Integration of OntoClean in *XOL*
XOL++

Laboratório de Métodos Formais, Matemática e Ciências da Computação,
Departamento de Informática, Universidade do Minho, Campus de Gualtar. Braga.
{marlene.azevedo}@correio.di.uminho.pt

7th March 2003

”*What is essential is invisible to the eyes.*”
by Antoine de Saint-Exupery

Dedication

To my mother and my sisters Elisabete and Rosa.
To all my colleagues of course.
And to my beloved Marco.

Acknowledgements

I want to leave registered my gratitude to all people who somehow had collaborated so that this work arrived at the end.

In first place, I'd like to thank the chance that the professor José Nuno Oliveira gave me participating in this project.

Finally, I want to thank in a special way my friends Nuno Rodrigues and Cândida Silva for the patience and advice dispensed.

Contents

Abstract

This project emerged by the framework of the European *EUREKA* project E!2235 "*IKF*" (Intelligent Knowlege Fusion) that, at this moment in Portugal, counts on the participation of the Computer Science Department of the Minho University, in the quality of "Technology Producers". Therefore, and in this scope, the *XOL++* project was developed for the school discipline *Laboratório de Métodos Formais* lectured in the 5^{th} year of *Matemática e Ciências de Computação* Degree [1].

The main goal of *EUREKA* project *IKF* consists on the analysis, drawing and implementation of an surrounding innovator of Knowledge Warehousing, making use of advanced functionalities of Knowledge Management and Business Intelligence, guided for specifics vertical application domains. This way, problems like the Information Acquisiton Model, the *IKF-CW* repository, the specification of Topic Maps or the specification of an ontology are in the line of the *EUREKA* project aim.

[1] http://www.di.uminho.pt/ ˜jno/html/labmf.html#sec:0203

1 Introduction

The *Xol++* project presents the results of using the *XOL* specification [11] with a well-founded methodology for ontological analysis called *OntoClean* [1].
Therefore, after a brief overview about the *XOL* Language [11] and the *OntoClean* method, I present the *DTD* (Document Type Definition) specification of *XOL* merged with a set of OntoClean meta-properties that, in my opinion, will help to perceive and clarify the nature of many ontological choices.

I'm certain that my work is very ambitious, as it necessarily faces already debated philosophical and technical problems. So I've tried to be as humble as possivel, making drastic simplifications whenever possivel, but trying however to save the logical rigor. The most notorious simplification I've made is related to the treatment of time (in meta-properties), which is very difficult to specify in *VDM++*, the others simplifications will be treated in the following sections.

1.1 Motivation

In the last years, the study on *Topic Maps* (TMs) has increased significantly. *TMs* provide a new powerful way to organise and navigate large quantities of information. The model is simple and flexible, enabling a wide range of potential applications and it can be seen as an intelligent index providing easy access to resources. That is, a *TM* is a model for mapping 'things' into data structures. All these things are 'topics' in *TM*; topics are connected through 'associations'; 'occurrences' are the places where these topics are found.
Inside of this context, if we speak in *TMs*, we speak in ontologies, because these are implicitly united to *TMs*. Besides, all *TMs* have associated, behind, the respective ontology. The project*EUREKA*, in which this work is inserted, is developing a deep study on *TMs*, so, if the idea of making the *TMs VDM++* specification appeared, then, and almost obligatory, it will also have to appear the ontologies *VDM++* specification.

2 The Ontology concept

An ontology is, simply, a specification of a conceptialization, that is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. It is different from the way the word is used in philosophy[2]. What is important is what the ontology is for.

Ontologies are about languages for expressing contracts between entities. Ontologies provide a way of capturing a shared understanding of terms that can be used by humans and programs to aid in information exchange and gives a method of providing a specification of a controlled vocabulary. For example, a taxonomy provide notions of generality and term relations, but classical ontologies attempt to capture precise meanings of terms. In order to specify meanings, an ontology language must be used.

Taxonomies are central to most ontologies. Well structured taxonomies help in bringing substancial order to elements of a model. They are particularly useful in presenting limited views of a model for human interpretation, and play a critical role in reuse and integration tasks. By contrast, structured taxonomies have the opposite effect, making models confusing and difficult to reuse or integrate.

[2]In philosophy an ontology is an systematic accout of Existence

3 XOL language

Before documenting *XOL* specification is necessary introduce some basic notions like: introduce *XOL*, examine how *XML*[3] can be used to express Ontology Languages and compare *DTD* and Ontologies because it will complement the next section however for a dip approach of these and others items is vital to consult [11]. Therefore:

XOL

The ontology definitions that *XOL* is designed to encode include both schema information (meta-data), such as class definitions from object databases - as well as non-schema information (ground facts) such as object definitions from object databases.

The syntax of *XOL* is based on *XML*. The modeling primitives and semantics of *XOL* are based on *OKBC-Lite*, which is a simplified form of the knowledge model for the Open Knowledge Base Connectivity (*OKBC*)[4]. *OKBC* is an application program interface for accessing frame knowledge representation systems. Its knowledge model supports features most commonly found in knowledge representation systems, object databases, and relational databases. *OKBC-Lite* extracts most of the essential features of *OKBC*, but omits some of its more complex aspects. *XOL* was inspired by Ontolingua. *XOL* differs from *Ontolingua*, however, as it has an *XML-based* syntax rather than a *Lisp-based* syntax. Still, the semantics of *OKBC-Lite* which underly *XOL* are extremely similar to the semantics of *Ontolingua*.

The design of *XOL* deliberately uses a generic approach to define ontologies, meaning that the single set of *XML* tags (defined by a single *XML DTD*) defined for *XOL* can describe any and every ontology. This approach contrasts with the approaches taken by other *XML* schema languages, in which a generic set of tags is typically used to define the schema portion of the ontology and the schema itself is used to generate a second set of application-specific tags (and an application-specific *DTD*), which in turn are used to encode a separate *XML* file that contains the data portion of the ontology. *XOL* appears rather promising because it provides ontological modeling primitives expressed in one of the most important information exchange standards: *XML*.

XML and Ontology Languages

XML is a tag-based language for describing tree structures with a linear syntax. It is a successor of *SGML*, which was developed long ago for describing document structures. *XML* provides semantic information as a by-product of defining the structure of the document. It prescribes a tree structure for documents and the different leafs of the tree have well-defined tags and contexts in which the information can be understood. That is, structure and semantics of document are interwoven.

Comparing DTD and Ontologies

The closest thing that *XML* offers for ontological modeling is the Document Type Definition (*DTD*) which defines the legal nestings of tags and introduces attributes

[3] http://www.w3.org/XML/
[4] http://www.ai.sri.com/~okbc/

for them. Defining tags, their nesting, and attributes for tags may be seen as defining an ontology. However, there are significant differences between an ontology and an *DTD*:

1. A *DTD* specifies the legal lexical nesting in a document, which may or may not coincide with an ontological hierarchy (subclass relationship). That is, there is nothing in a *DTD* that corresponds to the is-a relationship of classes that is usually central in an ontology.
2. In consequence, *DTDs* lack any notion of inheritance. In an ontology, subclasses inherit attributes defined for their super classes and super classes inherit instances defined for their subclasses. Both inheritance mechanisms do not exist for *DTDs*.
3. *DTDs* provide a rather poor means for defining the semantics of elementary tags. Basically, a tag can be defined as being composed of other tags or being a string. Usually, ontologies provide a much richer typing concept for describing elementary types.
4. *DTDs* define the order in which tags appear in a document. For ontologies, in contrast, the ordering of attribute descriptions does not matter.

In a nutshell, *DTDs* are rather weak in regard to what can be expressed with them. Work on *XML-schemes* may well contribute to bridging the gap between *DTDs* and ontologies. *DTDs* are therefore translated automatically into a representation of an ontology in description logic. This ontology simply consists of each element in the *DTD*.

3.1 XOL Specification

It is important to relate that, nor all *XML* documents are valid *XOL* documents, insofar as only *XML* documents that follow the structure of *XOL DTD* are considered valid, however all *XOL* documents must be valid *XML* documents.
Therefore, the *XOL DTD* specification in appendix A embrace all the considerations about *XOL* document and according that I will describe how *XOL* define ontologies and parallelly I will justify the options taken for the respective *VDM++* specification.

3.1.1 Ontology

In a first appreciation of the *DTD* we verify that *XOL* document begins with the **ontology** element (but could begin optionally by any of these five elements: **module, ontology, kb, database,** or **dataset**), which identifies the single ontology contained in that *XOL* file.
The **name** element within the ontology is required and specifies the name of the ontology. The remaining ontology elements (**kb-type, db-type, package, version** and **documentation**) are optional (but if provided, they must be inserted in this position, and in this order).
Then come a number of **class** elements, which define classes within that ontology. A series of **slot** elements list the slots that are defined on those classes. Finally, a series of **individual** elements define the objects within the ontology.
The follow example illustrate, in a *XML* language, how could be defined an ontology.

Exemplo 1

```
<ontology>
  <name>...</name>
  <kb-type>...</kb-type>
  <package>...</package>
  <version>...</version>
  <documentation>...</documentation>
  <class>...</class>
  <class>...</class>
  ...
  <slot>...</slot>
  <slot>...</slot>
  ...
  <individual>...</individual>
  <individual>...</individual>
  ...
</ontology>
```

In this way, it was easy to arrive at the respective specification because it was enough to consider that an ontology aggregates an element **Name** and all the information of all **Classes, Slots** and **Individuals**. Optionally we could include the other elements.
The invariant that follows was defined with the intention to garantee some additional rules that will be treated aheah. Later and having in account the *DTD*, the name and the others optinal elements were defined as `Strings`.

3.1.2 Class

A *XOL* class is defined by the element **name** that is required and is the name of the class. Then the element documentation that is optional and provides **documentation** about the class. At last follow elements of three possible types: **subclass-of**, **instance-of** and **slot-values**. A slot-values is defined by the element **name** that is required and is the name of the slot-value. Then one or more **value** elements that specify the one or more values of that slot. Finally follow elements of eleven types: **facet-values, value-type, inverse, cardinality, maximum-cardinality, minimum-cardinality, numeric-minimum, numeric-maximum, some-values, collection-type** and **documentation-in-frame**.

Exemplo 2

```
<class>
<name>...</name>
<documentation>...</documentation>
<subclass-of>...</subclass-of>
</class>
```

The class element specification wasn't so linear, because all **Classes** information is gathered in a partial function mapping a class identifier to the respective class data: name,

documentation and the three possivel types.
This small alteration does not affect the *DTD* structure, instead guarantees the existence of several classes in an ontology requested in the previous specification.
This would be the specification deduced through the *DTD*, however I added another field to class data: **C-Id**, that defines the identifier of the ”father” class. Is important to underline the fact that this new field does not affect the *DTD* structure, but enriches it, so that helped in some invariantes definition.
The other aspect that is necessary to notice is the fact that I have considered the fields **subclass-of** and **instance-of**, equals to the field **C-Id** that defines the identifiers of the ”sons” classes .

3.1.3 Slot

A *XOL* slot is defined by the element **name** that is required and is the name of the slot. Then the element documentation that is optional and provides **documentation** about the slot. At last follow elements of ten possible types: **domain, slot-value-type, slot-inverse, slot-cardinality, slot-maximum-cardinality, slot-minimum-cardinality, slot-numeric-minimum, slot-numeric-maximum, slot-collection-type** and **slot-values**.

