General purpose modeling languages for configuration

Queva, Matthieu Stéphane Benoit

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
General purpose modeling languages for configuration

Matthieu Quéva

Kongens Lyngby 2010
IMM-Technical Report-2010-12
3.1.3 Constraints .................................................. 19
3.2 The Storage System model .................................. 21
  3.2.1 Structure and attributes of the model ............... 21
  3.2.2 Constraints .................................................. 23

4 Existing languages used for modeling ..................... 27
  4.1 Types of languages .......................................... 27
  4.2 UML and OCL ................................................. 29
    4.2.1 The Unified Modeling Language ...................... 29
    4.2.2 The Object Constraint Language .................... 31
    4.2.3 Implementation of the cases studies in UML/OCL .... 32
    4.2.4 Discussion ................................................. 41
  4.3 SysML .......................................................... 42
    4.3.1 Description ................................................. 42
    4.3.2 Implementation of the case studies ................. 44
    4.3.3 Discussion ................................................. 48
  4.4 EXPRESS/STEP (ISO 10303) ............................... 50
    4.4.1 Introduction .............................................. 50
    4.4.2 Implementation of the case studies ................. 51
    4.4.3 Discussion ................................................. 55

5 Discussion and conclusion .................................... 57
  5.1 Comparison of the languages based on the requirements .. 57
5.2 Conclusion ............................................................... 59

A Additional SysML diagrams ........................................... 61
   A.1 Home Multimedia Station ......................................... 61
   A.2 Storage System ..................................................... 64

B Source files for EXPRESS models .................................... 67
   B.1 The Home Multimedia Station ................................. 67
   B.2 The Storage System .............................................. 74
In the later years, there has been an important need for companies to reduce their costs while proposing highly customized products. Indeed, today’s customers demand products with lower prices, higher quality and faster delivery, but they also want products customized to match their unique needs.

1.1 Mass Customization

To meet these demands, the manufacturers have to adapt their business model to Mass Customization, allowing the customers to order customized products, often choosing among hundreds of product features and options, for a competitive price.

Mass customization can however result in an expansion of the specification process, causing iterations to be developed in the process. This process represents the tasks that are done on an individual order before production, and that defines the product to be manufactured (Figure 1.1).

The concept of Mass Customization thus differs from previous industrial processes such as:
Introduction

• **Mass Production** consists in producing an important amount of the same standard items, with no option to customize them. This type of process was popularized by Henry Ford, who was describing it by:

  “Any customer can have a car painted in any color he wants, as long as it is black.”

• **One-of-a-kind Production** consists in producing a small amount of items, but with a wide panel of different options to customize these few items. This type of process is usually used when producing large, highly complex and specific products.

• **Small Series Production** consists in manufacturing customized products in small series. The companies working with this type of process do usually not produce as much as in Mass Production, and neither are they customizing their products as well as in the One-of-a-king Production.

These three types of processes can be seen in Figure 1.2.

To successfully implement Mass Customization, the manufacturers have to overcome three major challenges ([22]):

1. **Lead time**: producing a custom configuration for each product becomes a highly complex and time-consuming task. As the number of parts increases, a simple
1.2 Product configuration

The Product Configuration problem is defined by Junker in [18] as characterized by two constituents:

product can easily end up generating thousands of product variations. This can affect the specification process, increasing the lead time in a significant proportion.

2. **Quality assurance**: producing a significant amount of product variations involving hundreds of configurable parts increases the possibility of making errors in the process. This can create major schedule slips and can lead to costly unplanned iterations in the process.

3. **Expertise**: being able to configure complex products requires a comprehensive product knowledge and expertise from the engineers that are in charge of the configuration process. This results in a need for substantial training, which may even be repeated along the years, as technical changes occur frequently.

These issues can make the implementation of Mass Customization very challenging for the companies, or even become barriers. In order to improve this implementation, it is important to develop significant information technology capabilities. One of these technologies is *product configurators*, described in Section 1.2.

![Diagram showing the three main types of processes](image)

Figure 1.2: Three main types of processes ([11])
1. A catalog which describes the generic components in terms of their functional and technical properties and the relationship between both.

2. User requirements and user preferences about the functional characteristics of the desired configuration.

and the corresponding configuration task, which consists in finding the following answer:

1. One or more configurations that satisfy all requirements and that optimize the preferences if those requirements are consistent.

2. An explanation of failure in the other case.

Product configuration is applied during the specification processes, as can be seen in Figure 1.3.

![Figure 1.3: Specification process with configuration ([11])](image)

Implementing product configuration can be made through the use of a product configurator: a software tool that captures the customer’s requirements as input and automatically generates customized product designs matching the customer’s specific needs, based on predefined design constraints.

There are actually two main issues in the configuration problem:

1. The first one concerns the modeling time, i.e. the time where a design engineer will define a model for the product. At this time, there is a need to find
1.3 Goal and outline of the report

As has been pointed out previously, Product Configuration poses both modeling and solving challenges that have to be overcome when implementing a product configurator. The aim of this report is to provide an analysis of several existing modeling languages.

Chapter 1 has briefly introduced important concepts, such as Mass Customization and Product Configuration. The aim of this introduction was to give the reader some background in Configuration, in order to easily follow the rest of the report.

Chapter 2 covers the analysis of the requirements for modeling product configuration knowledge. These requirements are defined from the literature and previous user experience in Product Configuration.

The role of Chapter 3 is to describe two case studies designed to illustrate the different requirements defined in the Chapter 2. These cases are intended to provide the reader an easy way to materialize the requirements into concrete and easily understandable examples.

Chapter 4 describes several existing modeling languages. The case studies defined in Chapter 3 are used there to analyze the languages and see whether they can fit the requirements established for product configuration modeling.

Finally, Chapter 5 covers the overall comparison between all the modeling languages analyzed and draws the conclusion of the report.
Chapter 2

Requirements for product configuration modeling

In order to design a framework for product configuration, it is essential to understand the requirements for all the parts of this framework. This chapter aims at defining all the requirements for both the modeling environment, especially in the case of a modeling language. These requirements are defined, in order to be applicable to the different case studies in Chapter 3, and are derived from various literature.

The requirements form a basis on which to evaluate the different modeling environment existing currently. Of course, some of these requirements have more importance than others, but a complete environment should try to fulfill them all.

2.1 General modeling requirements

In this part are presented the general requirements for a modeling environment:

1. Separation between product modeling and configuration process: The first requirement concerns the structure of the configuration framework itself. As underlined in [3], the configurable product model should be clearly separated
from the configuration process. Indeed, these two tasks are usually performed by different persons, most of the time with different skills: the modeling task would usually be performed by a design engineer, while the configuration task would be performed by a sales person. Thus this abstraction is needed so that there is no confusion between these two parts.

2. **Support of object-oriented concepts**: The modeling environment should allow the user to create a model using the object-oriented approach. This approach have been favored by several researchers ([11, 18, 14, 4, 12]): it is indeed very suitable for product modeling, as products’ components can naturally be seen as objects. It thus makes the system easier to maintain, compared to systems with flat models. Two essential features are the support of *partonomy* (part-of, aggregation) and *taxonomy* (kind-of, generalization) relations.

*Example:* The example of a very simple *Car* model with two partonomy relations (to *Wheel* and *Engine*) and two taxonomy relations (*Standard* and *Cabriolet*) is shown in Figure 2.1.

![Figure 2.1: Car model with partonomy and taxonomy relations in UML notation (see Section 4.2)](image)

3. **Easy-to-use**: The persons that will interact with the modeling environment are usually design engineers. Thus they usually possess only basic programming skills. The modeling environment should be accessible without advanced training in programming [3], and support easy development through tools for a fast implementation. Also, the terms used should be based on a widely accepted terminology, e.g. following well-known conceptualizations of configuration [19, 7].

4. **Graphical representation**: Providing a graphical representation to the user is important for an easy comprehension and a lower maintenance effort [3]. Thus
the modeling environment should be able either to provide directly a graphical view, or to have a structure such that the implementation of a Graphical User Interface on top of it is possible. Such a graphical representation should contains information about the different attributes of the products as well as their structure (e.g. using different views [12]).

**Example:** A example of a graphical representation can be the Product Variant Master (PVM), developed by [11] (Figure 2.2).

![Image of a Clock model](image)

**Figure 2.2: PVM view of a Clock model**

5. **Extensibility:** Companies use many applications around configurators. The system should provide an easy integration of CAD tools, databases, ERP or other systems in the configuration process.

### 2.2 Structure modeling

One part of modeling in configuration deals with representing the structure of the product. Several key features can be highlighted:
1. **(Dynamic) partonomy relations**: Partonomy (or part-of) relations define a subcomponent hierarchy in the product model. Along the object-oriented approach used in modeling, the components appearing in product models should have a multiplicity that can vary from 1 to an “unbound number”. This allows reuse of component models, and the possibility of specifying an “unbound” multiplicity permits to have a dynamic structure.

**Example**: In the case of a model of an alimentation rack that contains one or more power cards having common attributes, the power card model will be involved in a partonomy relation with the rack with a multiplicity different than 1. In the case the need for power increase due to some selections in other parts of the product, the number of power cards may increase dynamically at runtime.

2. **(Multiple) taxonomy relations**: Using multiple taxonomy (or kind-of) relations, a model can inherit the attributes from several different products. This can allow a better reuse of previously defined models, and thus facilitate the model’s maintenance. However, this comes with some inherent problems of potential interference between the different parents’ attributes.

**Example**: An example can be seen in Figure 2.3, where a Combo TV/VCR is a specialization of both a TV and a Video Recorder. The combo would then inherits attributes from both the TV and the Video Recorder, and can add his own. However, if both TV and Video Recorder have the same attribute (e.g. output), this multi-specialization cannot be done, as the Combo will not be able to determine which attribute to use.

![Figure 2.3: Example of a multiple taxonomy relation](image)

3. **Connection ports**: A connection port represents another type of connection between components. It specifies a bidirectional and non-hierarchical connection, on the contrary to the taxonomy and partonomy relations, which both represents a relation with a parent and a child.

