exchange -- Part 1: Overview and fundamental principles.

ISO. (1994b). Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual. International Organization for Standardization.

ISO. (1994c). Industrial automation systems and integration -- Product data representation and exchange -- Part 203: Application protocol: Configuration controlled 3D design of mechanical parts and assemblies.

ISO. (2001). Industrial automation systems and integration -- Product data representation and exchange -- Part 209: Application protocol: Composite and metallic structural analysis and related design.

ISO. (2003). Industrial automation systems and integration -- Integration of life-cycle data for process plants including oil and gas production facilities -- Part 2: Data model.

ISO. (2004). Industrial automation systems and integration -- Product data representation and exchange -- Part 1275: Application module: External class.

ISO. (2005). Industrial automation systems and integration -- Product data representation and exchange -- Part 41: Integrated generic resource: Fundamentals of product description and support. International Organization for Standardization.

ISO. (2006). Information technology -- Document Schema Definition Languages (DSDL) -- Part 3: Rule-based validation -- Schematron. International Organization for Standardization.

ISO. (2010). Industrial automation systems and integration -- Product data representation and exchange -- Part 214: Application protocol: Core data for automotive mechanical design processes.

ISO. (2011). Industrial automation systems and integration -- Product data representation and exchange -- Part 210: Application protocol: Electronic assembly, interconnect, and packaging design.

Irimia, V. (2011). ECONOMIC INFORMATION SYSTEMS INTEROPERABILITY. *Annales Universitatis Apulensis Series Oeconomica*, *13*(1), 42–47. Retrieved from http://search.proquest.com/docview/895870682?accountid=11262

Johnson Lim, S. C., Liu, Y., & Lee, W. B. (2010). Multi-facet product information search and retrieval using semantically annotated product family ontology. *Information Processing & Management*, *46*(4), 479–493. doi:10.1016/j.ipm.2009.09.001

Klein, L., Liutkus, G., Nargelas, V., Sileikis, P., Baltramaitis, T., Schowe-von der Brelie, B., Alfter, A., et al. (2008). *Ontologies derived from STEP data models*. Retrieved from http://www.s-ten.eu/index.php?sel=4-2

Knublauch, H., Hendler, J. A., & Idehen, K. (2011). SPIN - Overview and Motivation. Retrieved from http://www.w3.org/Submission/spin-overview/

Koide, S., Aasman, J., & Haflich, S. (2005). OWL vs . Object Oriented Programming. *System*, 1–15. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.8108&amp;rep=rep1&amp;type=pdf

Korthaus, A. (1998). Using UML for Business Object Based Systems Modeling. *The Unified Modeling Language --- Technical Aspects and Applications* (pp. 220–237). Physica-Verlag. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.2944

Krima, S., Barbau, R., Fiorentini, X., Rachuri, S., & Sriram, R. D. (2009). *OntoSTEP: OWL-DL Ontology for STEP*. Retrieved from http://www.nist.gov/manuscript-publication-search.cfm?pub_id=901544

Krima, S., Bertucat, R., Lubell, J., Rachuri, S., & Foufou, S. (2011). DEXML: A first step toward a UML based implementation framework for PLCS. *Proceedings of the ASME 2011 International Design Engineering Techical Conferences & Computers and Information in Engineering Conference*. ASME.

Manchester, U. of. (2004). FaCT++. Retrieved from http://code.google.com/p/factplusplus/

Moon, T., Fewell, S., & Reynolds, H. (2008). The What, Why, When and How of Interoperability. *Defense Security Analysis*, *24*(1), 5–17. doi:10.1080/14751790801903178

Motik, B., Horrocks, I., & Sattler, U. (2007). Adding Integrity Constraints to OWL. *Proceedings of the 3rd International Workshop on OWL: Experiences and Directions*. CEUR.

NIST. (2006). NIST SE Interoperability Worksite. Retrieved from http://syseng.nist.gov/se-interop/sysml/validator

OAGi. (2003). *User's guide for extending OAGIS 8.0*.

OASIS. (2009). Assigning_reference_data PLCS template. Retrieved from http://www.plcs-resources.org/plcs/dexlib/data/templates/assigning_reference_data/sys/cover.htm

OASIS. (2010a). Data Exchange Specifications (DEXs). Retrieved from http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_dex.htm

OASIS. (2010b). Introduction to PLCS template. Retrieved from http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_template.htm

OASIS. (2010c). Reference Data. Retrieved from http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_refdata.htm

OMG. (2009). Model Interchange Working Group. Retrieved from http://www.omgwiki.org/model-interchange/doku.php

OMG. (2010). Object Constraint Language 2.2. Object Management Group.

OMG. (2011). *MOF 2 XMI mapping 2.4.1*.