Exemplo 3

```
<slot>
  <name>...</name>
  <documentation>...</documentation>
  <domain>...</domain>
  <slot-value-type>...</slot-value-type>
</slot>
```

The previous example illustrates a possible definition of a slot. Its specification is analogous to the class specification assuming, on the one hand that the information is also gathered in a partial function mapping a slot identifier to the respective slot data: name, documentation and the ten possivel types (except the slot-values, they are all defined as `Strings`), on the other hand the added field that defines the identifier of the ”father” class, however, this is only complete after inserting a field relative to slot attribute.

The slot-values element aggregates the fields cited previously: the name, the documentation and the eleven possible types, that are all defined as `Strings` except the facet-values element that is defined by a name and a set of values.

3.1.4 Individual

A *XOL* individual is defined by the element **name** that is required and is the name of the individual. Then the element documentation that is optional and provides **documentation** about the individual. At last follow elements of two possible types: **type** and **slot-values**.
The follow example illustrate how could be defined an individual.

Exemplo 4

```
<individual>
  <name>...</name>
  <documentation>...</documentation>
  <instance-of>...</instance-of>
  <slot-values>
    <name>...</name>
    <value>...</value>
  </slot-values>
</individual>
```

Similarly the two previous specifications, all **Individuals** information is gathered in a partial function mapping a individual identifier to the respective individual data: name, documentation and the two possivel types.

3.1.5 Additional Rules

Inside the context of what was said relatively to comparison between *DTD*s and ontologies, is not excessive to point out that *DTD* formalism cannot express all of the rules necessary to define valid *XOL* documents, that is, *XML DTD*s do not have sufficient power to express all of the necessary constraints on the form of *XOL* documents. Therefore, the appendix A provides additional rules that *XOL* documents must follow.

In the VDM++ specification this rules are seen as invariants. I don't go to describe the specification of all they, in a first place because some had been impossible to specify and in a second place because the specification illustrates what really it is asked for. However, and in my point of view, the fifth rule is sufficiently complex to perceive, therefore I constructed a small diagram that it simplifies what it is intended.
The transistion E - F is redundant, because F already make reference to parents of C,

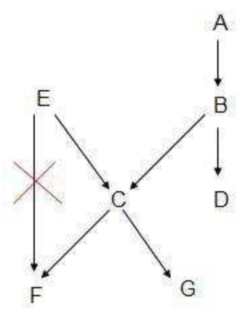

Figure 1: Diagram

where E is enclosed. The semantics load of this invariant is very significant, because it marks the difference between a *XML* document and a *XOL* document relatively to its woody structure. The first is always seen from top to bottom, while the second can also be seen of low for top.

4 OntoClean method

OntoClean was been elaborated by the Ontology Group of the LADSEB-CNR in Padova (Italy). It is a method to clean taxonomies according to notions such as: *rigidity, identity, unity* and *dependence*.

These notions provide a solid logical framework within which the properties that form a taxonomy can be analysed. This anaysis helps in rendering the intended meaning more explicit in improving human understanding and in reducing the cost of the integration. The definition of that notions refer to properties of properties and that are called **meta-properties**.

class $Meta$
$Meta\text{-}Properties$ :: $R : Rigidity$
$I : Identity$
$U : Unity$
$D : Dependence$

4.1 Rigidity

This notion is defined based on the idea of essence. A property is essencial to an individual if and only it is necessary to that undividual.

- A property ϕ is *rigid*, marked with (+R), if and only if is necessarily essential to all instances:

$$\Box(\forall x, a \;\; \phi(x,a) \rightarrow \Box\forall b \;\; \phi(x,b))$$

- A property is *non-rigid*, marked with (-R), if and only if is not essential to some of its instances:

$$\Diamond(\exists\, x \cdot, a \;\; \phi(x,a) \bigwedge \Diamond\exists\, b \cdot \;\; \neg\phi(x,b))$$

- Finally, a property is *anti-rigid*, marked with (˜ R), if and only if is not essential to all its instances.

$$\Box(\forall x, a \;\; \phi(x,a) \rightarrow \Diamond\exists\, b \cdot \;\; \neg\phi(x,b))$$

For example, the concept `person` is usually considered rigid, since every person is essentially such, while the concept `student` is not normally considered anti-rigid, since every student can possibly be a non-student a few years later.

$Rigidity$ = RIGID | NON_RIGID | ANTI_RIGID;

4.2 Identity

An *identity condition*, IC, is necessary if it satisfies (1) and sufficient if it satisfies (2), and need not be both:

1. $\Box(\exists(\cdot x, t) \bigwedge \phi(x,t) \bigwedge \exists(\cdot y, t') \bigwedge \phi(y,t') \bigwedge x = y \rightarrow \sum(x,y,t,t'))$

2. $\Box(\exists(\cdot x, t) \bigwedge \phi(x,t) \bigwedge \exists(\cdot y, t') \bigwedge \phi(y,t') \bigwedge \sum(x,y,t,t') \rightarrow x = y)$

- A property *carries an IC*, marked with (+I or -I otherwise) if and only if all its instances can be (re)identified by means of situable ”sameness” relation.
- A property *supplies an IC*, marked with (+O or -O otherwise) if and only if such criterion is not inherited by any subsuming property.

For example, `person` is usually considered a supplier of an identity criterion (for example the fingerprint), while `student` just inherits the identity criterion of `person`, whithout supplying any further identity criteria.

- Any property *carries* an IC iff it is subsumed by a property supplying this IC;
- A property ϕ *supplies* an IC iff
  1. it is rigid;
  2. there is an IC for it;
  3. the same IC is not carried by all the properties subsuming ϕ. This means that, if fi inherits different ICs from multiple properties, it still counts as supplying an IC.
- An property carrying an IC is called a *sortal*.

$$\textit{Identity} = \text{CARRIES\_IC} \mid \text{NOTCARRIES\_IC} \mid \text{SUPPLIES\_IC} \mid \text{NOTSUPPLIES\_IC};$$

4.3 Unity

An individual is a *whole* if and only if it is made by a set of parts unified by relation R. For example, the enterprise Iberia is a whole because it is composed by a set of people that are linked by the relation `have the same president`.

- A property P is said *carry unity*, marked with (+U or -U otherwise), if there is a *common* unifying relation R such that all the instances of P are wholes under R. For example, the concept `enterprise-with-president` carries unity because every enterprise with president is made up people linked through the relation `having` the same `president`.
- A property carries *anti-unity*, marked with (˜U) if all its instances can possible be non-wholes (˜U implies -U). Properties that refer to amounts of matter, like gold, water, etc., are good examples of anti-unity.

Depending on the ontological nature of the R relation, which can be understood as a ”generalized connection”, we may distinguish three main kinds of unity for concret entities (i.e., those having a spatio-temporal location). Briefly, these are:

- `Topological unity`: based on some kind of topological or phisycal connection, such as the relationship between the parts of a piece of coal or an apple.
- `Morphological unity`: based on some combination of topological unity and shape, such us a ball, or a morphological relation between wholes such as for a constellation.
- `Functional unity`: based on a combination of other kinds of unity with some notion of purpose as with artifacts such as hammers, or a functional relation between wholes as with artifacts such as a bikini.

$Unity$ = CARRIES_UC | NOTCARRIES_UC | ANTI_UNITY;

4.4 Dependence

An individual *x* is constantly dependent on *y* if and only if, at any time, *x* cannot be present unless *y* is fully present, and *y* is not part of *x*.

$$\forall x \Box (\phi(x) \rightarrow \exists\, y \cdot \psi(y) \bigwedge \neg P(y, x) \bigwedge \neg C(y, x))$$

For example, a hole in a wall is constantly dependent on the wall. The hole cannot be present if the wall is not present. A property P is constantly dependent if and only if, for all its instances, there exists something they are constantly dependent on. For instance, the concept `hole` is constantly dependent. A dependent property is marked with +D (or -D otherwise).

$Dependence$ = DEPENDENT | NON_DEPENDENT
end *Meta*

4.5 Constraints and assumptions

The first observation descending immediately from the last definitions regards some subsumption constraint. If ϕ and ψ are two properties then the following constraints hold:

1. $\phi^{\tilde{R}}$ must subsume $\psi^{\tilde{R}}$, i.e., ϕ^{+R} can't subsume $\psi^{\tilde{R}}$
2. ϕ^{+U} must subsume ψ^{+U}, i.e., ϕ^{-U} can't subsume ψ^{+U}
3. $\phi^{\tilde{U}}$ must subsume $\psi^{\tilde{U}}$, i.e., ϕ^{+U} can't subsume $\psi^{\tilde{U}}$
4. ϕ^{+I} must subsume ψ^{+I}, i.e., ϕ^{-I} can't subsume ψ^{+I}

5. ϕ^{+D} must subsume ψ^{+D}, i.e., ϕ^{-D} can't subsume ψ^{+D};
6. Properties with imcomplete ICs/UCs are disjoint.

All these constraints but the last one are specifed more ahead through the following invariants: *antiRig, unity, antiUnity, ident* and *depend*, respectively.

Therefore, the formal ontology of properties distinguishes eight different kinds of properties based on the valid and most useful combinations of the meta-properties (see Figure 2). These property kinds enrich a modeler's ability to specify the meaning of properties in an ontology, since the definition of each property kind includes an intuitive and domain-independent description of how this kind of property should be used in an ontology.

Sortal	Type	+D	+R	+I	+O
		-D			
	Quasy-type	+D	+R	+I	-O
		-D			
	Material role	+D	~R	+I	-O
	Phased sortal	-D	~R	+I	-O
	Mixin	+D	¬R	+I	-O
		-D			
Non-sortal	Category	+D	+R	-I	-O
		-D			
	Formal role	+D	~R	-I	-O
	Attribuition	-D	~R	-I	-O
		+D	¬R		
		-D			
	Incoherent			-I	+O
			~R	+I	
			-R		

Figure 2: All possible combinations of the meta-properties

Other assumptions must be considered:

- *Sortal individuation*: every domain element must instantiate some property carrying an IC (+I).
- *Sortal expandability*: if two entities (instances of different properties) are the same, they must be instances of a property carrying a condition for their identity.

4.6 Metodology

The specific steps to clean the wrong subclass of links in a taxonomy are:

1. *Put tags to every property assigning meta-properties*. This eases the analysis, because all the meta-properties are simultaneously visible.
2. *Focus just on the rigid properties*. A taxonomy without rigid properties is called *backbone taxonomy*. It is the base of the rest of the taxonomy, that is, the essential part.

3. *Evaluate the taxonomy taking into account principles based on the meta-properties*. For instance, a rule suggested in OntoClean is ” a property carrying anti-unity has to be disjoint of a property carrying unity”. As consequence, ”a property carrying unity cannot be a subclass of a property carrying anty-unity”. Therefore, `bronze statue` (it carries unity) cannot be a subclass of `bronze` (it carries anti-unity), for example.
4. *Consider non-rigid properties*. When the backbone taxonomy has been examined, the modeler has to evaluate the non-rigid properties. One of the proposed rules is: ”a non-rigid property and a anti-rigid property are ever disjoint”. A consequence, ”a non-rigid property cannot be a subcalss of an anti-rigid property”. Therefore, `person` (rigid) cannot be a subclass of `student` (anti-rigid).
5. *Complete the taxomony with other concepts and relations*. There can be several reasons to introduce new concepts. One of them is the transformation of concepts in relations, for example, `student` could be transformed into a relation between `person` and `university`.