**Example**: Connection ports can for example correspond to a special cabling between two parts of a product - a power supply and a computing unit for example. Though the components are connected, this association does not involve any parent-child relationship. Finally, it is also possible to associate a “cable” component to this connection.
4. **Component groups**: Allowing the user to define groups of components is a simple yet important feature in product modeling when it comes to product maintenance. Indeed, this makes it much easier to organize product knowledge data, as it allows to structure the model and its components according to specific criteria.

*Example:* A chair model can have several groups: one dealing with subparts/components relative to the chair color and appearance, one with its shape, etc...

5. **Default values**: Another very useful feature is the possibility to declare default values for the attributes of the models, as pointed out in [12]. This values will then be assigned to the attributes when the configuration process is started, and the user would have the possibility to change them during configuration time. This permits to improve the configuration experience, avoiding long default set-up of the models during configuration.

6. **Hidden/Locked attributes**: Providing the possibility to specify the visibility of the attributes helps the designer to create models easier to maintain, as some of the attributes may be used as intermediary data containers, and thus not accessible to the end-user. Some other attributes can also be used as read-only, in order to provide the customer with some unmodifiable information. Janitza [12] concludes that it lets the customer be “free in his level of detail, but always guided in his decision making process”.

7. **Data types and units**: A product can be complex and contain more than one data type with a specific unit. That is why other data types such as enumerations, sets, dates or integers can be needed. It should also be possible to declare different units, in order to make the model’s creation and maintenance easier. These units may even be used in constraints, where a conversion would be needed.

*Example:* User-defined units can be useful to relate the model with the physical product. It also permits to avoid ambiguities when dealing with different scales or dimensions, even when all values are real or integer numbers. An attribute width for a small piece could be declared in centimeters while for another (bigger) piece it could be in meters. Finally, an attribute price could then be declared in euros.

## 2.3 Constraint modeling

Another important aspect of product modeling is the definition of constraints on the model. Requirements for constraint modeling are:
1. **Built-in functions**: A panel of built-in functions and constraints should be made available to the modeler: aside from simple arithmetic and logical constraints, advanced functions (e.g. sum) or constraints (e.g. allEqual) provide a better support to the product modeler.

2. **Tailored components (use of continuous domains)**: Introducing tailored components or real domains for components’ attributes allows to have much more advanced models [3], e.g. when using advanced arithmetic formulas.

3. **Product catalogs**: Product catalogs (or table constraints) are database tables that contain series of acceptable configurations for specific products and attributes (see Table 2.1). These tables are widely used when defining product models, as they provide a database-like way of storing data and are easy to maintain. That is why an easy-to-use modeling tool for these constraints should be available.

<table>
<thead>
<tr>
<th>Type</th>
<th>Output</th>
<th>Input</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>cam1</td>
<td>analog</td>
<td>none</td>
<td>200</td>
</tr>
<tr>
<td>cam2</td>
<td>AVI</td>
<td>none</td>
<td>300</td>
</tr>
<tr>
<td>cam3</td>
<td>anal.+AVI</td>
<td>AVI</td>
<td>400</td>
</tr>
<tr>
<td>cam4</td>
<td>AVI</td>
<td>AVI</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 2.1: Product catalog for a video camera model [18]

4. **Optimization function**: Another interesting feature for product modeling is the ability to declare optimization (or cost) functions. These functions aim at calculating a specific value in order to assess the performance of the system being configured, or simply to determine a specific value calculated in terms of weighted criteria of the model.

**Example**: A typical optimization function to be minimized can be the price of a product. Its value can often be calculated as a function of the price of different elements of the system, and the user usually tends to look for product configuration ending with the lowest final price.

5. **Balance of resources**: Products are often configured according to the resources they produce/consume, such as the energy produced and consumed in a system for example. These resources can then be used in constraints, specifying for example that the energy consumed should not be more than what is produced by the system.

6. **Hard/Soft constraints**: Hard constraints are constraints that must not be violated, while soft constraints may be violated if they are overridden by a user selection or indirectly as a consequence of a constraint with higher priority. This kind of distinction provides the designer with the ability to guide the
configuration process with specific “recommendations”, as he has a consequent knowledge of the product, though it is still possible for the customer not to follow them.

7. **Layout constraints/Positioning**: As explained in [5], a same set of components can correspond to two different products with two different layouts. Adding support for positioning constraints definition provides the design engineer with a more detailed way to specify the different variations of his products, following Mittal and Frayman notions of “ports and connections” ([16]). This can also be used in collaboration with external tools, for example CAD tools.

8. **Inconsistency detection mechanism**: Defining a product model is usually not an easy task, even for a highly qualified product design engineer. Doing so without errors is even harder. That is why the modeling environment should provide a way to check whether the modeled product is consistent, as well as a debugging environment, that can be used for testing purposes.
Requirements for product configuration modeling
Case studies

In order to have a clear view of the possibilities of the different languages described in this report, two models have been designed. This chapter describes these two models: their structure, attributes and constraints. The aim is to illustrate the modeling requirements defined in the previous chapter, and use the models as a base to evaluate the existing modeling languages. The first model represents a Home Multimedia Station, and is presented in the first section. The second section introduces the second model, that represents a Storage System. It must be pointed out that the models have not been designed to represent fiercely the reality, and are not supposed to be fully featured. Instead, they only include the attributes and constraints that are essential to illustrate the modeling requirements previously defined.

3.1 The Home Multimedia Station model

This section introduces the model of a Home Multimedia Station. This model illustrates most of the modeling requirements presented in Chapter 2, though some are missing, and will be shown using the second model in Section 3.2.
3.1.1 Structure of the model

The structure of the model is presented in Figure 3.1. The notation used to draw the structure of the model is taken from [18].

![Diagram of Home Multimedia Station](image)

**Figure 3.1: Structure of the Home Multimedia Station**

The Home Multimedia station is composed by at least one computer, a keyboard, a mouse and some additional accessories:

- The computer is itself composed by an internal hard drive and a primary graphical card, both necessary to make the system work. It then possible to add other cards, such as additional graphical cards or network cards.

- Both the mouse and the keyboard can be wireless or classical. In the model is defined a generic module *Wireless devices*, and both the wireless mouse and the wireless keyboard are specialization of this module (taxonomy/kind-of relation).

- Finally, the accessories available are either external hard drives, software packages or network routers. The external hard drive are specializations of the hard drives. The Network Router module introduces another type of relation: it uses a *connection port* to specify that each of its three outputs (RJ-45 ports) can be connected to the RJ-45 connector of a network card. This connection is done using a cable, that should be able to be either a coaxial cable or an optic fiber.
3.1 The Home Multimedia Station model

3.1.2 Attributes of the model

As explained before, the list of attributes of the model has been kept short, because it takes a lot of time to design a complete model and the focus is on illustrating the modeling requirements for the different modeling languages. That is why usually only one attribute is defined per module.

Here is the list of attributes for each module:

- **Home Multimedia station**: The attribute `price` represents the total price of the product. It is an integer number, and it can range from 0 to infinity. This attribute should be made visible to the user, but should not be modifiable: thus it should be a read-only attributes. The price is declared in USD (dollars).

- **Computer**: This module also have an attribute `price`, which can is also a positive integer. However, this attribute should be actually hidden, as it is only use as an intermediary value for the computation of the total price.

- **Hard drive**: The hard drive module contains an attribute `capacity`. The domain of this attribute is an enumeration of the different options for the hard drive’s capacity: “60GB”, “80GB” or “160GB”. The unit used is the GigaByte (GB). Each hard drive produces an amount of the resource HD Capacity, equal to its attribute `capacity`.

- **Internal HD**: The internal hard drive has an attribute `size` that represents its size, and can have the values: “1,0 inch”, “2,5 inches” or “3,5 inches”.

- **External HD**: The external hard drive contains three boolean attributes `eSATA_connection`, `firewire_connection` and `USB_connection`, representing its compatibility ports. For example, the attribute `eSATA_connection` should be `true` if the external hard drive can be connected to the computer via eSATA, and `false` otherwise.

- **Card**: The module class has an attribute `interface_type` which can have two values: “Integrated” or “PCI”. This attribute specifies whether the card is to be integrated to the motherboard or is a PCI card.

- **Network card**: The attribute `type` for the network card can have two values: “Wifi adapter” or “Ethernet”, depending on whether the network card is for a wireless network and used as an ethernet connection.

- **RJ-45 connector**: This connector is a part of the network card, and can be connected to a RJ-45 port via a “cable” connection port.
• **Graphical card**: The graphical card has an attribute `video_memory` that specifies the amount of memory the card has. Its values can be “128 MB”, “256 MB” or “512 MB”, where MB is the unit (MegaByte).

• **Accessory**: Each accessory has an attribute `price`, which is to be locked as for the price of the whole Home Multimedia Station.

• **Software package**: Each software package has three attributes: the `packageType`, which can be either “Office”, “Photo editing” or “Games”; the `version` (whether it is “Basic” or “Pro”) and its `size`, varying between 1 and 20 GigaBytes. It also inherits the attributes from the Accessory type. Depending on the value of its attributes, a software package consumes more or less of the HD Capacity resource.

• **Network router**: The `boolean` attribute `wireless` defines whether the network router is “wired” (attribute set to `false`) or “wireless” (attribute set to `true`).

• **RJ-45 port**: Each network router has three ports, that all have a boolean attribute `isFiberConnection`. Each port can be associated to a RJ-45 connector via a “cable” connection port.

• **Cable**: This is the connection between a RJ-45 port and a RJ-45 connector. It has an attribute `type`, which can be either “coaxial” or “optic”.

• **Wireless device**: Each wireless device has a `connectivity_type` attribute that represents how it connects with the computer. It can have either the value “Bluetooth” or “USB dongle”. Such a device needs specific drivers, and thus consumes a small part of the HD Capacity resource.

• **Mouse**: The mouse contains an attribute `pointing_technology` that represents the pointing technology of the device. This attribute can have two values: “optic” or “laser”. The mouse also have a locked `price` attribute.

• **Classic mouse**: The classic mouse does not have specific attributes except the one inherited from the Mouse module.

• **Wireless mouse**: On top of the attribute inherited from the Wireless device module, the wireless mouse can have an On/Off button, whose existence is represented by the `boolean buttonOnOff`.