Parsia, C. &. (2011). Stardog: The RDF database. Retrieved from http://stardog.com/

Pratt, M. J. (2001). Introduction to ISO 10303—the STEP Standard for Product Data Exchange. *Journal of Computing and Information Science in Engineering*, *1*(1), 102. doi:10.1115/1.1354995

Price, D., & Bodington, R. (2004). Applying Semantic Web Technology to the Life Cycle Support of Complex Engineering Assets. *Proceedings of the 3rd International Semantic Web Conference* (pp. 812–822). Springer-Verlag.

Rachuri, S., Han, Y.-H., Foufou, S., Feng, S. C., Roy, U., Wang, F., Sriram, R. D., et al. (2006). A Model for Capturing Product Assembly Information. *Journal of Computing and Information Science in Engineering*, *6*(1), 11–21. Retrieved from http://link.aip.org/link/?CIS/6/11/1

Reiter, R. (1998). On Integrity Constraints. In Vardi & MEditor (Eds.), *Proceedings of Conference on Theoretical Aspects and Reasoning about Knowledge* (pp. 97–111). Kaufmann, Morgan.

Sirin, E., & Tao, J. (2009). Towards Integrity Constraints in OWL. *Proceedings of the 6th International Workshop on OWL: Experiences and Directions*. CEUR.

Szykman, S., Racz, J. W., Bochenek, C., & Sriram, R. D. (2000). A web-based system for design artifact modeling. *Design Studies*, *21*(2), 145–165. doi:10.1016/S0142-694X(99)00044-7

Szykman, S., & Sriram, R. D. (2006). Design and implementation of the Web-enabled NIST design repository. *ACM Transactions on Internet Technology*, *6*(1), 85–116. doi:10.1145/1125274.1125278

Troussier, N., Bricogne, M., Belkadi, F., Durupt, A., & Ducellier, G. (2010). An extension of the Core Product Model for Reverse Engineering. *International Conference on Product Lifecycle Management PLM10*.

Usman, M., Nadeem, A., Kim, T.-H., & Cho, E.-S. (2008). A Survey of Consistency Checking Techniques for UML Models. *2008 Advanced Software Engineering and its Applications*, 57–62.

W3C. (2004). OWL Web Ontology Language. Retrieved from http://www.w3.org/TR/owl-ref/

Wang, F., Fenves, S. J., Rachuri, S., & Sriram, R. D. (2003). Towards modeling the evolution of product families. *Proceedings of CIE'03 ASME Computers and Information in Engineering Conference*.

West, M. (1994). *Developing High Quality Data Models Volume 2: The Generic Entity Framework*.

West, M. (1995). *Developing High Quality Data Models*.

Zhang, L. Z. L., Jiao, J. J. J., & Helo, P. Integrated Product and Process Family Data Modeling for Product Lifecycle Management. , 2006 4th IEEE International Conference on Industrial Informatics 531–536 (2006). doi:10.1109/INDIN.2006.275612

Zhao, W., & Liu, J. K. (2008). Computers in Industry OWL / SWRL representation methodology for EXPRESS-driven product information model Part I . Implementation methodology. *Computers in Industry*, *59*(6), 580–589.

van Engers, T. M., Gerrits, R., Boekenoogen, M., Glassée, E., & Kordelaar, P. (2001). POWER: using UML/OCL for modeling legislation - an application report. *Proceedings of the 8th international conference on Artificial intelligence and law - ICAIL '01* (pp. 157–167). New York, New York, USA: ACM Press. doi:10.1145/383535.383554

List of publications

• OntoSTEP: OWL-DL Ontology for STEP - Krima S. , Barbau R. , Fiorentini X. , Sudarsan R. , Sriram R. D - NIST Interagency/Internal Report (NISTIR) – 7561 – May 2009

• OntoSTEP: OWL-DL Ontology for STEP - Krima S. , Barbau R. , Fiorentini X. , Sudarsan R., Foufou S., Sriram R. D - 6th International Conference on Product Lifecycle Management, 2009, University of Bath, Bath, UK – 2009

• DEXML: A first step toward a UML based implementation framework for PLCS - Krima S, Bertucat R, Lubell J, Sudarsan R, Foufou S - Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011, August 28-31, 2011, Washington, DC

• OntoSTEP: Enriching product model data using ontologies - Barbau R., Krima S., Sudarsan R., Narayanan A., Fiorentini X., Foufou S., Sriram R. D – 2012 Computer Aided Design, Volume44, Issue 6, June 2012, pp. 575-590. DOI=10.1016/j.cad.2012.01.008 http://dx.doi.org/10.1016/j.cad.2012.01.008

• Dynamic customization and validation of product data models using semantic web tools – Krima S., Barnard-Feeney A., Foufou S. – 9th International Conference on Product Lifecycle Management, July 9th - 11th 2012, Montreal, Canada

Résumé :

Ceci est le résumé en français

Mots-clés : Mot-clé 1, Mot-clé 2

Abstract:

This is the abstract in English

Keywords: Keyword 1, Keyword 2