5 XOL++

The OntoClean method is very simple, but very powerful.
Simple because is summarized in specifying the meta-properties type that aggregate the properties: **rigidity, identity, unity** and **dependence**, and these are therefore expressed according its definition (see sections 4.1, 4.2, 4.3 and 4.4), as you can observe in appendix C1.2.
Powerful because is enough:

- to add the field meta-properties to the class data;
- to have in attention the constraints and the assumptions focused previously that will provoke the specification of new invariantes;
- to consider some basic design principles:
  1. be clear about the domain
     - particulars(individuals);
     - universals (classes and relations);
     - linguistic entities (nouns, verbs, adjectives...);
  2. take identity seriously
     - different identity criteria imply disjoint classes;
  3. isolate a baic taxonomic structure
     - only sortals like ”person” (as opposite to ”red”) are good candidates for being taxons;
  4. make an explicit distinction between types and roles (and other property kinds);

and we get a well-founded ontology.
If to the *XOL DTD* specification we increase these three previous points we obtain the *XOL++*.

This is an excellent *recipe*, and goes to the encounter of what was proposed me, however is important to underline that it is *one drop of water in an ocean* of several researches on the ontologies study preconised by innumerable experts.

5.1 An ontology-cleaning example

In this section I provide a brief example of the way my analysis can be used. A complete version of this example is available in [5]. We begin with a set of properties arranged in a taxonomy, as shown in Fig.3.

[5]http://www.cs.vassar.edu/faculty/welty/aaai-2000

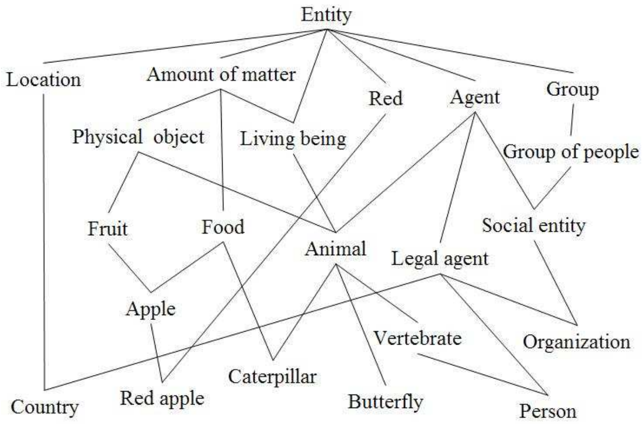

Figure 3: A messy taxonomy

And assign the meta-properties.

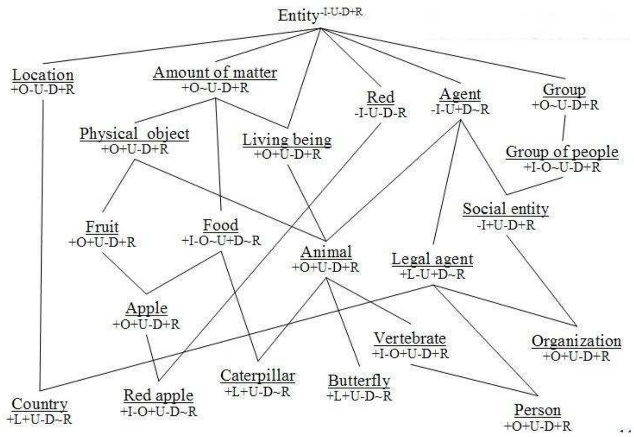

Figure 4: A messy taxonomy

Then we must remove the non-rigid properties and analyze taxonomic links:

˜U can't subsume +U

> **Living being** can change parts and remain the same, but **amounts of matter** can not (incompatible ICs) so **living being** is constituted of matter.
> **Physical objects** can change parts and remain the same, but **amounts of matter** can not (incompatible ICs) so **physical object** is constituted of matter.
> Meta-properties are fine but identity-check fails: when an entity stops being an **animal**, it does not stop being a **physical object** (when an animal dies, its body remains). Therefore we have to give attention to the constitution.
> A **group**, and **group of people**, can't change parts - it becomes a different group.

A **social entity** can change parts - it's more than just a group (incompatible IC). Constitution again.

After this analysis, we obtain one taxonomy slightly different.

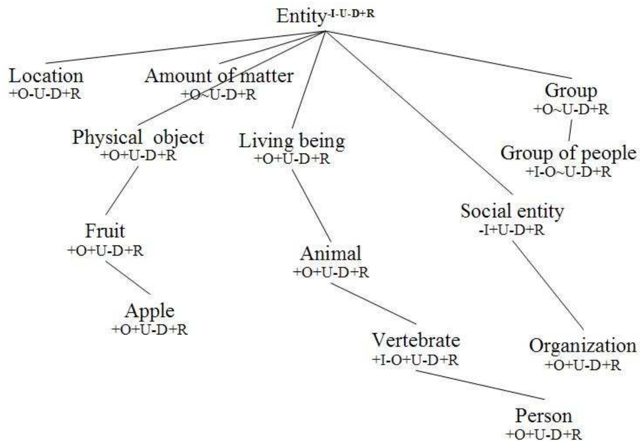

Figure 5: Taxonomic links analyzed

The next step is analyze the non-rigid properties:

˜R can't subsume +R

Really want a type restriction: all **agents** are **animals** or **social entities**. Subsumption is not disjunction!
Another disjunction is that all legal **agents** are **persons** or **organizations**.
Apple is not necessarily **food**. A poison-apple, e.g., is still an apple. ˜U can't subsume +U, so **caterpillars** are wholes, **food** is stuff.
Checking identity we verify that a **location** can't change parts... 2 senses of **country**: **geographical region** and **political entity** so we split the two senses into two concepts, both rigid, both types. There is a relationship between the two, but not subsumption.

Looking for missing types we notice that **caterpillars** and **butterflies** cannot be **vertebrate**. There must a rigid property that subsumes the two, supplying identity across temporary phases: **Lepidopteran**.

Finally we analyze attributions and there is no violations. Attributions are discouraged and can be confusing so, often, is better to use attribute values (i.e. Apple color red).
The final corrected taxonomy is shown in Fig.6.

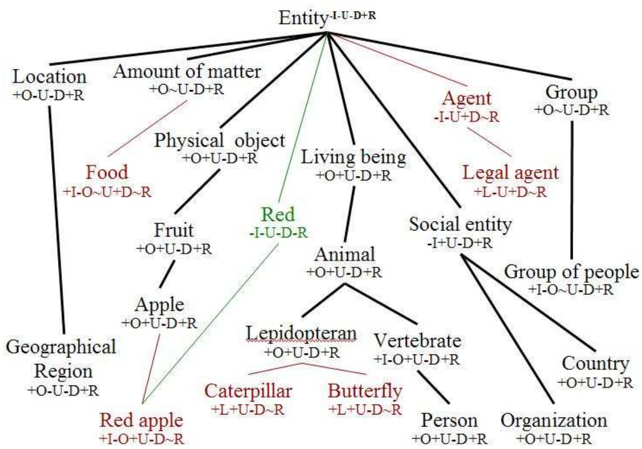

Figure 6: The backbone taxonomy

This example is too much generic and only illustrates the impact of taxonomic constraints on ontology design. That is, the stratification replaces the multiple inheritance in many cases: simpler taxonomies, moderate proliferation of individuals and co-localization of entities of different kind.
Non-taxonomic relations, become important: dependence, co-localization, constitution and participation.

In this way, and with the intention to corroborate my data type the class Example is specified (in apendix C.4) and translates the example supplied in *XML* (this also can be consulted in annex in apendix E).

Comparativily with what it was said, we could insert the different values: class, slot, individual through the respective functions of insertion, however a bad meta-properties definition reports us for the same problems cited previously. The user is, therefore collated with error messages provoked by the invariantes definition. Therefore:

Use OntoClean for all your ontology cleaning needs!

6 Conclusion

Developing a well-founded ontology is a very difficult task, that requires a carefully designed methodology and rigorous formal framework. I hope to have contributed on both these aspects, presenting in this report the *XOL++* that implements OntoClean, the method to clean Ontologies elaborated in the LADSEB-CNR of Padua(Italy).

The *XOL++* has been built using as base *XOL DTD*, specified in VDM++. I have used the *XML* language to exemplify how I add it into the OntoClean evaluation rules before specifying it in VDM++.
The main contributions that my work has accomplished are:

- to provide a stronger ontological commitments in order to get a ”disciplined” taxonomy;
- to reduce the risk of classification mistakes in the ontology development process;
- to simplify the update and maintenance process.

The knowledge used to evaluate ontologies is formally specified, which means that new meta-properties could be added easily by just introducing new elements in the meta-properties field. New invariants could also be added or modified to enrich the specification.

6.1 Future Work

The OntoClean is use in severel places. At the Italian National Research Council Laboratories (LADSEB-CNR and ITBM-CNR), in Padua and Rome, OntoClean is used in the development of an upper-level ontology based on a restructuring of WordNet project. Adding OntoClean top-level to *XOL++* will bring added value to the specification.

The example that corroborated the data type is merely academic, but in case it has great dimensions would be more viable to generate the values automatically. With the purpose to reach this objective I started for generating the schema (see appendix D) of the already defined *DTD* and applied it an example. The following step would be to construct one stylesheet that applied to this schema originate an output file with the values generated automatically. It was, in fact, a very simple process, but as my specification does not respect the encapsulation rules I would have to modify almost all specification, what I didn't considered very viable staying for a future work.