• **Keyboard**: The keyboard can have several possible layouts, represented by the attribute `layout`, that can take three different values: “French (AZERTY)”, “US (QWERTY)” and “Danish (QWERTY)”. As the mouse, the keyboard also have a read-only `price` attribute.

• **Classic keyboard**: The classic keyboard does not have specific attributes except the one inherited from the Keyboard module.
• **Wireless keyboard:** The wireless keyboard does not have specific attributes except the ones inherited from the Keyboard and the Wireless device modules.

It has to be pointed out that each specialized module also contains the attributes of the module(s) it is a specialization of.

The Home Multimedia Station modeling would also benefit from declaration of components groups. Indeed, three groups could be declared:

- The group “Computer Parts” that contains the Computer, the Internal Hard Drive and the Card/Graphical Card/Network Card (+ RJ45-connector).
- The group “Accessories” that contains the components Accessory, CD Player, MP3 Player, Portable Music Player and Network Router (+ the RJ45-port).
- The group “Input devices” that contains the Mouse/Wireless Mouse/Classic Mouse, the Keyboard/Wireless Keyboard/Classic Keyboard and the Wireless Device components.

These groups would bring a better organization between the components, providing easier maintenance of the model, and permit to group the components at configuration time for a better presentation to the user.

### 3.1.3 Constraints

This last part introduces the constraints defined in the Home Multimedia Station model:

1. If a *network router* is selected, the *network card* associated with it must have the same type: if the *network router* **wireless** attribute is set to true, the associated *network card* must have a **type** set to “Wifi adapter”; otherwise, the *network card* must have a **type** set to “Ethernet”.

2. A constraint should be set up for *price calculation*, specifying that the total price of the Home Multimedia Station must be equal to the sum of the price of the mouse, the keyboard, the computer and the potential additional accessories.

3. The *wireless mouse* should have a **button On/Off** if and only if it is connecting via Bluetooth. Also, wireless keyboards are not sold separately from wireless mouses, so choosing such a keyboard implies choosing a wireless mouse as well.
4. Each external *Hard drives* (declared as accessories) must all have an equal or a bigger capacity than all the *primary Hard drive* attributes of the computers. This constraint is used in order to test the languages’ ability to declare “complex” constraints.

5. Several product catalogs table are used to narrow some of the possible combinations, and set the price of different parts of the product. They can be seen in the Tables 3.1, 3.2, 3.3, 3.4 and 3.5. Moreover, the prices of the *external hard drives* and the *network routers* are fixed respectively at 40 and 50 dollars.

<table>
<thead>
<tr>
<th>Connectivity type</th>
<th>Pointing technology</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth</td>
<td>laser</td>
<td>35</td>
</tr>
<tr>
<td>USB dongle</td>
<td>laser</td>
<td>30</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>optic</td>
<td>25</td>
</tr>
<tr>
<td>USB dongle</td>
<td>optic</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3.1: Product catalog for the wireless mouse

<table>
<thead>
<tr>
<th>Pointing technology</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>laser</td>
<td>15</td>
</tr>
<tr>
<td>optic</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.2: Product catalog for the classic mouse

<table>
<thead>
<tr>
<th>Type</th>
<th>Connectivity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>wireless</td>
<td>Bluetooth</td>
<td>30</td>
</tr>
<tr>
<td>wireless</td>
<td>USB dongle</td>
<td>25</td>
</tr>
<tr>
<td>classic</td>
<td>–</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3.3: Product catalog for the keyboard

<table>
<thead>
<tr>
<th>Package type</th>
<th>Version</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office</td>
<td>Basic</td>
<td>12</td>
<td>50</td>
</tr>
<tr>
<td>Photo editing</td>
<td>Pro</td>
<td>6</td>
<td>700</td>
</tr>
<tr>
<td>Games</td>
<td>Basic</td>
<td>19</td>
<td>650</td>
</tr>
</tbody>
</table>

Table 3.4: Product catalog for the software packages

6. A *soft* constraint is also defined: “If the graphical card is integrated, then its video memory should preferably be of 128MB”. This constraints is used to set
3.2 The Storage System model

The second model, representing a Storage System and presented in this section, is smaller than the Home Multimedia Station model. The aim of this model is to illustrate requirements such as use of continuous domains, hard and soft constraints, existence or layout constraints. This model is derived from the one presented by Aldanondo et al. in [3].

3.2.1 Structure and attributes of the model

The structure of the model is presented in Figure 3.2. The notation used to draw the structure of the model is still the one from [18], where attributes have been added to each component.

The Storage System is composed by a mandatory Book Case (BC), and two additional elements:

- A Low Cabinet (LC)

<table>
<thead>
<tr>
<th>Primary HD capacity</th>
<th>Number of additional cards</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 GB</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>20 GB</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td>80 GB</td>
<td>0</td>
<td>800</td>
</tr>
<tr>
<td>80 GB</td>
<td>1</td>
<td>850</td>
</tr>
<tr>
<td>160 GB</td>
<td>1</td>
<td>950</td>
</tr>
<tr>
<td>160 GB</td>
<td>2</td>
<td>1100</td>
</tr>
</tbody>
</table>

Table 3.5: Product catalog for the computer

the value of the video memory in case the interface type is Integrated, but this value can be changed afterward by the user, or another - hard - constraint.

7. Finally, a simple optimization function (or cost function) is defined: this function represents the ratio between the total price of the station and the sum of capacity of all the hard drives.

Table 3.5: Product catalog for the computer
Figure 3.2: Structure of the Storage System

- A High Cabinet (HC), that can have up to three roll-out shelves and three drawers.

Several attributes are also part of this model:

- **Storage System**: The overall Storage System has a **color** attribute that represents the unique color of the system. This color can then take two values: “Wood” or “Painted”.

- **Book Case**: The book case only has one attribute, **height**. This attribute is a real number, and represents the height of the book case.

- **High Cabinet**: The high cabinet also has an attribute **height** of type real. It also have partonomy relationships with the roll-out shelf and the drawer components.

- **Low Cabinet**: As the book case and the high cabinet, the low cabinet has an attribute **height**. This attribute has a **default value** of 72, though it can be modified during configuration.

- **Roll-out shelf and Drawer**: These two components do not have any configurable attributes.

Simple production attributes can also be added to this case study:
1. When producing the items, the Storage System would need a high cabinet of a small size if the high cabinet exists and its height is less or equal to 72, while a high cabinet with a big size would need to be used if the height is greater than 72.

2. In order to produce the high cabinet of the Storage System, a standard assembly operation will be performed (the color of the system would specify what type of operation it is). Then, in the case of a high cabinet with a greater height, an extra assembly operation would be needed to keep up with the bigger cabinet.

### 3.2.2 Constraints

This model is a basis for two problems: one with tailored attributes and advanced logical constraints, and one with layout constraints.

#### 3.2.2.1 Basic problem

In the basic problem, the layout of the system is fixed, as can be see in Figure 3.3.

![Figure 3.3: Storage system with fixed layout](image)

Here, the Storage system is constrained is several ways:

1. The **height** of the *Book Case* must be between 72 and 216.
2. If the *High Cabinet* exists, then its **height** must be between 50 and 144.

3. If the *High Cabinet* exists, then it implies that the Low Cabinet exists and the height of the Book Case is greater than the height of the Low Cabinet plus the minimum height of the High Cabinet, which is 50. That is:

\[ BC_{\text{height}} \geq LC_{\text{height}} + 50 \]

4. Finally, the quantity of drawers and roll-out shelves cannot exceed 4 in total:

\[ \text{numOf}(\text{shelves}) + \text{numOf}(\text{drawers}) \leq 4 \]

where \( \text{numOf} \) is a function returning the multiplicity of the partonomy relation.

### 3.2.2.2 Problem with layout constraints

The second problem deals with layout constraints, i.e. constraints that specifies the different layouts the system can be in. Indeed, as explained in Chapter 2, the same set of components can correspond with several different products according to their layouts. In order not to complicate the model too much, discrete attributes or attributes with fixed values will be used instead of continuous ones, as is done by Aldanondo et al. in [3].

Thus, the model’s attributes have now discrete domains or fixed values:

- The **height** of the *Book Case* has now only two possible values: 72 or 216.
- The *High Cabinet* has a fixed **height** of 144.
- The *Low Cabinet* has a fixed **height** of 72.

The two cabinets (Low and High) are still not mandatory, and now the *High Cabinet* exists if and only if the *Low Cabinet* exists and the *Book Case* has a **height** of 216.

Depending on the possibilities of the modeling language used, several approaches can be used to solve the layout problem: using component absolute locations or relative locations. The solution with absolute locations consists in associating location attributes to the element of the system, while the solution with relative locations consists in locating each component relatively to each other. The problem with location attributes (solution with absolute locations) can be seen in Figure 3.4.
3.2 The Storage System model

This solution consists in adding location parameters to the model. This can be very handy when dealing with small models, but it becomes really complicated when the number of possible layouts increases.

Aldanondo et al. refers in [3] to the notion of component ports. This concept points that two components can be connected together through components ports. This port mapping allows to design a product with layout constraints using the components relative locations. This is illustrated by Figure 3.5.

In this model, the BookCase has four ports (P_{BC-1}, P_{BC-2}, P_{BC-3} and P_{BC-4}), the High Cabinet has three ports (P_{HC-1}, P_{HC-2} and P_{HC-3}) and the Low Cabinet also have three (P_{LC-1}, P_{LC-2} and P_{LC-3}). The figure shows the different connection possible, $\emptyset$ representing that the port is not used.

Though this kind of layout problems refers to space location and will obviously need assistance from a graphical view of the product, it can be interesting to see whether such components ports can be modeled in the different languages studied in this report.

Figure 3.4: Storage system with layout attributes using location attributes [3]
Figure 3.5: Storage system with layout attributes using ports and connections [3]
This chapter gives a presentation of several major languages used for modeling. The aim is to provide the reader with an overview of what is currently existing as modeling languages, and to perform a comparison between these languages, according to the modeling requirements defined in Chapter 2. The two test cases developed in Chapter 3 are thus implemented in each of the modeling languages, in order to confront the languages with the modeling needs raised by these two specific cases.

4.1 Types of languages

A modeling language is a language that is used to represent knowledge or information in a structured way. It can be used to express a lot of different systems, from the Enterprise Architecture ([13]), to Software Engineering ([21]), through products architecture, among others. Representing these concepts in a formal way can have several advantages, including:

- Providing formal specifications that can be reused and exchanged between coworkers for example, improving team understanding and communication.
• Defining the conceptual design of a system, which allows to avoid mistakes due to the lack of a clear vision of the system’s architecture.

• Formalizing a system’s structure and requirements, and providing a basis for a clear and well-defined structure for implementation (in case of software development), manufacturing (e.g. for a product), etc...

A modeling language is defined by two fundamental parts: its syntax (the rules defining how the language should be arranged) and its semantic (how the language should be interpreted). The syntax of a modeling language consists usually of abstract syntax and concrete syntax. The concrete syntax represents the visual part of the language that will be interfaced to the user to be manipulated, while the abstract syntax express the inner representation of the data from the models.

Before presenting the languages, it is worth splitting them into two main categories: the graphical and the textual modeling languages [10]:

• The graphical languages are using diagrams with symbols to express the different concept needed to represent a specific information. The symbols are usually linked together by lines that represent the relationships between them. The concrete syntax of these languages is thus their graphical notation. The graphical languages for product modeling that will be described in this report are: the Unified Modeling Language (UML) supplemented by the Object Constraint Language (OCL) and the Systems Modeling Language (SysML), an extension to UML.

• The textual languages are using standardized keywords as concrete syntax in order to structure the knowledge representation. This representation is usually interpreted by a computer in the abstract syntax. The textual language studied in this report is the EXPRESS language (Part 11 of the ISO 10303).

Not all modeling languages are executable, however, and in the case of product configuration, at least a link to an executable language is required in order to interface to a solving engine.

The modeling languages presented here have been chosen because of their potential in product modeling, but this list is of course not made to be exhaustive, as there exists a large number of modeling languages in the literature.
4.2 UML and OCL

The first language to be presented is the Unified Modeling Language (UML), associated with the Object Constraint Language (OCL).

4.2.1 The Unified Modeling Language

The **Unified Modeling Language (UML)** is an international standard defined in 1997 at the Object Management Group (OMG). It started as version 1.1, and a major revision has followed, with the adoption of the UML 2.0 version in 2003 by the OMG. The current version 2.1.2.

UML is a visual specification language for object-oriented modeling. It has been created as a general-purpose modeling language, and includes a graphical notation used to create an abstract model of a system, that is referred to as a UML model.

UML 2.0 contains 13 types of diagrams, that are organized hierarchically (Figure 4.1).

![UML diagrams](image)

Figure 4.1: UML diagrams [21]

There exists a relatively important amount of concepts used in UML 2.0 for object-oriented design, from structure concepts (e.g. classes, components, packages...) to relationships (e.g. aggregations, associations, generalization...) through behavioral concepts (e.g. events, messages, methods...).
In order to model products for product configuration, only a subset of all these concepts and diagrams is used. Indeed, most of the interest in product modeling for configuration lies in the product structure, its attributes, subcomponents and the constraints around all of them. That is why the UML class diagram is of prime interest.

Several modeling concepts have to pointed out here:

- UML contains several different types of relations between components, including:
  
  1. The simple association: an association establishes a semantical relationship between two components. A good way to illustrate that is by comparing it to a marriage: an association is binding a man and a woman. In the case of the marriage, the multiplicity is one-to-one, but in general it can have different multiplicities.
  
  2. The aggregation: the aggregation is a specialized form of association, and can be either shared or composite. It represents typical whole/part relationship, where a notion of ownership exists. In the case of the shared aggregation, all object have their own life cycle. For example, an object representing a Department can have a shared aggregation relationship with a Professor object: the Professor belongs to this Department, but if the Department is deleted, the Professor will still exist. On the contrary, for the composite aggregation (composition), the lifecycle of the child is linked to the life cycle of the parent: if the parent is deleted, then the child will be too.
  
  3. The generalization: this relation is used to model inheritance for data and code reuse: the child element inherits all the properties of its parent, and can define new ones.

- UML 2.0 contains an extension mechanism called stereotypes. A stereotype allows designers to extend UML by creating new model elements from existing ones. The new nodes are then stereotyped, which is reflected graphically by adding a name enclosed by guillemets above the name of another element. A stereotype can contain attributes, called tagged values.

An example of a very simple model of car can be seen in Figure 4.2. The Car class represent the Car component, and has two attributes: color, that can be either Black or White, and a boolean sunRoof. It also have four wheels, that are modeled by a composition relation with the Wheel class, and an engine, thanks to the aggregation with the Engine class. Finally, it has a constraint that specifies that “If there is a sun roof, the color is White”. Moreover, two specialized classes are designed: a Standard car and a Cabriolet, which contains two more constraints, specifying that “it has no sun roof” and “it has no spare tire”.

4.2 UML and OCL

4.2.2 The Object Constraint Language

It is worth being pointed out that the constraints in Figure 4.2 are expressed in English. Another way to express constraints in UML is by using a programming language. What is missing in UML is the ability to describe constraints in a more formal way than natural language or raw code. This is what the Object Constraint Language (OCL) is being used for.

OCL is an extension to UML that allows to write standardized constraints. It is actually a textual language that provides constraint and object query expressions that cannot be expressed using notations like diagrams. The aim of OCL is to provide an unambiguous language for constraints specifications, but that can stay accessible to persons with few or no programming skills. OCL is a pure specification language, which means that an OCL expression will not have any side effects. Indeed, when an OCL expression is evaluated, it simply returns a value, and does not change anything in the model.

An OCL statement is always evaluated in a specific context. The context defines the situation in which the statement is valid. It can be a class for example. Then the body of the constraint is defined. OCL statements can contain navigation expressions such as c.sunRoof which, if c is a Car, results in fetching the value of its attribute sunRoof. Finally, OCL constraints can be invariants for a specific class, or pre/post conditions.
for a specific operation, though only invariants are used in product configuration, as components do not contain operations. Although the OCL functions used in this report will be detailed, it is possible to have a complete view of OCL constraints in [17].

It is now possible to express the constraints of Figure 4.2 using OCL (note that the keyword \texttt{self} refers to an object of the class being constrained, though it can be left out in most cases, when the context is clear):

- For the constraint on the Car “If there is a sun roof, the color is White”:
  
  ```oclass
  context Car
  inv: sunRoof implies color = 'White'
  ```

- Specifying that a Cabriolet has no sun roof can be done in two ways, depending whether the constraint has to be evaluated in the context of the Car or of the Cabriolet itself:
  
  ```oclass
  context Cabriolet
  inv: sunRoof = false
  ```

  ```oclass
  context Car
  inv: selfoclIsTypeOf(Cabriolet) implies sunRoof = false
  ```

  The OCL function \texttt{oclIsTypeOf(Type t)} checks if a given instance is an instance of a certain type.

- Finally, specifying that a Cabriolet has no spare tire:
  
  ```oclass
  context Car
  inv: selfoclIsTypeOf(Cabriolet) implies wheels -> forall(w:Wheel | w.isSpareTire = false)
  ```

  The operator \texttt{->} is used to call an operation on a collection, in the following way:

  ```oclass
  collection->operation(arguments)
  ```

  Also, the \texttt{forall} construct permits to test a boolean expression on all elements of a collection. The declaration of the elements’ type (\texttt{w:Wheel}) can be left out when unambiguous.

4.2.3 Implementation of the cases studies in UML/OCL

This part relates the implementation in UML/OCL of the two case studies presented in Chapter 3: the \textit{Home Multimedia Station} and the \textit{Storage System}.
The UML diagrams have been extended here using stereotypes in order to reflect the product configuration concepts, in the same way as Felfernig et al. in [8, 9]. The stereotypes ComponentType, RootComponent, Port and Resource have been defined from the UML metamodel element Class, while the requires, consumes and produces are from the UML metamodel element Dependency, and the default stereotype from the Generalization element.

The Home Multimedia Station

The first model to be implemented is the Home Multimedia Station model, defined in Section 3.1. Figure 4.3 shows the Home Multimedia Station implemented in UML 2.0.

Several points are worth being highlighted:

- It is not possible to specify units like Dollars for the price attribute of the HomeMultimediaStation root element for example. Indeed, only data types related to software engineering are available. Thus the price attribute is declared as an integer.

- Some of the classes are declared as abstract. It is the case for Card, Accessory, Mouse and Keyboard. This means that they cannot be instantiated on their own, and only a specialized class will be acceptable as a final component.

- Each partonomy relation has been represented as a UML composition, as it is often the case in product configuration that the child has no meaning on his own (assuming that it is not possible to buy spare elements separately, in which case shared aggregation could be used). The multiplicity of each composition represents the multiplicity of the partonomy relation, where expression like “0..*” is equivalent to “0..n” with n undefined.

- Taxonomy relations are represented by UML generalization. It is possible to annotate a generalization arrow with the stereotype default, which specifies for example that when a Mouse is chosen, its default type will be a Classic Mouse, though it can be changed afterward in the configuration process.

- Each attribute is represented by its visibility, its name and its type:
  - A public attribute (represented by a “+”) is an attribute that is visible by all, while a private attribute (represented by a “-”) is hidden (e.g. the attribute price of the Computer class).
  - Different predefined data type are available in UML, including all the most classic ones (integer, boolean ...). The real number are represented in
Figure 4.3: UML model for the Home Multimedia Station
UML by floating point numbers (defined with the keyword `float`). Moreover, it is necessary to define a specific UML enumeration each time the domain of an attribute is composed by predefined values (e.g. the attribute `packageType` of the `SoftwarePackage` class uses the enumeration `PackageType`). It is possible to display those enumeration directly in the diagram as “default values”, although it does not really represent the right concept.

- It is also possible to define `locked` parameters in UML 2.0 using the `{readonly}` keyword after the attribute’s definition.