A XOL DTD Specification

```
<!ELEMENT (module | ontology | kb | database | dataset)
          (name,  ( kb-type | db-type )?, package?, version?,
          documentation?, class*, slot*, individual*)>
```

– module, ontology, kb, database, dataset are all synonimous

class $xol\text{-}Ontology$

$$
\begin{aligned}
Ontology :: \; & N : Name \\
& C1 : [Kb\text{-}type \mid Db\text{-}type] \\
& P : [Package] \\
& V : [Version] \\
& D : [Documentation] \\
& C : Classes \\
& S : Slots \\
& I : Individuals
\end{aligned}
$$

inv $ont \triangleq$

$$
\begin{aligned}
& snameCC\,(ont) \wedge snameSS\,(ont) \wedge snameII\,(ont) \wedge \\
& snameCS\,(ont) \wedge \\
& snameCI\,(ont) \wedge snameSI\,(ont) \wedge \\
& subClass\,(ont) \wedge \\
& transClass\,(ont);
\end{aligned}
$$

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT kb_type (#PCDATA)>
<!ELEMENT Db_type (#PCDATA)>
<!ELEMENT Versin (#PCDATA)>
<!ELEMENT documentation (#PCDATA)>
```

$Name = \text{char}^*;$
$Kb\text{-}type = \text{char}^*;$
$Db\text{-}type = \text{char}^*;$
$Package = \text{char}^*;$
$Version = \text{char}^*;$
$Documentation = \text{char}^*;$

```
<!ELEMENT class ( (name, documentation?, ( subclass-of |
                                    instance-of | slot-values)*)>
```

$Classes = C\text{-}Id \xrightarrow{m} Class;$

$C\text{-}Id = \text{token};$

$$\begin{aligned} \mathit{Class} :: N &: \mathit{Name} \\ D &: [\mathit{Documentation}] \\ C2 &: (\mathit{C\text{-}Id} \mid \mathit{Slot\text{-}values})\text{-set} \\ P &: [\mathit{C\text{-}Id}\text{-set}] \end{aligned}$$

```
<!ELEMENT subclass-of (#PCDATA)>
<!ELEMENT instance-of (#PCDATA)>
```

$$\begin{aligned} &\mathit{Subclass\text{-}of} = \text{token}; \\ &\mathit{Instance\text{-}of} = \text{token}; \end{aligned}$$

```
<!ELEMENT slot
  (name, documentation?,
     ( domain |
       slot-value-type | slot-inverse |
       slot-cardinality |
       slot-maximum-cardinality |
       slot-minimum-cardinality |
       slot-numeric-minimum |
       slot-numeric-maximum |
       slot-collection-type |
       slot-values )* >
```

$$\mathit{Slots} = \mathit{S\text{-}Id} \xrightarrow{m} \mathit{Slot};$$

$$\mathit{S\text{-}Id} = \text{token};$$

$$\begin{aligned} \mathit{Slot} :: N &: \mathit{Name} \\ D &: [\mathit{Documentation}] \\ C3 &: \mathit{Slot\text{-}Ch}\text{-set} \\ A &: \mathit{SlotAtt} \\ P &: [\mathit{C\text{-}Id}\text{-set}] \end{aligned}$$

$$\begin{aligned} \mathit{Slot\text{-}Ch} = {} & \mathit{Domain} \mid \mathit{Slot\text{-}value\text{-}type} \mid \mathit{Slot\text{-}inverse} \mid \\ & \mathit{Slot\text{-}cardinality} \mid \mathit{Slot\text{-}maximum\text{-}cardinality} \mid \\ & \mathit{Slot\text{-}minimum\text{-}cardinality} \mid \mathit{Slot\text{-}numeric\text{-}minimum} \mid \\ & \mathit{Slot\text{-}numeric\text{-}maximum} \mid \mathit{Slot\text{-}collection\text{-}type} \mid \\ & \mathit{Slot\text{-}values}; \end{aligned}$$

```
<!ATTLIST slot
       type ( template | own ) "own">
```

$$\mathit{SlotAtt} :: T : (\mathit{Template} \mid \text{OWN});$$

$Template = \text{token};$

```
<!ELEMENT domain (#PCDATA)>
<!ELEMENT slot_value_type (#PCDATA)>
<!ELEMENT slot_inverse (#PCDATA)>
<!ELEMENT slot_ardinality (#PCDATA)>
<!ELEMENT slot_maximum_cardinality (#PCDATA)>
<!ELEMENT slot_minimum_cardinality (#PCDATA)>
<!ELEMENT slot_numeric_minimum (#PCDATA)>
<!ELEMENT slot_numeric_maximum (#PCDATA)>
<!ELEMENT slot_collection_type (#PCDATA)>
```

$Domain = \text{char}^*;$
$Slot\text{-}value\text{-}type = \text{char}^*;$
$Slot\text{-}inverse = \text{char}^*;$
$Slot\text{-}cardinality = \text{char}^*;$
$Slot\text{-}maximum\text{-}cardinality = \text{char}^*;$
$Slot\text{-}minimum\text{-}cardinality = \text{char}^*;$
$Slot\text{-}numeric\text{-}minimum = \text{char}^*;$
$Slot\text{-}numeric\text{-}maximum = \text{char}^*;$
$Slot\text{-}collection\text{-}type = \text{char}^*;$

```
<!ELEMENT individual (name, documentation?,(type | slot-values)*>
```

$Individuals = I\text{-}Id \xrightarrow{m} Individual;$

$I\text{-}Id = \text{token};$

$$\begin{aligned} Individual :: \; & N : Name \\ & D : [Documentation] \\ & C4 : (Type \mid Slot\text{-}values)\text{-set} \\ & P : [C\text{-}Id\text{-set}] \end{aligned}$$

$Type = \text{token};$

```
<!ELEMENT slot-values
    (name, value*,
       (facet-values |
        value-type | inverse |
        cardinality | maximum-cardinality | minimum-cardinality |
        numeric-minimum | numeric-maximum | some-values |
        collection-type | documentation-in-frame)*
     )>
```

$$\textit{Slot-values} :: N : \textit{Name} \\ V : \textit{Value}\text{-set} \\ C5 : \textit{Val-Ch}\text{-set}$$

$$\textit{Val-Ch} = \textit{Facet-values} \mid \textit{Value-type} \mid \textit{Inverse} \mid \textit{Cardinality} \mid \textit{Maximum-cardinality} \mid \textit{Minimum-cardinality} \mid \textit{Numeric-minimum} \mid \textit{Numeric-maximum} \mid \textit{Some-values} \mid \textit{Collection-type} \mid \textit{Documentation-in-frame};$$

```
<!ELEMENT facet-values (name, value*)>
```

$$\textit{Facet-values} :: N : \textit{Name} \\ V : \textit{Value}\text{-set}$$

```
<!ELEMENT value-type (#PCDATA)>
<!ELEMENT inverse (#PCDATA)>
<!ELEMENT cardinality (#PCDATA)>
<!ELEMENT maximum-cardinality (#PCDATA)>
<!ELEMENT minimum-cardinality (#PCDATA)>
<!ELEMENT numeric-minimum (#PCDATA)>
<!ELEMENT numeric-maximum (#PCDATA)>
<!ELEMENT some-values (#PCDATA)>
<!ELEMENT collection-type (#PCDATA)>
<!ELEMENT documentation-in-frame (#PCDATA)>
```

$$\textit{Value} = \text{token}; \\ \textit{Value-type} = \text{char}^*; \\ \textit{Inverse} = \text{char}^*; \\ \textit{Cardinality} = \text{char}^*; \\ \textit{Maximum-cardinality} = \text{char}^*; \\ \textit{Minimum-cardinality} = \text{char}^*; \\ \textit{Numeric-minimum} = \text{char}^*; \\ \textit{Numeric-maximum} = \text{char}^*; \\ \textit{Some-values} = \text{char}^*; \\ \textit{Collection-type} = \text{char}^*; \\ \textit{Documentation-in-frame} = \text{char}^*$$

A.1 Additional Rules Governing XOL Documents

1. The identifier provided in every **name** element within all **class**, **individual** and **slot** elements must be unique within an *XOL* file. For example, the same name may not be used for two individuals, or for a slot and a class, within the same *XOL* file.

A.1.1 Function $snameCC$

Specification:

$$
\begin{aligned}
& snameCC : Ontology \rightarrow \mathbb{B} \\
& snameCC\,(ont) \triangleq \\
& \quad \forall\, c1 \in \mathsf{dom}\ ont.C \cdot \\
& \qquad (\forall\, c2 \in (\mathsf{dom}\ ont.C) \setminus \{c1\} \cdot ont.C\,(c1).N \neq ont.C\,(c2).N);
\end{aligned}
$$

Description:

The same name may not be used for two classes

Calls:

Standard VDM-SL only

A.1.2 Function $snameSS$

Specification:

$$
\begin{aligned}
& snameSS : Ontology \rightarrow \mathbb{B} \\
& snameSS\,(ont) \triangleq \\
& \quad \forall\, s1 \in \mathsf{dom}\ ont.S \cdot \\
& \qquad (\forall\, s2 \in (\mathsf{dom}\ ont.S) \setminus \{s1\} \cdot ont.S\,(s1).N \neq ont.S\,(s2).N);
\end{aligned}
$$

Description:

The same name may not be used for two slots

Calls:

Standard VDM-SL only

A.1.3 Function $snameII$

Specification:

$$
\begin{aligned}
& snameII : Ontology \rightarrow \mathbb{B} \\
& snameII\,(ont) \triangleq \\
& \quad \forall\, i1 \in \mathsf{dom}\ ont.I \cdot \\
& \qquad (\forall\, i2 \in (\mathsf{dom}\ ont.I) \setminus \{i1\} \cdot ont.I\,(i1).N \neq ont.I\,(i2).N);
\end{aligned}
$$

Description:

The same name may not be used for two individuals

Calls:

Standard VDM-SL only

A.1.4 Function $snameCS$

Specification:

$$snameCS : Ontology \to \mathbb{B}$$
$$snameCS\,(ont) \triangleq$$
$$\forall\, s \in \mathsf{rng}\; ont.S \cdot (\forall\, c1 \in \mathsf{dom}\; ont.C \cdot ont.C\,(c1).N \neq s.N);$$

Description:

The same name may not be used for class and a slot

Calls:

Standard VDM-SL only

A.1.5 Function $snameCI$

Specification:

$$snameCI : Ontology \to \mathbb{B}$$
$$snameCI\,(ont) \triangleq$$
$$\forall\, i \in \mathsf{rng}\; ont.I \cdot (\forall\, c1 \in \mathsf{dom}\; ont.C \cdot ont.C\,(c1).N \neq i.N);$$

Description:

The same name may not be used for class and an individual

Calls:

Standard VDM-SL only

A.1.6 Function $snameSI$

Specification:

$$snameSI : Ontology \to \mathbb{B}$$
$$snameSI\,(ont) \triangleq$$
$$\forall\, i \in \mathsf{rng}\; ont.I \cdot (\forall\, s1 \in \mathsf{dom}\; ont.S \cdot ont.S\,(s1).N \neq i.N);$$

Description:

The same name may not be used for a slot and an individual

Calls:

Standard VDM-SL only

---

2. Each class must be defined earlier in an *XOL* file than is its subclasses.

A.1.7 Function $subClass$

Specification:

$$
\begin{array}{l}
subClass : Ontology \rightarrow \mathbb{B} \\
subClass\,(ont) \triangleq \\
\quad (\forall\, c \in \mathsf{rng}\ ont.C \cdot \\
\qquad (\forall\, tid \in c.C2 \cdot \mathsf{is}\text{-}Class\,(ont.C\,(tid)) \;\Rightarrow \\
\qquad\qquad tid \in \mathsf{dom}\ ont.C \wedge \\
\qquad\qquad \mathsf{is}\text{-}Slot\,(ont.S\,(tid)) \;\Rightarrow \\
\qquad\qquad tid \in \mathsf{dom}\ ont.S \wedge \\
\qquad\qquad \mathsf{is}\text{-}Individual\,(ont.I\,(tid)) \;\Rightarrow \\
\qquad\qquad tid \in \mathsf{dom}\ ont.I) \wedge \\
\qquad c.P \neq \mathsf{nil} \;\Rightarrow \\
\qquad c.P \subseteq \mathsf{dom}\ ont.C) \wedge \\
\quad (\forall\, s \in \mathsf{rng}\ ont.S \cdot s.P \neq \mathsf{nil} \;\Rightarrow\; s.P \subseteq \mathsf{dom}\ ont.C) \wedge \\
\quad (\forall\, i \in \mathsf{rng}\ ont.I \cdot i.P \neq \mathsf{nil} \;\Rightarrow\; i.P \subseteq \mathsf{dom}\ ont.C);
\end{array}
$$

Description:

Each class must be defined earlier than a subclass

Calls:

Standard VDM-SL only

---

3. Each class must be defined earlier in an *XOL* file than is its instances.

   This rule is already guaranteed by the invariant *subclass*.

4. The identifier provided within the **subclass-of** and **instance-of** elements must be identical to the identifier within the **name** element of a class that is defined in that *XOL* file.

   Semantic questions of unicity of names are guaranteed when we use unique identifiers for each ”node”.

5. Only the subclass-of and instance-of elements for direct relationships to a parent class must be included in *XOL* files. Indirect relationships should not be included (e.g., if class A is a subclass of class B, which in turn is a subclass of class C, only the subclass-of link between A and B should be included in the *XOL* file). n addition, the superclass-of and type-of links that are the inverses of the subclass-of and instance-of links are optional.

A.1.8 Function $transClass$

Specification:

$$
\begin{array}{l}
transClass : Ontology \rightarrow \mathbb{B} \\
transClass\,(ont) \triangleq \\
\quad \forall\, c \in (\mathsf{rng}\ ont.C \cup \mathsf{rng}\ ont.S \cup \mathsf{rng}\ ont.I) \cdot \\
\qquad \bigcap getParentsC\,(ont.C, c) = \{\} \wedge \\
\qquad \bigcap getParentsS\,(ont.C, c) = \{\} \wedge \\
\qquad \bigcap getParentsI\,(ont.C, c) = \{\};
\end{array}
$$

Description:

The getParents function is defined for each kind of possible element in an Ontology, i.e., Class, Slot and Individual. This way, it calculates a set of set of parents of the current node, using the relative level of each parent to decide in wich set to put him. The result is a structure describing each diferent level of parents the node has.

Calls:

Standard VDM-SL only

A.1.9 Function $getParents$

Specification:

$$
\begin{array}{l}
getParentsC : Classes \times Class \to C\text{-}Id\text{-set-set} \\
getParentsC\,(cs, c) \triangleq \\
\quad \text{if } (c.P \neq \text{nil}) \\
\quad \text{then } \{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\} \\
\quad \text{else } \{\{\}\};
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

A.1.10 Function $getParents$

Specification:

$$
\begin{array}{l}
getParentsS : Classes \times Slot \to C\text{-}Id\text{-set-set} \\
getParentsS\,(cs, c) \triangleq \\
\quad \text{if } (c.P \neq \text{nil}) \\
\quad \text{then } \{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\} \\
\quad \text{else } \{\{\}\};
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

A.1.11 Function $getParentsI$

Specification:

$getParentsI : Classes \times Individual \rightarrow C\text{-}Id\text{-set-set}$
$getParentsI\,(cs, c) \triangleq$
if $(c.P \neq \text{nil})$
then $\{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\}$
else $\{\{\}\}$
end *xol-Ontology*

Description:

No description?

Calls:

Standard VDM-SL only

---

6. The identifier provided within the **name** element of a **slot-values** element must be identical to the identifier within the **name** element of a slot that is defined in that *XOL* file.

   Semantic question treated by the programmer.

7. Slots may only be used in classes and instances within their domain [expand].

   The firt part of this rule is already guaranteed by the invariant *subclass* and the second part are semantic questions treated by the programmer.

8. Values of a slot must obey the value-type definition for the slot.

   Semantic questions treated by the programmer.

9. Each class must be defined earlier in an *XOL* file than are its slots.

   This rule is already guaranteed by the invariant *subclass*.

B XOL++ DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT ontology
 (name,  ( kb-type | db-type )?, package?, version?, documentation?,
  class*, slot*, individual*)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT kb-type (#PCDATA)>
<!ELEMENT db-type (#PCDATA)>
<!ELEMENT package (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT documentation (#PCDATA)>

<!ELEMENT class
 (name, documentation?, ( subclass-of | instance-of | slot-values)*,
  ((c-id)*)?, meta-properties)>

<!ELEMENT subclass-of (#PCDATA)>
<!ELEMENT instance-of (#PCDATA)>
<!ELEMENT c-id (#PCDATA)>

<!ELEMENT meta-properties (rigidity, identity, unity, dependence)>

<!ELEMENT rigidity (rigid | non-rigid | anty-rigid)>

<!ELEMENT identity (carries-ic | notcarries-ic |
 supplies-ic | notsupplies-ic)>

<!ELEMENT unity (carries-uc | notcarries-uc | anty-unity)>

<!ELEMENT dependence (dependent | non-dependent)>

<!ELEMENT rigid (#PCDATA)>
<!ELEMENT non-rigid (#PCDATA)>
<!ELEMENT anty-rigid (#PCDATA)>
<!ELEMENT carries-ic (#PCDATA)>
<!ELEMENT notcarries-ic (#PCDATA)>
<!ELEMENT supplies-ic (#PCDATA)>
<!ELEMENT notsupplies-ic (#PCDATA)>
<!ELEMENT carries-uc (#PCDATA)>
<!ELEMENT notcarries-uc (#PCDATA)>
<!ELEMENT anty-unity (#PCDATA)>
<!ELEMENT dependent (#PCDATA)>
<!ELEMENT non-dependent (#PCDATA)>


<!ELEMENT slot
```

```
 (name, documentation?, ( domain | slot-value-type | slot-inverse |
  slot-cardinality | slot-maximum-cardinality |
  slot-minimum-cardinality |   slot-numeric-minimum |
  slot-numeric-maximum | slot-collection-type |
  slot-values )*, ((s-id)*)? )>

<!ELEMENT s-id (#PCDATA)>

<!ATTLIST slot  type ( template | own ) "own">

<!ELEMENT individual
 (name, documentation?, ( type | slot-values )*, ((i-id)*)?) >

<!ELEMENT type (#PCDATA)>
<!ELEMENT template (#PCDATA)>
<!ELEMENT i-id (#PCDATA)>

<!ELEMENT domain (#PCDATA)>
<!ELEMENT slot-value-type (#PCDATA)>
<!ELEMENT slot-inverse (#PCDATA)>
<!ELEMENT slot-cardinality (#PCDATA)>
<!ELEMENT slot-maximum-cardinality (#PCDATA)>
<!ELEMENT slot-minimum-cardinality (#PCDATA)>
<!ELEMENT slot-numeric-minimum (#PCDATA)>
<!ELEMENT slot-numeric-maximum (#PCDATA)>
<!ELEMENT slot-collection-type (#PCDATA)>


<!ELEMENT slot-values
 (name, value*, (facet-values | value-type | inverse | cardinality |
  maximum-cardinality | minimum-cardinality | numeric-minimum |
  numeric-maximum | some-values | collection-type |
  documentation-in-frame)*)>

<!ELEMENT facet-values (name, value*)>

<!ELEMENT value (#PCDATA)>
<!ELEMENT value-type (#PCDATA)>
<!ELEMENT inverse (#PCDATA)>
<!ELEMENT cardinality (#PCDATA)>
<!ELEMENT maximum-cardinality (#PCDATA)>
<!ELEMENT minimum-cardinality (#PCDATA)>
<!ELEMENT numeric-minimum (#PCDATA)>
<!ELEMENT numeric-maximum (#PCDATA)>
<!ELEMENT some-values (#PCDATA)>
<!ELEMENT collection-type (#PCDATA)>
<!ELEMENT documentation-in-frame (#PCDATA)>
```

C XOL++ Specification

C.1 Types

C.1.1 Ontology

class $Ontology$
$Ontology :: N : Name$
$C1 : [Kb\text{-}type \mid Db\text{-}type]$
$P : [Package]$
$V : [Version]$
$D : [Documentation]$
$C : Classes$
$S : Slots$
$I : Individuals$
inv $ont \triangleq$
$snameCC\,(ont) \wedge snameSS\,(ont) \wedge snameII\,(ont) \wedge$
$snameCS\,(ont) \wedge$
$snameCI\,(ont) \wedge snameSI\,(ont) \wedge$
$subClass\,(ont) \wedge$
$transClass\,(ont) \wedge antiRig\,(ont) \wedge$
$unity\,(ont) \wedge antiUnity\,(ont) \wedge ident\,(ont) \wedge$
$depend\,(ont);$

$Name = \text{char}^*;$
$Kb\text{-}type = \text{char}^*;$
$Db\text{-}type = \text{char}^*;$
$Package = \text{char}^*;$
$Version = \text{char}^*;$
$Documentation = \text{char}^*;$

C.1.2 Class

$Classes = C\text{-}Id \xrightarrow{m} Class;$

$C\text{-}Id = \mathbb{Z};$

$Class :: N : Name$
$D : [Documentation]$
$C2 : (C\text{-}Id \mid Slot\text{-}values)\text{-set}$
$P : [C\text{-}Id\text{-set}]$
$M : Meta\text{-}Properties$

$$\textit{Meta-Properties} :: R : \textit{Rigidity} \\ I : \textit{Identity} \\ U : \textit{Unity} \\ D : \textit{Dependence}$$

$$\textit{Rigidity} = \text{RIGID} \mid \text{NON\_RIGID} \mid \text{ANTI\_RIGID};$$

$$\textit{Identity} = \text{CARRIES\_IC} \mid \text{NOTCARRIES\_IC} \mid \\ \text{SUPPLIES\_IC} \mid \text{NOTSUPPLIES\_IC};$$

$$\textit{Unity} = \text{CARRIES\_UC} \mid \text{NOTCARRIES\_UC} \mid \text{ANTI\_UNITY};$$

$$\textit{Dependence} = \text{DEPENDENT} \mid \text{NON\_DEPENDENT};$$

C.1.3 Slot

$$\textit{Slots} = \textit{S-Id} \xrightarrow{m} \textit{Slot};$$

$$\textit{S-Id} = \mathbb{Z};$$

$$\textit{Slot} :: N : \textit{Name} \\ D : [\textit{Documentation}] \\ C3 : \textit{Slot-Ch}\text{-set} \\ A : \textit{SlotAtt} \\ P : [\textit{C-Id}\text{-set}]$$

$$\textit{Slot-Ch} = \textit{Domain} \mid \textit{Slot-value-type} \mid \textit{Slot-inverse} \mid \\ \textit{Slot-cardinality} \mid \textit{Slot-maximum-cardinality} \mid \\ \textit{Slot-minimum-cardinality} \mid \textit{Slot-numeric-minimum} \mid \\ \textit{Slot-numeric-maximum} \mid \textit{Slot-collection-type} \mid \\ \textit{Slot-values};$$

$$\textit{SlotAtt} :: T : (\textit{Template} \mid \text{OWN});$$

$$\textit{Template} = \text{token};$$

$Domain = \mathsf{char}^*;$
$Slot\text{-}value\text{-}type = \mathsf{char}^*;$
$Slot\text{-}inverSlot\text{-}numeric\text{-}minimumse = \mathsf{char}^*;$
$Slot\text{-}cardinality = \mathsf{char}^*;$
$Slot\text{-}maximum\text{-}cardinality = \mathsf{char}^*;$
$Slot\text{-}minimum\text{-}cardinality = \mathsf{char}^*;$
$Slot\text{-}numeric\text{-}minimum = \mathsf{char}^*;$
$Slot\text{-}numeric\text{-}maximum = \mathsf{char}^*;$
$Slot\text{-}collection\text{-}type = \mathsf{char}^*;$
$Slot\text{-}inverse = \mathsf{char}^*;$