- The `connection port` between the network router’s ports and the network card is here represented using a UML `association` with a multiplicity of `0..1` on the network card’s side. This denote that it is possible to connect (through a RJ45Connector) a Network Card to one of the three RJ45Port components of the network router, though it is not mandatory. The association is bound to an `association class`, called Cable, that contains information on the cable used to connect the router ports with each graphical card.

- A `requires` dependency is defined between the Wireless Keyboard and the Wireless Mouse, using a stereotyped UML `dependency` relation.

- The use of `packages` permits to group the different components according to their similarities.

- Finally, the need for balanced hard drive capacity has been modeled using a class `HDCapacity` annotated with the `Resource` stereotype. Then, some components produce some of this resource (the hard drives) while other needs some (the software packages or the wireless devices for their management software).

Using OCL, it is possible to specify some rules in order to describe the correct usage of the stereotypes [8]. For example:

```
-- Comment: the component classes does not have operations
case ComponentType
inv: self.allOperations -> size = 0

-- Comment: a 'requires' dependency can only be between two classes of stereotype ''
context requires
case requires
inv: self.client.stereotype = ''ComponentType'' and self.supplier.stereotype = ''ComponentType''
```

In the first example, the OCL `allOperations` attribute represents the collection of all operations defined in a class, while in the second example, `client` and `supplier`
Existing languages used for modeling

represent both ends of a UML Dependency relation.

Following the UML implementation, it is still necessary to define, using OCL, some of the constraints that cannot be represented graphically:

1. A network router and its associated network card must have the same type:

   ```
   context RJ-45port inv:
   associatedConnector.parent.type = 'Wifi adapter' implies
   parent.wireless and
   parent.wireless implies
   associatedConnector.parent.type = 'Wifi adapter'
   ```

   Unfortunately, no equivalence operator is present in OCL. Thus the if-and-only-if constraints gets bigger, due to the need for two mirrored implications.

2. Verification of the price of the station:

   ```
   context HomeMultimediaStation
   inv: self.price = self.accessories->collect(a: Accessory|a.price)->sum() + self.computers->collect(c:Computer|c.price)->sum() + mouse.price + keyboard.price
   ```

   Here the OCL function `collect` is used to get the collection of `price` values of accessories (resp. computers), and then the `sum` function is used to sum all the elements in the collections.

3. A wireless mouse has a button On/Off if and only if it is connecting via Bluetooth:

   ```
   context WirelessMouse
   inv: buttonOnOff implies connectivityType = 'Bluetooth' and
   connectivityType = 'Bluetooth' implies buttonOnOff
   ```

4. Each external hard drive must have an equal or a bigger capacity than all the primary hard drives:

   ```
   context HomeMultimediaStation
   inv: self.accessories->select(a|a.oclIsKindOf(HardDrive))
   ->forall(h:HardDrive|h.capacity_in_GB >=
   self.computers->collect(c|c.primaryHD.capacity_in_GB)->max())
   ```
This constraint is composed sequentially: first the accessories collection is screened to only retrieve the external Hard Drives or subtypes; then all external Hard drives’ capacities are tested against the maximum of all primary hard drives’ capacities (computed through the OCL max function).

5. Product catalogs: no expressions for table constraints have been implemented in UML/OCL. This does not make impossible the implementation of product catalogs (i.e. table) constraints, but it makes them much more tedious to write, and thus to maintain, as they have to be converted into logical expressions:

```plaintext
context WirelessMouse
inv: (connectivityType='Bluetooth' and pointingTechnology='laser' and price=35) or
     (connectivityType='USB dongle' and pointingTechnology='laser' and price=30) or
     (connectivityType='Bluetooth' and pointingTechnology='optic' and price=25) or
     (connectivityType='USB dongle' and pointingTechnology='optic' and price=20)

context ClassicMouse
inv: (pointingTechnology='laser' and price='15') or
     (pointingTechnology='optic' and price='10')

context Keyboard
inv: (self.oclIsTypeOf(WirelessKeyboard) and connectivityType='Bluetooth' and price=30) or
     (self.oclIsTypeOf(WirelessKeyboard) and connectivityType='USB dongle' and price=25) or
     (self.oclIsTypeOf(ClassicKeyboard) and price='15')

context SoftwarePackage
inv: (packageType='Office' and version='Basic' and size='12' and price=400) or
     (packageType='Photo editing' and version='Pro' and size='6' and price=700) or
     (packageType='Games' and version='Basic' and size='19' and price=650)

context Computer
inv: (primaryHD.capacity_in_GB='20' and additionalCards->size=0 and price=400) or
     (primaryHD.capacity_in_GB='20' and
     primaryHD.capacity_in_GB='20' and
     primaryHD.capacity_in_GB='20' and
     primaryHD.capacity_in_GB='20' and
     primaryHD.capacity_in_GB='20' and
     primaryHD.capacity_in_GB='20' and
     ...)
```
The OCL `size` function returns the size of the specified collection (here the `additionalCard` collection).

6. Soft constraint: no soft constraint support is available in UML/OCL. Thus it is preferable not to declare this constraint, as it could prevent the user to select the product configuration he needs.

7. There is no straightforward way to declare the optimization function in UML, or OCL.

The Storage System

The second case study to be implemented is the Storage System (see Section 3.2). Two problems exist for this problem: the basic problem and the problem with layout constraints.

The basic problem is represented in Figure 4.4.

In this model, the attributes `height` of the `BookCase`, the `HighCabinet` and the `LowCabinet` are declared as floating-point numbers, in order to express that they are real numbers.

The OCL constraints are defined by:

1. The height of the Book Case must be between 72 and 216:
   ```
   context BookCase
   inv: height <= 216 and height >= 72
   ```

2. The height of the High Cabinet must be between 50 and 144:
   ```
   context HighCabinet
   inv: height <= 144 and height >= 50
   ```
3. If the High Cabinet exists then the Low Cabinet exists and the Book Case’s height is greater or equal than the Low Cabinet’s height plus the minimum height of the High Cabinet:

context StorageSystem
inv: highcabinet->notEmpty() implies
   (lowcabinet->notEmpty() and bookcase.height >=
   lowcabinet.height + 50)

In OCL, the notEmpty function returns true if the specified collection is not empty. The test is done here to know whether the elements exist or not, as the multiplicity for both the highcabinet and the lowcabinet composition relations are “0..1”

4. The quantity of drawers and roll-out shelves cannot exceed 4 in total:

context HighCabinet
inv: shelves->size + drawers->size <= 4

As explained in 3.2.2.2, the problem with layout constraints can be solved in two ways: using absolute constraints or using relative constraints. As the solution using absolute constraints can become quickly overloaded when numerous components are used, it is more interesting to look into how to solve the solution using relative constraints. The UML implementation of this version of the problem can be seen in Figure 4.5.
In order to model these component ports, special attributes have been created with the stereotype *port*. The name of the ports reflects the definition made in Figure 3.5. Indeed, it is not possible to define graphically to what these ports correspond other than by using CAD system integration.

These ports are defined to be mapped to other ports, and an OCL constraint has to be added to define the possible layouts; this constraint is similar to a table constraint, and define all the possible combination for the connections between the ports. The beginning of the constraint is shown below (this one corresponds to the layout (1) in Figure 3.4):

```
context StorageSystem inv:
let l=self.lowcabinet in
let b=self.bookcase in
let h=self.highcabinet in
(b.P_BC-1='P_LC-1' and b.P_BC-2='none' and b.P_BC-3='P_HC-1'
and b.P_BC-4='none'
and l.P_LC-1='P_BC-1' and l.P_LC-2='none'
and l.P_HC-1='P_BC-3' and l.P_HC-2='none'
and l.P_HC-3='P_LC-3') or ...
```

The *let* operator in OCL allows to declare local variables that contains values or attributes that are often used in a constraint. In case they would be available, it would be possible to express the latter constraint in table constraints. Still, the issue here is the redundancy of the constraints: for example, the connection between *P_BC-1* and *P_LC-1* has to be declared twice (*b.P_BC-1='P_LC-1'* and *l.P_LC-1='P_BC-1'*).

UML also contains a concept of *Port*, which corresponds to interfaces attached to a class, and can be mapped to other classes or Ports. However, it is not possible to change their binding dynamically, or to specify several possible connections using the same Port.
4.2 UML and OCL

4.2.4 Discussion

This part summarizes the insights that have been provided by the experience in modeling the two case studies using UML 2.0 and OCL.

UML provides a direct graphical view of object-oriented structure such as object-oriented software architecture or product architecture in the case of this report. This permits to have a clear view of how the components are related to each others. Moreover, as UML is widely applied in industrial software development as a standard model design, its concepts are relatively well known, which makes it an accessible modeling tool. The UML diagram library contains an important amount of object-oriented diagrams, and the use of stereotype permits to extend the existing concepts to configuration-specific concepts.

Then, the integration of the Object Constraint Language (OCL) allows the designer to define complex constraints in a relatively accessible language, that does not require advanced programming skills either. OCL contains a wide variety of constraints, which would be sufficient for most configuration problems.

However, there are some issue with UML and OCL. First of all, the UML language is not aimed specifically at general product modeling, and even less for product configuration, and it can be thus difficult for knowledge engineer to adapt it to product configuration: indeed, the UML language itself had to be extended through the use of stereotypes to fit the configuration problem. All the concepts are aimed at software engineering, which is illustrated by the fact that it is not possible to directly declare new units other that the one used in programming languages (float, integer...), or optimization functions. The lack of support for product configuration specific constraints is also a problem: indeed, constraints defining product catalogs or soft constraints do not have a specific implementation in UML. Although a series of logical constraints can represent product catalogs, table constraints would make it much easier for a knowledge engineer.

Finally, one of the biggest drawbacks in using UML and OCL is the problem of interpretation of the language. Indeed, it is necessary to be able to interpret the models into a declarative representation of the configuration knowledge, in order to apply Constraint Satisfaction Problem solving. Felfernig et al. [9] have provided UML/OCL interpretation in first order logic in order to be able to use it in a configurator. Again, this type of automatic interpretation requires a strictly defined UML profile for product modeling in configuration, instead of using user-defined stereotypes. Research has also been going on in this area [1, 2].
4.3 SysML

4.3.1 Description

The Systems Modeling Language (SysML) is a recent modeling language specified by the OMG. It is actually a UML profile, and thus inherits the characteristics of UML. The aim of SysML is to represent systems and product architectures, as well as their behavior and functionalities, where UML was used for software engineering. The relationship between UML and SysML can be seen in Figure 4.6.