C.1.4 Individual

$Individuals = I\text{-}Id \xrightarrow{m} Individual;$

$I\text{-}Id = \mathbb{Z};$

$$\begin{aligned} Individual :: \; & N : Name \\ & D : [Documentation] \\ & C4 : (Type \mid Slot\text{-}values)\text{-}\mathsf{set} \\ & P : [C\text{-}Id\text{-}\mathsf{set}] \end{aligned}$$

$Type = \mathsf{token};$

$$\begin{aligned} Slot\text{-}values :: \; & N : Name \\ & V : Value\text{-}\mathsf{set} \\ & C5 : Val\text{-}Ch\text{-}\mathsf{set} \end{aligned}$$

$$\begin{aligned} Val\text{-}Ch = \; & Facet\text{-}values \mid Value\text{-}type \mid Inverse \mid \\ & Cardinality \mid Maximum\text{-}cardinality \mid \\ & Minimum\text{-}cardinality \mid Numeric\text{-}minimum \mid \\ & Numeric\text{-}maximum \mid Some\text{-}values \mid \\ & Collection\text{-}type \mid Documentation\text{-}in\text{-}frame; \end{aligned}$$

$$\begin{aligned} Facet\text{-}values :: \; & N : Name \\ & V : Value\text{-}\mathsf{set} \end{aligned}$$

$Value = \text{char}^*;$
$Value\text{-}type = \text{char}^*;$
$Inverse = \text{char}^*;$
$Cardinality = \text{char}^*;$
$Maximum\text{-}cardinality = \text{char}^*;$
$Minimum\text{-}cardinality = \text{char}^*;$
$Numeric\text{-}minimum = \text{char}^*;$
$Numeric\text{-}maximum = \text{char}^*;$
$Some\text{-}values = \text{char}^*;$
$Collection\text{-}type = \text{char}^*;$
$Documentation\text{-}in\text{-}frame = \text{char}^*;$

$ClassTuple = C\text{-}Id \times Name \times [Documentation] \times (C\text{-}Id \mid Slot\text{-}values)\text{-set} \times [C\text{-}Id\text{-set}] \times Meta\text{-}Properties;$
$SlotTuple = S\text{-}Id \times Name \times [Documentation] \times Slot\text{-}Ch\text{-set} \times SlotAtt \times [C\text{-}Id\text{-set}];$
$IndividualTuple = I\text{-}Id \times Name \times [Documentation] \times (Type \mid Slot\text{-}values)\text{-set} \times [C\text{-}Id\text{-set}]$

C.2 Invariants Functions

C.2.1 Function $snameCC$

Specification:

$$snameCC : Ontology \rightarrow \mathbb{B}$$
$$snameCC\,(ont) \triangleq$$
$$\forall\, c1 \in \text{dom}\; ont.C \cdot$$
$$(\forall\, c2 \in (\text{dom}\; ont.C) \setminus \{c1\} \cdot ont.C\,(c1).N \neq ont.C\,(c2).N);$$

Description:

The same name may not be used for two classes

Calls:

Standard VDM-SL only

C.2.2 Function $snameSS$

Specification:

$$snameSS : Ontology \rightarrow \mathbb{B}$$
$$snameSS\,(ont) \triangleq$$
$$\forall\, s1 \in \text{dom}\; ont.S \cdot$$
$$(\forall\, s2 \in (\text{dom}\; ont.S) \setminus \{s1\} \cdot ont.S\,(s1).N \neq ont.S\,(s2).N);$$

Description:

The same name may not be used for two slots

Calls:

Standard VDM-SL only

C.2.3 Function $snameII$

Specification:

$$
\begin{array}{l}
snameII : Ontology \to \mathbb{B} \\
snameII\,(ont) \triangleq \\
\quad \forall\, i1 \in \mathsf{dom}\ ont.I \cdot \\
\qquad (\forall\, i2 \in (\mathsf{dom}\ ont.I) \setminus \{i1\} \cdot ont.I\,(i1).N \neq ont.I\,(i2).N);
\end{array}
$$

Description:

The same name may not be used for two individuals

Calls:

Standard VDM-SL only

C.2.4 Function $snameCS$

Specification:

$$
\begin{array}{l}
snameCS : Ontology \to \mathbb{B} \\
snameCS\,(ont) \triangleq \\
\quad \forall\, s \in \mathsf{rng}\ ont.S \cdot (\forall\, c1 \in \mathsf{dom}\ ont.C \cdot ont.C\,(c1).N \neq s.N);
\end{array}
$$

Description:

The same name may not be used for class and a slot

Calls:

Standard VDM-SL only

C.2.5 Function $snameCI$

Specification:

$$
\begin{array}{l}
snameCI : Ontology \to \mathbb{B} \\
snameCI\,(ont) \triangleq \\
\quad \forall\, i \in \mathsf{rng}\ ont.I \cdot (\forall\, c1 \in \mathsf{dom}\ ont.C \cdot ont.C\,(c1).N \neq i.N);
\end{array}
$$

Description:

The same name may not be used for class and an individual

Calls:

Standard VDM-SL only

C.2.6 Function $snameSI$

Specification:

$$
\begin{aligned}
& snameSI : Ontology \to \mathbb{B} \\
& snameSI\,(ont) \triangleq \\
& \quad \forall\, i \in \mathsf{rng}\ ont.I \cdot (\forall\, s1 \in \mathsf{dom}\ ont.S \cdot ont.S\,(s1).N \neq i.N);
\end{aligned}
$$

Description:

The same name may not be used for a slot and an individual

Calls:

Standard VDM-SL only

C.2.7 Function $subClass$

Specification:

$$
\begin{aligned}
& subClass : Ontology \to \mathbb{B} \\
& subClass\,(ont) \triangleq \\
& \quad (\forall\, c \in \mathsf{rng}\ ont.C \cdot \\
& \qquad (\forall\, tid \in c.C2 \cdot \mathsf{is\text{-}}Class\,(ont.C\,(tid)) \Rightarrow \\
& \qquad\qquad tid \in \mathsf{dom}\ ont.C \land \\
& \qquad\qquad \mathsf{is\text{-}}Slot\,(ont.S\,(tid)) \Rightarrow \\
& \qquad\qquad tid \in \mathsf{dom}\ ont.S \land \\
& \qquad\qquad \mathsf{is\text{-}}Individual\,(ont.I\,(tid)) \Rightarrow \\
& \qquad\qquad tid \in \mathsf{dom}\ ont.I) \land \\
& \qquad c.P \neq \mathsf{nil} \Rightarrow \\
& \qquad c.P \subseteq \mathsf{dom}\ ont.C) \land \\
& \quad (\forall\, s \in \mathsf{rng}\ ont.S \cdot s.P \neq \mathsf{nil} \Rightarrow s.P \subseteq \mathsf{dom}\ ont.C) \land \\
& \quad (\forall\, i \in \mathsf{rng}\ ont.I \cdot i.P \neq \mathsf{nil} \Rightarrow i.P \subseteq \mathsf{dom}\ ont.C);
\end{aligned}
$$

Description:

Each class must be defined earlier than a subclass

Calls:

Standard VDM-SL only

C.2.8 Function $transClass$

Specification:

$$
\begin{array}{l}
transClass : Ontology \to \mathbb{B} \\
transClass\,(ont) \triangleq \\
\quad \forall\, c \in (\mathsf{rng}\ ont.C \cup \mathsf{rng}\ ont.S \cup \mathsf{rng}\ ont.I) \cdot \\
\qquad \bigcap getParentsC\,(ont.C, c) = \{\} \wedge \\
\qquad \bigcap getParentsS\,(ont.C, c) = \{\} \wedge \\
\qquad \bigcap getParentsI\,(ont.C, c) = \{\};
\end{array}
$$

Description:

This invariant garantees that if class A is a subclass of class B, which in turn is a subclass of class C, only the subclass-of link between A and B should be included in the XOL file. The getParents function is defined for each kind of possible element in an Ontology, i.e., Class, Slot and Individual. This way, it calculates a set of set of parents of the current node, using the relative level of each parent to decide in wich set to put him. The result is a structure describing each diferent level of parents the node has.

Calls:

Standard VDM-SL only

C.2.9 Function $getParents$

Specification:

$$
\begin{array}{l}
getParentsC : Classes \times Class \to C\text{-}Id\text{-}\mathsf{set}\text{-}\mathsf{set} \\
getParentsC\,(cs, c) \triangleq \\
\quad \mathsf{if}\ (c.P \neq \mathsf{nil}\ ) \\
\quad \mathsf{then}\ \{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\} \\
\quad \mathsf{else}\ \{\{\}\};
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.2.10 Function $getParents$

Specification:

$$
\begin{array}{l}
getParentsS : Classes \times Slot \rightarrow C\text{-}Id\text{-set-set} \\
getParentsS\,(cs, c) \triangleq \\
\quad \text{if } (c.P \neq \text{nil}) \\
\quad \text{then } \{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\} \\
\quad \text{else } \{\{\}\};
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.2.11 Function $getParents$

Specification:

$$
\begin{array}{l}
getParentsI : Classes \times Individual \rightarrow C\text{-}Id\text{-set-set} \\
getParentsI\,(cs, c) \triangleq \\
\quad \text{if } (c.P \neq \text{nil}) \\
\quad \text{then } \{c.P\} \cup \bigcup \{getParentsC\,(cs, cs\,(c)) \mid c \in c.P\} \\
\quad \text{else } \{\{\}\};
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.2.12 Function $antiRig$

Specification:

$$
\begin{array}{l}
antiRig : Ontology \rightarrow \mathbb{B} \\
antiRig\,(ont) \triangleq \\
\quad \forall\, c \in \text{rng } ont.C \cdot \\
\qquad (c.M).R = \text{ANTI\_RIGID} \;\Rightarrow\; \forall\, cid \in c.C2 \cdot \\
\qquad\quad cid \in \text{dom } ont.C \;\Rightarrow\; (ont.C\,(cid).M).R \neq \text{RIGID};
\end{array}
$$

Description:

A `Anti-Rigid` class cannot have a `Rigig` subclass.

Calls:

Standard VDM-SL only

C.2.13 Function $unity$

Specification:

$$
\begin{array}{l}
unity : Ontology \rightarrow \mathbb{B} \\
unity\,(ont) \triangleq \\
\quad \forall\, c \in \mathsf{rng}\ ont.C \cdot \\
\qquad (c.M).U = \text{CARRIES\_UC} \;\Rightarrow\; \forall\, cid \in c.C2 \cdot \\
\qquad\quad cid \in \mathsf{dom}\ ont.C \;\Rightarrow\; (ont.C\,(cid).M).U \neq \text{NOTCARRIES\_UC};
\end{array}
$$

Description:

A `Carries-UC` class cannot have a `NotCarries-UC` subclass.

Calls:

Standard VDM-SL only

C.2.14 Function $unity$

Specification:

$$
\begin{array}{l}
antiUnity : Ontology \rightarrow \mathbb{B} \\
antiUnity\,(ont) \triangleq \\
\quad \forall\, c \in \mathsf{rng}\ ont.C \cdot \\
\qquad (c.M).U = \text{ANTI\_UNITY} \;\Rightarrow\; \forall\, cid \in c.C2 \cdot \\
\qquad\quad cid \in \mathsf{dom}\ ont.C \;\Rightarrow\; (ont.C\,(cid).M).U \neq \text{CARRIES\_UC};
\end{array}
$$

Description:

A`Anti-Unity` class cannot have a `Carries-UC` subclass.

Calls:

Standard VDM-SL only

C.2.15 Function $ident$

Specification:

$$
\begin{array}{l}
ident : Ontology \rightarrow \mathbb{B} \\
ident\,(ont) \triangleq \\
\quad \forall\, c \in \mathsf{rng}\ ont.C \cdot \\
\qquad (c.M).I = \text{CARRIES\_IC} \;\Rightarrow\; \forall\, cid \in c.C2 \cdot \\
\qquad\quad cid \in \mathsf{dom}\ ont.C \;\Rightarrow\; (ont.C\,(cid).M).I \neq \text{NOTCARRIES\_IC};
\end{array}
$$

Description:

A `Carries-IC`class cannot have a `NotCarries-IC` subclass.

Calls:

Standard VDM-SL only

C.2.16 Function *depend*

Specification:

$depend : Ontology \to \mathbb{B}$
$depend\,(ont) \triangleq$
$\quad \forall\, c \in \mathsf{rng}\; ont.C \cdot$
$\qquad (c.M).D = \text{DEPENDENT} \;\Rightarrow\; \forall\, cid \in c.C2 \cdot$
$\qquad\quad cid \in \mathsf{dom}\; ont.C \;\Rightarrow\; (ont.C\,(cid).M).D \neq \text{NON\_DEPENDENT}$
end *Ontology*

Description:

A `Dependent` class cannot have a `Non-Dependent` subclass.

Calls:

Standard VDM-SL only

C.3 Operations

class *Variables*
instance variables
$\quad ontology : Ontology\text{`}Ontology := Example\text{`}ontology;$

C.3.1 Operation *insertClass*

Specification:

$insertClass : Ontology\text{`}ClassTuple \overset{o}{\to} ()$
$insertClass\,(\mathsf{mk\text{-}}\,(cid, n, d, ch, cId, mp)) \triangleq$
$\quad ontology := \mathsf{mk\text{-}}Ontology\text{`}Ontology\,(ontology.N,$
$\qquad ontology.C1,$
$\qquad ontology.P,$
$\qquad ontology.V,$
$\qquad ontology.D,$
$\qquad ontology.C \uplus_m \{cid \mapsto$
$\qquad\; \mathsf{mk\text{-}}Ontology\text{`}Class\,(n, d, ch, cId, mp)\},$
$\qquad ontology.S,$
$\qquad ontology.I);$

Description:

No description?

Calls:

Standard VDM-SL only

C.3.2 Operation *removeClass*

Specification:

$$
\begin{array}{l}
removeClass : Ontology`C\text{-}Id \xrightarrow{o} () \\
removeClass\,(cid) \triangleq \\
\quad ontology := \mathsf{mk\text{-}}Ontology`Ontology\,(ontology.N, \\
\qquad ontology.C1, \\
\qquad ontology.P, \\
\qquad ontology.V, \\
\qquad ontology.D, \\
\qquad \{cid\} \lhd ontology.C, \\
\qquad ontology.S, \\
\qquad ontology.I);
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.3.3 Operation *insertSlot*

Specification:

$$
\begin{array}{l}
insertSlot : Ontology`SlotTuple \xrightarrow{o} () \\
insertSlot\,(\mathsf{mk\text{-}}\,(sid, n, d, ch, sa, cId)) \triangleq \\
\quad ontology := \mathsf{mk\text{-}}Ontology`Ontology\,(ontology.N, \\
\qquad ontology.C1, \\
\qquad ontology.P, \\
\qquad ontology.V, \\
\qquad ontology.D, \\
\qquad ontology.C, \\
\qquad ontology.S \uplus_{m} \{sid \mapsto \\
\qquad\quad \mathsf{mk\text{-}}Ontology`Slot\,(n, d, ch, sa, cId)\}, \\
\qquad ontology.I);
\end{array}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.3.4 Operation *removeSlot*

Specification:

$$
\begin{aligned}
&removeSlot : Ontology`S\text{-}Id \xrightarrow{o} ()\\
&removeSlot\,(sid) \triangleq\\
&\quad ontology := \mathsf{mk\text{-}}Ontology`Ontology\,(ontology.N,\\
&\qquad ontology.C1,\\
&\qquad ontology.P,\\
&\qquad ontology.V,\\
&\qquad ontology.D,\\
&\qquad ontology.C,\\
&\qquad \{sid\} \lhd ontology.S,\\
&\qquad ontology.I);
\end{aligned}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.3.5 Operation *insertIndividual*

Specification:

$$
\begin{aligned}
&insertIndividual : Ontology`IndividualTuple \xrightarrow{o} ()\\
&insertIndividual\,(\mathsf{mk\text{-}}\,(iid, n, d, ch, cId)) \triangleq\\
&\quad ontology := \mathsf{mk\text{-}}Ontology`Ontology\,(ontology.N,\\
&\qquad ontology.C1,\\
&\qquad ontology.P,\\
&\qquad ontology.V,\\
&\qquad ontology.D,\\
&\qquad ontology.C,\\
&\qquad ontology.S,\\
&\qquad ontology.I \uplus_{m} \{iid \mapsto\\
&\qquad\quad \mathsf{mk\text{-}}Ontology`Individual\,(n, d, ch, cId)\});
\end{aligned}
$$

Description:

No description?

Calls:

Standard VDM-SL only

C.3.6 Operation *removeIndividual*

Specification:

$$removeIndividual : Ontology`I\text{-}Id \xrightarrow{o} ()$$
$$removeIndividual\,(iid) \triangleq$$
$$ontology := \mathsf{mk}\text{-}Ontology`Ontology\,(ontology.N, ontology.C1, ontology.P, ontology.V, ontology.D, ontology.C, ontology.S, \{iid\} \lhd ontology.I)$$

end *Variables*

Description:

No description?

Calls:

Standard VDM-SL only

C.4 Values

```
class Example
values

  classes : Ontology`Classes = {1 |-> mk-Ontology`Class
                                  (
                                   "Person",
                                   "The class of all persons",
                                   {2, 3},
                                   nil ,
                                   mk-Ontology`Meta-Properties (RIGID,
                                                               NOTCARRIES_IC,
                                                               CARRIES_UC,
                                                               NON_DEPENDENT)),
                                2 |-> mk-Ontology`Class
                                  (
                                   "man",
                                   "The class whose sex is male",
                                   {},
                                   {1},
                                   mk-Ontology`Meta-Properties (RIGID,
                                                               SUPPLIES_IC,
                                                               CARRIES_UC,
                                                               NON_DEPENDENT)),
                                3 |-> mk-Ontology`Class
                                  (
                                   "woman",
                                   "The class whose sex is female",
                                   {},
                                   {1},
                                   mk-Ontology`Meta-Properties (RIGID,
                                                               SUPPLIES_IC,
                                                               CARRIES_UC,
                                                               NON_DEPENDENT))};
```

$$
\begin{array}{l}
slots : Ontology`Slots = \{1 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}year\text{-}of\text{-}birth\texttt{"}, \\
\quad \texttt{"}An\ integer\ that\ represents\ the\ year\ the\ person\ was\ born\texttt{"}, \\
\quad \{\texttt{"}person\texttt{"}, \texttt{"}1\texttt{"}, \texttt{"}1800\texttt{"}, \texttt{"}integer\texttt{"}\}, \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{1\}), \\
2 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}brothers\texttt{"}, \\
\quad \texttt{"}The\ brothers\ of\ a\ person\texttt{"}, \\
\quad \{\texttt{"}person\texttt{"}, \texttt{"}man\texttt{"}\}, \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{1\}), \\
3 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}citizenship\texttt{"}, \\
\quad \texttt{"}Describes\ the\ citizenship\ status\ of\ a\ person\texttt{"}, \\
\quad \{\texttt{"}person\texttt{"}, \texttt{"}set\text{-}of\ citizen\ resident\text{-}alien\ permanent\text{-}resident\texttt{"} \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{1\}), \\
4 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}life\text{-}history\texttt{"}, \\
\quad \texttt{"}A\ written\ history\ of\ a\ person's\ life\texttt{"}, \\
\quad \{\texttt{"}string\texttt{"}\}, \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{1\}), \\
5 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}father\text{-}of\texttt{"}, \\
\quad \texttt{"}father\text{-}of(X, Y)\ holds\ when\ X\ is\ the\ father\ of\ Y\texttt{"}, \\
\quad \{\texttt{"}man\texttt{"}, \texttt{"}person\texttt{"}, \texttt{"}father\texttt{"}\}, \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{2\}), \\
6 \mapsto \textsf{mk-}Ontology`Slot \\
\quad (\\
\quad \texttt{"}has\text{-}father\texttt{"}, \\
\quad \texttt{"}has\text{-}father(X, Y)\ holds\ when\ the\ father\ of\ X\ is\ Y\texttt{"}, \\
\quad \{\texttt{"}person\texttt{"}, \texttt{"}man\texttt{"}, \texttt{"}father\text{-}of\texttt{"}\}, \\
\quad \textsf{mk-}Ontology`SlotAtt\ (\textsc{own}), \\
\quad \{1\})\};
\end{array}
$$

```
individuals : Ontology`Individuals = {1 ↦ mk-Ontology`Individual
                                          (
                                           "John",
                                           nil ,
                                           {mk-Ontology`Slot-values
                                                 (
                                                  "year-of-birth",
                                                  {"1987"},
                                                  {}),
                                            mk-Ontology`Slot-values
                                                 (
                                                  "citizenship",
                                                  {"permanent-resident"},
                                                  {}),
                                            mk-Ontology`Slot-values
                                                 (
                                                  "has-father",
                                                  {"Carl"},
                                                  {})},
                                           {2}),
                                      2 ↦ mk-Ontology`Individual
                                          (
                                           "Carl",
                                           nil ,
                                           {mk-Ontology`Slot-values
                                                 (
                                                  "year-of-birth",
                                                  {"1961"},
                                                  {}),
                                            mk-Ontology`Slot-values
                                                 (
                                                  "father-of",
                                                  {"John"},
                                                  {}),
                                            mk-Ontology`Slot-values
                                                 (
                                                  "life-history",
                                                  {"Carl worked hard all his life"},
                                                  {})},
                                           {2})};
```

$ontology : Ontology`Ontology = \mathsf{mk\text{-}}Ontology`Ontology$
$($
$"Genealogy",$
$"ocelot",$
$"user",$
$"",$
$"",$
$classes,$
$slots,$
$individuals)$

end *Example*

D XOL++ Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v5 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
       <xs:complexType name="MclassType">
              <xs:sequence>
                     <xs:element ref="c-id"/>
                     <xs:element name="class" type="classType"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="MindType">
              <xs:sequence>
                     <xs:element ref="i-id"/>
                     <xs:element name="individual"
                                 type="individualType"/>
              </xs:sequence>
       </xs:complexType>
       <xs:complexType name="MslotType">
              <xs:sequence>
                     <xs:element ref="c-id"/>
                     <xs:element name="slot" type="slotType"/>
              </xs:sequence>
       </xs:complexType>
       <xs:element name="anty-rigid">
              <xs:complexType/>
       </xs:element>
       <xs:element name="anty-unity">
              <xs:complexType/>
       </xs:element>
       <xs:element name="c-id" type="xs:string"/>
       <xs:element name="cardinality" type="xs:string"/>
       <xs:element name="carries-ic">
              <xs:complexType/>
       </xs:element>
       <xs:element name="carries-uc">
              <xs:complexType/>
       </xs:element>
       <xs:complexType name="classType">
              <xs:sequence>
                     <xs:element ref="name"/>
                     <xs:element ref="documentation"
                                 minOccurs="0"/>
                     <xs:choice minOccurs="0"
                                maxOccurs="unbounded">
                            <xs:element ref="c-id"/>
                            <xs:element name="slot-values"
                                        type="slot-valuesType"/>
```