![Figure 4.6: Overview of SysML/UML interrelationship](image)

The development team of SysML aimed on the first hand at limiting the concepts too close from software engineering, and on the other hand at simplifying UML original notations by limiting the number of diagrams available, in order to make it easier to use. Figure 4.7 shows the SysML diagrams.

Some diagrams, like the UML *Class Diagram*, renamed *Block Diagram*, has just been modified through the concept of *Block*, that allows to express any structural element of a system.

Two new diagrams are also present in SysML:

- The *Requirement Diagram* is used to specify the needs of the system. These requirements can be mapped together and to the different components of the
The Parametric Diagram is the second new diagram. It is used to specify mathematical expressions between the different elements of the model.

These new diagrams aim at making the systems and product modeling more complete. The Parametric Diagram is particularly interesting as it permits to declare complex constraints on systems, using mathematical expressions or any other constraint mechanism already available before. Figure 4.8(a) represents a Block Definition Diagram for defining constraints on a special car model, while Figure 4.8(b) uses the equations in a Parametric Diagram to constrain value properties of the model.

The Parametric Diagram can also be used to evaluate performance, using two special stereotypes: Measures of Effectiveness (moe) and Objective Functions. A Measure
of Effectiveness represents a parameter whose value is critical for calculating the cost effectiveness, while an Objective Function acts as an optimization function (see Section 2.1). An example can be seen in Figure 4.9, where Measures of Effectiveness are defined from several equations, and then used in an Objective Function.

![Parametric Diagram defining Measures of Effectiveness](image)

Figure 4.9: Parametric Diagram defining Measures of Effectiveness [20]

These new concepts directly implemented in SysML bring interesting improvements for product modeling, especially when concerned about **product configuration**. Another interesting feature is the possibility to declare one’s own dimension and units, that can thus be used in the model (see Section 4.3.2).

### 4.3.2 Implementation of the case studies

**The Home Multimedia Station**

The implementation of the Home Multimedia Station in SysML is decomposed in several different diagrams. Indeed, SysML allows the definition of all the constraints using Parametric Diagrams, which can be practical when combined with other diagrams such as Requirement Diagrams, if the whole product specification is to be done in SysML. Figure 4.10 shows the Block Definition Diagram for the HMS.

A first point is that this block diagram looks like the one built in UML (with blocks, aggregations, packages...). Modeling using SysML is very similar than doing so with UML, except that almost no user-defined stereotype is needed. Indeed, only the **default** stereotype for default value in generalizations and the **Resource** stereotype on resources have been added; the resource **HDCapacity** is not shown, as it is the same as in UML (Figure 4.3). Units are shown in Figure 4.11.
Figure 4.10: SysML Block Definition Diagram for the Home Multimedia Station
Existing languages used for modeling

Figure 4.11: Units definition with SysML

Figure 4.12: Block Definition Diagrams defining the constraints used
The advantage of declaring units lies in the closer modeling experience, as well as the fact that it helps maintaining the model, by providing clearer specification of the product.

As described before, it is possible to declare objective (or optimization) functions and parametrized constraints using Parametric Diagrams in SysML. Figure 4.12 shows the definition of the constraints that are used in the model, while Figure 4.13(a) and 4.13(b) represents the Parametric Diagrams for the HomeMultimediaStation root block, with both its constraints and objective functions.

Figure 4.13: Parametric Diagrams for the HomeMultimediaStation block
The goal of these Parametric Diagrams is to map the inner attribute of the block to the constraints. This allows to reuse constraints in a more structured way than with raw OCL declaration. However, it must be pointed out that the constraints can still be written as formulas or OCL statements.

The other diagrams can be seen in Appendix A.

**The Storage System**

The implementation in SysML of the second case study, the Storage System, is also very similar to the one made in UML/OCL. The Block Definition Diagram in Figure 4.14 represents the structure of the Storage System, and the constraints used in the model. The model uses the same units than the Home Multimedia Station (Figure 4.11).

Again, it is important to note that OCL is still very useful to express constraints such as the ones constraining the existence of some parts of the product. The constraint on the root element StorageSystem can be seen in Figure 4.15.

The implementation of the Storage System for the problem with layout constraints is identical to the one in UML. Indeed, SysUML does not bring any other solution to this problem, so the best solution seems to be adding a Port stereotype for attributes.

### 4.3.3 Discussion

SysML represents an interesting and powerful extension of UML when it comes to model systems and products. It is indeed possible for a knowledge engineer to design a product model for configuration in SysML almost without adding any new user-defined concepts. The power of SysML also lies in its simplicity compared to UML (fewer diagrams), and the possibility to design a complete product model, including requirements, that can be added to the structural model that is needed for product configuration, though no specific feature is introduced to define BOM or routes information.

In a product configuration-only point of view, several interesting features are added compared to UML. The possibility to introduce constraints in diagrams, and to map model’s internal attributes to their use in the constraints, brings a much clearer view for knowledge engineers not used to raw programming language file, though OCL is still needed in complex constraints. Thanks to this graphical view, it is also possible to group constraints linked together (see example in Figure 4.8(a)), and to
declare them in a parametric way, so they can be generalized and reused.

SysML also permits the declaration of user-defined units, making the model closer to the physical product, and allowing a much easier maintenance of the system, especially in the case several different engineers are using it. Finally, SysML provides a straightforward way of declaring Measures of Effectiveness and optimization functions, that can also be computed using already declared constraints.

However, there are still things missing in SysML. Table constraints are still not available, though it is still possible to implement them using OCL, as in UML. Some concepts, like resources produced/consumed and default choice in inheritance, had also to be added using stereotypes, though much less than in UML. It is also not
possible to declare which block is the root element without using stereotypes. Finally, the interpretation of the language stays an open issue: as SysML is a graphical language, it needs to be efficiently converted so that a constraint engine can be used, as well as to support tools to assist the knowledge engineer during the model’s creation.

4.4 EXPRESS/STEP (ISO 10303)

4.4.1 Introduction

This section introduces the International Standard ISO 10303, which is referenced as STEP (STandard for the Exchange of Product data).

STEP was first released in 1994, and is published as a series of Parts:

- Part 1 provides an overview;
- Parts 11, etc., specify description methods (the EXPRESS family of information modeling languages);
- Parts 21, etc., specify implementation methods (data exchange mechanisms);
- Parts 31, etc., specify conformance testing methodology and framework;
- Parts 41, etc., specify integrated generic information models;
- Parts 101, etc., specify integrated application resource models;
- Parts 201, etc., specify Application Protocols (specific models targeted for product data exchange);
- Parts 301, etc., specify Abstract Test Suites (corresponding to the Application Protocol series);
- Parts 501, etc., specify Application Interpreted Constructs (interpreted models common to two or more Application Protocols).

The goal of STEP is to allow the exchange of data describing a product between Computer Aided system (CAD, CAM, ...etc). STEP is based on Integrated Generic Resources, that are refined for different industrial areas as Application Protocols (AP). Those AP are first defined independently of STEP, according to the concepts of the specific industrial area they refer to, and are then implemented in STEP using the EXPRESS language.

EXPRESS \[6\] is thus a data modeling language standardized as the Part 11 of STEP. It consists of two different representations: textual, or graphical (called EXPRESS-G). However, EXPRESS-G is not able to represent all details that can be formulated in the textual form, on which we will concentrate in this part.

### 4.4.2 Implementation of the case studies

The concepts behind EXPRESS can be used to define product configuration models. The whole implementation of the two case studies in EXPRESS can be seen in Appendix B. This part describes relevant parts of these models.

**The Home Multimedia Station**

The model is organized using EXPRESS schemas. These schemas permits to group the different elements of the model in relevant scopes. EXPRESS supports the object-oriented concepts: components are defined in EXPRESS as entities, and they are composed by attributes that can be of basic types or entities themselves, representing partonomy relations in the model:

```plaintext
SCHEMA HomeMultimediaStationFactory;
    USE FROM ComputerParts;
    USE FROM Accessories;
```
As can be seen in the definition of the HomeMultimediaStation structure, the entities can be aggregated as sets (among other lists, bags, ...) with dynamic multiplicity, ([1:?] means 1-to-many multiplicity, while omitting it means 0-to-many).

Taxonomy relations are also available through the use of (multiple) inheritance and abstract classes:

ENTITY Card ABSTRACT SUPERTYPE;
    interfaceType: INTERFACE_TYPE;
END_ENTITY;

ENTITY GraphicalCard SUBTYPE OF (Card);
    videoMemory: MB;
END_ENTITY;

These concepts permit a good organization of the components in the model, though there is no way to define which component is the root.

It is also possible to declare its own named types and units, which help to clarify the meaning and the context of the variable of these types.

TYPE DOLLAR = INTEGER;
WHERE
    SELF >= 0;
END_TYPE;
Constraints can be declared both locally and globally in EXPRESS, though only local constraints are used in the implementation of the Home Multimedia Station. These local constraints are written within the entities using the \textit{WHERE} keyboard:

\begin{verbatim}
ENTITY WirelessMouse SUBTYPE OF (Mouse, WirelessDevice);
    buttonOnOff: BOOLEAN;
WHERE
    -- For equivalence <->
    NOT(buttonOnOff XOR (connectivityType = Bluetooth));
    ((connectivityType = Bluetooth) AND (pointingTechnology = laser) AND (price=35)) OR
    ((connectivityType = USB_dongle) AND (pointingTechnology = laser) AND (price=30)) OR
    ((connectivityType = Bluetooth) AND (pointingTechnology = optic) AND (price=25)) OR
    ((connectivityType = USB_dongle) AND (pointingTechnology = optic) AND (price=20));
END_ENTITY;
\end{verbatim}

The WirelessMouse entity points out two issues when writing constraints in EXPRESS. The first one concerns the support of product catalogs: as other languages shown in this report, the only formulation available for product catalogs is using a heavy logical disjunction. The second issue is the lack of built-in functions available: the example shows that the equivalence operator is not available, and a more complex logical function must be used. EXPRESS does neither provide functions like \textit{sum}, nor built-in navigation within collections (such as \textit{forAll} or \textit{forEach} functions).

It is however possible for the user to declare his own functions that can be used in the entities, for example to calculate the total price of the HomeMultimediaStation:

\begin{verbatim}
FUNCTION SumPrice(hms:HomeMultimediaStation):DOLLAR;
    LOCAL
        size: INTEGER := 0;
        result: DOLLAR := 0;
    END_LOCAL;
    size := SIZEOF(hms.accessories);
    REPEAT i:=1 TO size;
        result:=result + hms.accessories[i].price;
    END_REPEAT;
    size := SIZEOF(hms.computers);
    REPEAT i:=1 TO size;
        result:=result + hms.computers[i].price;
    END_REPEAT;
    result := result + hms.mouse.price + hms.keyboard.price;
    RETURN(result);
END_FUNCTION;
\end{verbatim}
Existing languages used for modeling

ENTITY HomeMultimediaStation;
    ...
DERIVE
    price : DOLLAR := SumPrice(SELF);
    ...
END_ENTITY;

Being able to declare functions extends the possibilities in the model, but it requires some programming skills. Thus it should be reserved for complex functions, and function like sum should be supported out-of-the-box.

Finally, other points need to be noticed:

- EXPRESS does not provide any support for connection ports or associations: the one in the Home Multimedia Station case study is defined in the RJ45port entity and declared as INVERSE reference in the RJ45connector. It however brings a relation of parent-child that should not be present in this kind of relations. It is also not possible to declare the Cable entity as part of the association between the RJ45port and the RJ45connector.

- EXPRESS does not provide any way to declare resource consumption and production.

- It is not possible to declare read-only attributes. On the other hand, derived attributes can be used as hidden attributes, though their value has to be directly associated to a function.

- The optimization function can be calculated using a function, but EXPRESS does not provide any support for declaring it as a cost function and/or specifying how it should be optimized.

- Finally soft constraints are not supported in EXPRESS.

The Storage System

In the implementation of the Storage System, it has been possible to declare user-defined constants, allowing an easier maintenance. The different constraints did not pose any problems, apart from the lack of operators: the implies operator has been replaced by a NOT...XOR expression, as the IF..THEN..ELSE expression cannot be used in local constraints:

ENTITY StorageSystem;
    color: COLOR;
    bookcase : BookCase;
    highcabinet : SET[0:1] OF HighCabinet;
lowcabinet : SET[0:1] OF LowCabinet;
WHERE
  HCExistence:
    NOT(SIZEOF(highcabinet) = 1) XOR
    ((SIZEOF(lowcabinet) = 1) AND (bookcase.height >=
      lowcabinet[1].height + MinHCHeight)));
END_ENTITY;

The definition of the domain of attributes such as the height of the BookCase also
had to be done through constraints, as it is not possible to declare it inline. Finally,
it was not possible to declare default values in EXPRESS; thus the height of the Low
Cabinet has thus been fixed.

Concerning the problem with layout constraints, the EXPRESS language does not
bring any more constructs that could be used to specify component ports and their
connections.

4.4.3 Discussion

EXPRESS is a powerful language for product modeling. It supports object-oriented
concepts and a complex multiple inheritance mechanism, which makes it suitable for
the modeling of most of the product configuration problem. This is complemented by
the possibility to use a full procedural programming language to define functions and
constraints on data instances. It also includes nice features such as units and constant
declaration or dynamic multiplicity. Another point is the possibility to produce a
graphical representation of the model defined in EXPRESS using EXPRESS-G. This
can be a real advantage when modeling, and so is worth being pointed out.

However, the EXPRESS language is too general and is not suitable for knowledge
engineers that are not expert in the language itself. The definition of functions
requires advanced programming skills, and writing a complex model without these
functions can be very difficult or impossible, as a lot of functions are not built-in
(min/max, sum, forAll, ...). Other product configuration specific features are also
missing, such as the declaration of a root element, read-only and default attributes,
support for optimization functions, resources, soft and table constraints or connection
ports definition.

Moreover, the use of EXPRESS for product configuration modeling is restricted.
The major problem comes from the STEP standard itself. EXPRESS is meant to
be used to describe application protocols as part of the standard, and should thus
not include company-specific data, such as company-specific configuration models.
Männistö et al. describe in [15] a way to extend STEP in order to permit the definition
of such models in STEP, where the AP would be used to declare the area-specific part of the model, while company-specific extension would be added as instances of the AP concepts. However, this could reduce the freedom of the modeler, as he would have to comply to a basic structure for its model, according to the corresponding AP. Other issues arise as well, such as the definition of constraints in the model, as a STEP schema is supposed to describe only valid instances [15].
This last chapter summarizes relevant insights from the evaluation of the modeling languages studied in this report. The aim is to compare these languages and to conclude on a personal opinion of the efficiency for each language.

5.1 Comparison of the languages based on the requirements

Table 5.1 retains the main insights retrieved all along this report for the different modeling languages.
<table>
<thead>
<tr>
<th>Language</th>
<th>Type</th>
<th>Features</th>
<th>Requirements</th>
<th>General use</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML/OCL</td>
<td>Graphical + textual</td>
<td>- Full OO support</td>
<td>- Clear view of the structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraints</td>
<td>- Dynamic multiplicity</td>
<td>- Need adaptation by the user using stereotypes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Extensible through stereotypes</td>
<td>- Structure and constraints are separated</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Complete constraint language</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Missing: units, table constraints optimization functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SysML</td>
<td>Graphical</td>
<td>- Inherits features from UML</td>
<td>- Most of product oriented syntax already in place</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mapping between blocks/constraints</td>
<td>- Support for other product specifications apart from product configuration ones</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Parametric constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Units &amp; optimization functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Some stereotypes still needed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Table constraints missing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPRESS</td>
<td>Textual + graphical</td>
<td>- Full OO support</td>
<td>- Possible graphical representation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>representation</td>
<td>- Dynamic multiplicity</td>
<td>- <em>Needs</em> programming skills</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Units and constants declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- User-defined functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Lack of built-in functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Missing: read-only and default attributes, table constraints,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>associations, resources, optimization functions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of the modeling languages
This table shows the differences between the languages, and first of all highlights the capabilities of the graphical languages, in both the clear view they provide and the completeness of their specifications. The difference of capabilities is greatly due to the fact that defining these languages with a lot of features is easy while interpreting them is more difficult. This is shown by the few research done in the interpretation of UML-based model for product configuration [1, 2].

However, a language like UML is too general for configuration and its concepts too specific to software engineering. That is why SysML has been introduced, and it brings some clarity on product-related concepts, adding interesting features that can be used for product configuration modeling.

The advantage of textual languages is obviously that they can be directly interpreted as such. They also usually make possible to create a graphical representation on top of them, like EXPRESS-G. EXPRESS provides a full support of all object-oriented featured discussed in the requirements section. However, the definition of the constraints on the entities is made difficult by the lack of built-in functions or navigation within the aggregation collections. The possibility to declare functions using a full procedural programming language permits to do almost everything, but it then requires more advanced programming skills. Moreover, the use of EXPRESS for configuration is restricted because of its belonging to the STEP standard (see Section 4.3.3).

It is also interesting to note that none of these languages did solve the problem with layout constraints in an easy way. This problem obviously requires the assistance of a graphical view, but the definition of component ports could have been an interesting option.

5.2 Conclusion

The goal of this report was to make an analysis of general modeling language, and to evaluate them according to requirements previously defined from both experience and literature. Two case studies have been used to be implemented in all the languages presented: one created specifically (the Home Multimedia Station) and one derived from the literature (the Storage System, from [3]).

This report highlighted the capabilities of several well-known languages to fit the requirements emitted for modeling in product configuration. It also permitted to get insights on their specific way to address the modeling challenges such as the declaration of connection ports/associations, resources, or the definition of the different types of constraints.
Appendix A

Additional SysML diagrams

This appendix contains additional diagrams used to implement the case studies in SysML.

A.1 Home Multimedia Station
Figure A.1: Parametric Diagrams for the Computer block

Figure A.2: Parametric Diagrams for the NetworkRouter block
Figure A.3: Parametric Diagrams for the ClassicMouse block

Figure A.4: Parametric Diagrams for the WirelessKeyboard block
A.2 Storage System

Figure A.5: Parametric Diagrams for the WirelessMouse block

Figure A.6: Parametric Diagrams for the BookCase block
Figure A.7: Parametric Diagrams for the HighCabinet block
Additional SysML diagrams
This appendix contains the source files used to implement the case studies in EXPRESS.

**B.1 The Home Multimedia Station**

1 (* Main schema *)
2 SCHEMA HomeMultimediaStationFactory;
3 USE FROM ComputerParts;
4 USE FROM Accessories;
5 USE FROM InputDevices;
7 (* User-defined functions *)
8 FUNCTION SumPrice(hms:HomeMultimediaStation):DOLLAR;
10 LOCAL
11 size: INTEGER := 0;
12 result: DOLLAR := 0;
13 END_LOCAL;
14 size := SIZEOF(hms.accessories);
15 REPEAT i:=1 TO size;
result := result + hms.accessories[i].price;
END_REPEAT;
size := SIZEOF(hms.computers);
REPEAT i:=1 TO size;
result := result + hms.computers[i].price;
END_REPEAT;
result := result + hms.mouse.price + hms.keyboard.price;
RETURN(result);
END_FUNCTION;

FUNCTION SumHDCapacity(hms: HomeMultimediaStation): GigaBytes;
LOCAL
size : INTEGER := 0;
result : GigaBytes := 0;
END_LOCAL;
size := SIZEOF(hms.accessories);
REPEAT i:=1 TO size;
IF VALUE_IN(TYPEOF(hms.accessories[i]),'ExternalHD') THEN
result := result + hms.accessories[i].capacity;
END_IF;
END_REPEAT;
size := SIZEOF(hms.computers);
REPEAT i:=1 TO size;
result := result + hms.computers[i].primaryHD.capacity;
END_REPEAT;
RETURN(result);
END_FUNCTION;

FUNCTION MaxPrimaryHDCapacity(hms: HomeMultimediaStation): GigaBytes;
LOCAL
size : INTEGER := 0;
max : GigaBytes := 0;
END_LOCAL;
size := SIZEOF(hms.computers);
REPEAT i:=1 TO size;
IF max < hms.computers[i].primaryHD.capacity THEN
max := hms.computers[i].primaryHD.capacity;
END_IF;
END_REPEAT;
RETURN(max);
END_FUNCTION;

FUNCTION CheckComplexConstraint(hms: HomeMultimediaStation): BOOLEAN;
LOCAL
size : INTEGER := 0;
maxPrimaryHD : GigaBytes := MaxPrimaryHDCapacity(hms);
END_LOCAL;
size := SIZEOF(hms.accessories);
REPEAT i:=1 TO size;
  IF VALUE_IN(TYPEOF(hms.accessories[i]),'ExternalHD') THEN
    IF maxPrimaryHD > hms.accessories[i].capacity THEN
      RETURN(false);
    END_IF;
  END_IF;
END_REPEAT;
RETURN(true);
END_FUNCTION;

(* User-defined types *)
TYPE DOLLAR = INTEGER;
WHERE
  SELF >= 0;
END_TYPE;

TYPE GigaBytes = INTEGER;
WHERE
  SELF >= 0;
END_TYPE;

(* Root component *)
ENTITY HomeMultimediaStation;
  mouse : Mouse;
  keyboard : Keyboard;
  accessories : SET OF Accessory;
  computers : SET[1:?] OF Computer;
  DERIVE
    price : DOLLAR := SumPrice(SELF);
    cost_function : REAL := price/SumHDCapacity(SELF);
  WHERE
    CheckComplexConstraint(SELF);
END_ENTITY;

(* Abstract base component for hard drives *)
ENTITY HardDrive ABSTRACT SUPERTYPE;
  capacity : GigaBytes;
WHERE
  (capacity = 20) OR (capacity = 80) OR (capacity = 160);
END_ENTITY;

END_SCHEMA;
(* Schema for computer parts *)

SCHEMA ComputerParts;

REFERENCE FROM HomeMultimediaStationFactory;

TYPE MB = INTEGER;
WHERE
   SELF >= 0;
END_TYPE;

TYPE INCH = REAL;
WHERE
   SELF >= 0;
END_TYPE;

TYPE INTERFACE_TYPE = ENUMERATION OF (PCI, Integrated);
END_TYPE;

TYPE NETWORKCARD_TYPE = ENUMERATION OF (Wifi_adapter, Ethernet);
END_TYPE;

ENTITY Computer;
   price: DOLLAR;
   primaryCard: GraphicalCard;
   additionalCards: SET OF Card;
   primaryHD: InternalHD;
WHERE
   ((primaryHD.capacity = 20) AND (SIZEOF(additionalCards) = 0) AND (price = 400)) OR
   ((primaryHD.capacity = 20) AND (SIZEOF(additionalCards) = 1) AND (price = 450)) OR
   ((primaryHD.capacity = 80) AND (SIZEOF(additionalCards) = 0) AND (price = 800)) OR
   ((primaryHD.capacity = 80) AND (SIZEOF(additionalCards) = 1) AND (price = 850)) OR
   ((primaryHD.capacity = 160) AND (SIZEOF(additionalCards) = 1) AND (price = 950)) OR
   ((primaryHD.capacity = 160) AND (SIZEOF(additionalCards) = 2) AND (price = 1100));
END_ENTITY;

ENTITY Card ABSTRACT SUPERTYPE;
   interfaceType: INTERFACE_TYPE;
END_ENTITY;

ENTITY GraphicalCard SUBTYPE OF (Card);
B.1 The Home Multimedia Station

videoMemory: MB;
WHERE
(videoMemory = 128) OR (videoMemory = 256) OR (videoMemory = 512);
END_ENTITY;

ENTITY NetworkCard SUBTYPE OF (Card);
  typeCard: NETWORKCARD_TYPE;
  connector: RJ45connector;
END_ENTITY;

ENTITY RJ45connector;
INVERSE
  associatedPort: RJ45port FOR associatedConnector;
  parentCard: NetworkCard FOR connector;
END_ENTITY;

ENTITY InternalHD SUBTYPE OF (HardDrive);
  size: INCH;
WHERE
(size = 1.0) OR (size = 2.5) OR (size = 3.0);
END_ENTITY;
END_SCHEMA;

(* Schema for accessories*)
SCHEMA Accessories;
REFERENCE FROM HomeMultimediaStationFactory;

TYPE BATTERY_TYPE = ENUMERATION OF (Li_Ion_Battery, AA_Batteries);
END_TYPE;

TYPE PACKAGETYPE_TYPE = ENUMERATION OF (Office, Photo_editing, Games);
END_TYPE;

TYPE VERSION_TYPE = ENUMERATION OF (Basic, Pro);
END_TYPE;

ENTITY Accessory ABSTRACT SUPERTYPE;
  price: DOLLAR;
END_ENTITY;
ENTITY SoftwarePackage SUBTYPE OF (Accessory);
packageType: PACKAGETYPE_TYPE;
version: VERSION_TYPE;
size: GB;
WHERE
  1 <= size <= 20;
  ((packageType = Office) AND (version = Basic) AND (size = 12)
  AND (price = 400)) OR
  ((packageType = Photo_editing) AND (version = Pro) AND (size = 6)
  AND (price = 700)) OR
  ((packageType = Games) AND (version = Basic) AND (size = 19)
  AND (price = 650))
END_ENTITY;

ENTITY ExternalHD SUBTYPE OF (Accessory, HardDrive);
esSATA_connection: BOOLEAN;
firewire_connection: BOOLEAN;
usb_connection: BOOLEAN;
WHERE
  price = 40;
END_ENTITY;

ENTITY NetworkRouter SUBTYPE OF (Accessory);
wireless: BOOLEAN;
ports: SET [3:3] OF RJ45port;
WHERE
  price = 50;
END_ENTITY;

ENTITY RJ45port;
  associatedConnector: SET [0:1] OF RJ45connector;
INVERSE
  parentRouter: NetworkRouter FOR ports;
WHERE
  (SIZEOF(associatedConnector)=0) XOR (NOT(parentRouter.wireless
  XOR (associatedConnector[1].parentCard.typeCard =
  Wifi_adapter)));
END_ENTITY;

END_SCHEMA;

(* Schema for input devices *)
B.1 The Home Multimedia Station

```plaintext
SCHEMA InputDevices;
REFERENCE FROM HomeMultimediaStationFactory;

TYPE POINTING_TECHNOLOGY = ENUMERATION OF (optic, laser);
END_TYPE;

TYPE KEYBOARD_LAYOUT = ENUMERATION OF (French_AZERTY, US_QWERTY, Danish_QWERTY);
END_TYPE;

TYPE CONNECTIVITY_TYPE = ENUMERATION OF (Bluetooth, USB_dongle);
END_TYPE;

ENTITY Mouse ABSTRACT SUPERTYPE;
  pointingTechnology : POINTING_TECHNOLOGY;
  price : DOLLAR;
END_ENTITY;

ENTITY WirelessMouse SUBTYPE OF (Mouse, WirelessDevice);
  buttonOnOff : BOOLEAN;
  WHERE
  -- For equivalence <->
  NOT (buttonOnOff XOR (connectivityType = Bluetooth));
  ((connectivityType = Bluetooth) AND (pointingTechnology = laser) AND (price=35)) OR
  ((connectivityType = USB_dongle) AND (pointingTechnology = laser) AND (price=30)) OR
  ((connectivityType = Bluetooth) AND (pointingTechnology = optic) AND (price=25)) OR
  ((connectivityType = USB_dongle) AND (pointingTechnology = optic) AND (price=20));
END_ENTITY;

ENTITY ClassicMouse SUBTYPE OF (Mouse);
  WHERE
  ((pointingTechnology = laser) AND (price=15)) OR
  ((pointingTechnology = optic) AND (price=10));
END_ENTITY;

ENTITY Keyboard;
END_ENTITY;

ENTITY WirelessKeyboard SUBTYPE OF (Keyboard, WirelessDevice);
  WHERE
  ((connectivityType = Bluetooth) AND (price=30)) OR
  ((connectivityType = USB_dongle) AND (price=25));
```
B.2 The Storage System

```plaintext
SCHEMA StorageSystemFactory;

CONSTANT
MinBCHeight : cm := 72.0;
MaxBCHeight : cm := 216.0;
MinHCHeight : cm := 50.0;
MaxHCHeight : cm := 144.0;
END_CONSTANT;

TYPE COLOR = ENUMERATION OF (Painted, Wood);
END_TYPE;

TYPE cm = REAL;
WHERE
  not_negative: SELF >= 0;
END_TYPE;

ENTITY StorageSystem;
  color: COLOR;
  bookcase : BookCase;
  highcabinet : SET[0:1] OF HighCabinet;
  lowcabinet : SET[0:1] OF LowCabinet;
WHERE
  HExistence:
    NOT(SIZEOF(highcabinet) = 1) XOR
    ((SIZEOF(lowcabinet) = 1) AND (bookcase.height >=
      lowcabinet[1].height + MinHCHeight));
END_ENTITY;

ENTITY BookCase;
  height: cm;
```
WHERE
c1: \{ \text{MinBCHeight} \leq \text{height} \leq \text{MaxBCHeight} \};
END_ENTITY;

ENTITY HighCabinet;
  height: \text{cm};
  shelves: \text{SET}[0:3] \text{OF RollOutShelf};
  drawers: \text{SET}[0:3] \text{OF Drawer};
WHERE
c2: \{ \text{MinHCHeight} \leq \text{height} \leq \text{MaxHCHeight} \};
  quantityLimit: \text{SIZEOF}(\text{shelves}) + \text{SIZEOF}(\text{drawers}) \leq 4;
END_ENTITY;

ENTITY LowCabinet;
  height: \text{cm};
WHERE
  fixed: \text{height} = 72.0;
END_ENTITY;

ENTITY Drawer;
ENDENTITY;

ENTITY RollOutShelf;
ENDENTITY;

END_SCHEMA;