```
                                </xs:choice>
                                <xs:choice minOccurs="0" maxOccurs="unbounded">
                                        <xs:element ref="c-id"/>
                                </xs:choice>
                                <xs:element name="meta-properties"
                                            type="meta-propertiesType"/>
                        </xs:sequence>
                </xs:complexType>
                <xs:complexType name="classesType">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                                <xs:element name="Mclass" type="MclassType"/>
                        </xs:sequence>
                </xs:complexType>
                <xs:element name="collection-type" type="xs:string"/>
                <xs:element name="db-type" type="xs:string"/>
                <xs:complexType name="dependenceType">
                        <xs:choice>
                                <xs:element ref="dependent"/>
                                <xs:element ref="non-dependent"/>
                        </xs:choice>
                </xs:complexType>
                <xs:element name="dependent">
                        <xs:complexType/>
                </xs:element>
                <xs:element name="documentation" type="xs:string"/>
                <xs:element name="documentation-in-frame" type="xs:string"/>
                <xs:element name="domain" type="xs:string"/>
                <xs:complexType name="facet-valuesType">
                        <xs:sequence>
                                <xs:element ref="name"/>
                                <xs:element ref="value" minOccurs="0"
                                            maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
                <xs:element name="i-id" type="xs:string"/>
                <xs:complexType name="identityType">
                        <xs:choice>
                                <xs:element ref="carries-ic"/>
                                <xs:element ref="notcarries-ic"/>
                                <xs:element ref="supplies-ic"/>
                                <xs:element ref="notsupplies-ic"/>
                        </xs:choice>
                </xs:complexType>
                <xs:complexType name="individualType">
                        <xs:sequence>
                                <xs:element ref="name"/>
                                <xs:element ref="documentation" minOccurs="0"/>
                                <xs:choice minOccurs="0" maxOccurs="unbounded">
                                        <xs:element ref="type"/>
```

```
                                <xs:element name="slot-values"
                                            type="slot-valuesType"/>
                        </xs:choice>
                        <xs:choice minOccurs="0" maxOccurs="unbounded">
                                <xs:element ref="i-id"/>
                        </xs:choice>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="individualsType">
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="Mind" type="MindType"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="instance-of" type="xs:string"/>
        <xs:element name="inverse" type="xs:string"/>
        <xs:element name="kb-type" type="xs:string"/>
        <xs:element name="maximum-cardinality" type="xs:string"/>
        <xs:complexType name="meta-propertiesType">
                <xs:sequence>
                        <xs:element name="rigidity" type="rigidityType"/>
                        <xs:element name="identity" type="identityType"/>
                        <xs:element name="unity" type="unityType"/>
                        <xs:element name="dependence"
                                    type="dependenceType"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="minimum-cardinality" type="xs:string"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="non-dependent">
                <xs:complexType/>
        </xs:element>
        <xs:element name="non-rigid">
                <xs:complexType/>
        </xs:element>
        <xs:element name="notcarries-ic">
                <xs:complexType/>
        </xs:element>
        <xs:element name="notcarries-uc">
                <xs:complexType/>
        </xs:element>
        <xs:element name="notsupplies-ic">
                <xs:complexType/>
        </xs:element>
        <xs:element name="numeric-maximum" type="xs:string"/>
        <xs:element name="numeric-minimum" type="xs:string"/>
        <xs:element name="ontology">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="name"/>
```

```
                                                <xs:choice minOccurs="0">
                                                        <xs:element ref="kb-type"/>
                                                        <xs:element ref="db-type"/>
                                                </xs:choice>
                                                <xs:element ref="package" minOccurs="0"/>
                                                <xs:element ref="version" minOccurs="0"/>
                                                <xs:element ref="documentation"
                                                            minOccurs="0"/>
                                                <xs:element name="classes"
                                                            type="classesType"/>
                                                <xs:element name="slots"
                                                            type="slotsType"/>
                                                <xs:element name="individuals"
                                                            type="individualsType"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="package" type="xs:string"/>
                        <xs:element name="rigid">
<xs:complexType/>
                        </xs:element>
                        <xs:complexType name="rigidityType">
                                <xs:choice>
                                        <xs:element ref="rigid"/>
                                        <xs:element ref="non-rigid"/>
                                        <xs:element ref="anty-rigid"/>
                                </xs:choice>
                        </xs:complexType>
                        <xs:element name="s-id" type="xs:string"/>
                        <xs:complexType name="slotType">
                                <xs:sequence>
                                        <xs:element ref="name"/>
                                        <xs:element ref="documentation" minOccurs="0"/>
                                        <xs:choice minOccurs="0" maxOccurs="unbounded">
                                                <xs:element ref="domain"/>
                                                <xs:element ref="slot-value-type"/>
                                                <xs:element ref="slot-inverse"/>
                                                <xs:element ref="slot-cardinality"/>
                                                <xs:element ref="slot-maximum-cardinality"/
                                                <xs:element ref="slot-minimum-cardinality"/
                                                <xs:element ref="slot-numeric-minimum"/>
                                                <xs:element ref="slot-numeric-maximum"/>
                                                <xs:element ref="slot-collection-type"/>
                                                <xs:element name="slot-values"
                                                            type="slot-valuesType"/>
                                        </xs:choice>
                                        <xs:choice minOccurs="0" maxOccurs="unbounded">
                                                <xs:element ref="s-id"/>
                                        </xs:choice>
```

```
        </xs:sequence>
        <xs:attribute name="type" default="own">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="template"/>
                    <xs:enumeration value="own"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
</xs:complexType>
<xs:element name="slot-cardinality" type="xs:string"/>
<xs:element name="slot-collection-type" type="xs:string"/>
<xs:element name="slot-inverse" type="xs:string"/>
<xs:element name="slot-maximum-cardinality" type="xs:string"/>
<xs:element name="slot-minimum-cardinality" type="xs:string"/>
<xs:element name="slot-numeric-maximum" type="xs:string"/>
<xs:element name="slot-numeric-minimum" type="xs:string"/>
<xs:element name="slot-value-type" type="xs:string"/>
<xs:complexType name="slot-valuesType">
        <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="value" minOccurs="0"
                        maxOccurs="unbounded"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="facet-values"
                            type="facet-valuesType"/>
                <xs:element ref="value-type"/>
                <xs:element ref="inverse"/>
                <xs:element ref="cardinality"/>
                <xs:element ref="maximum-cardinality"/>
                <xs:element ref="minimum-cardinality"/>
                <xs:element ref="numeric-minimum"/>
                <xs:element ref="numeric-maximum"/>
                <xs:element ref="some-values"/>
                <xs:element ref="collection-type"/>
                <xs:element ref="documentation-in-frame"/>
            </xs:choice>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="slotsType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Mslot" type="MslotType"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="some-values" type="xs:string"/>
<xs:element name="subclass-of" type="xs:string"/>
<xs:element name="supplies-ic">
        <xs:complexType/>
</xs:element>
```

```
        <xs:element name="template" type="xs:string"/>
        <xs:element name="type" type="xs:string"/>
        <xs:complexType name="unityType">
                <xs:choice>
                        <xs:element ref="carries-uc"/>
                        <xs:element ref="notcarries-uc"/>
                        <xs:element ref="anty-unity"/>
                </xs:choice>
        </xs:complexType>
        <xs:element name="value" type="xs:string"/>
        <xs:element name="value-type" type="xs:string"/>
        <xs:element name="version" type="xs:string"/>
</xs:schema>
```

E XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module SYSTEM "xolpp.dtd">

<ontology xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and Settings\Marlene\
My Documents\Marlene\5Ano\Lab.MFP\xolpp.xsd">
   <name>genealogy</name>
   <kb-type>ocelot-kb</kb-type>
   <package>user</package>
   <class>
      <name>person</name>
      <documentation>The class of all persons</documentation>
      <meta-properties>
         <rigidity>rigid</ridity>
         <identity>notcarries-ic</identity>
         <unity>carries-uc</unity>
         <dependence>non-dependent</dependence>
      </meta-properties>
   </class>
   <class>
      <name>man</name>
      <documentation>
      The class of all persons whose sex is male.
      </documentation>
      <c-id>person</c-id>
      <meta-properties>
         <rigidity>rigid</ridity>
         <identity>supplies-ic</identity>
         <unity>carries-uc</unity>
         <dependence>non-dependent</dependence>
      </meta-properties>
   </class>
   <class>
      <name>woman</name>
      <documentation>
      The class of all persons whose sex is female.
      </documentation>
      <c-id>person</c-id>
      <meta-properties>
         <rigidity>rigid</ridity>
         <identity>supplies-ic</identity>
         <unity>carries-uc</unity>
         <dependence>non-dependent</dependence>
      </meta-properties>
   </class>
   <slot>
```

```
    <name>year-of-birth</name>
    <documentation>
    An integer that represents the year the person was born.
    </documentation>
    <domain>person</domain>
    <slot-cardinality>1</slot-cardinality>
    <slot-numeric-minimum>1800</slot-numeric-minimum>
    <slot-value-type>integer</slot-value-type>
</slot>
<slot>
    <name>brothers</name>
    <documentation>The brothers of a person.</documentation>
    <domain>person</domain>
    <slot-value-type>man</slot-value-type>
</slot>
<slot>
    <name>citizenship</name>
    <documentation>
    Describes the citizenship status of a person.
    </documentation>
    <domain>person</domain>
    <slot-value-type>(
    set-of citizen resident-alien permanent-resident)
    </slot-value-type>
</slot>
<slot>
    <name>life-history</name>
    <documentation>
    A written history of the person’s life.
    </documentation>
    <slot-value-type>string</slot-value-type>
</slot>
<slot>
    <name>father-of</name>
    <documentation>
    father-of(X,Y) holds when X is the father of Y.
    </documentation>
    <domain>man</domain>
    <slot-value-type>person</slot-value-type>
    <slot-inverse>father</slot-inverse>
</slot>
<slot>
    <name>has-father</name>
    <documentation>
    has-father(X,Y) holds when the father of X is Y.
    </documentation>
    <domain>person</domain>
    <slot-value-type>man</slot-value-type>
    <slot-inverse>father-of</slot-inverse>
```

```
    </slot>
    <individual>
      <name>John</name>
      <slot-values>
        <name>year-of-birth</name>
        <value>1987</value>
      </slot-values>
      <slot-values>
        <name>citizenship</name>
        <value>permanent-resident</value>
      </slot-values>
      <slot-values>
        <name>has-father</name>
        <value>Carl</value>
      </slot-values>
    </individual>
    <individual>
      <name>Carl</name>
      <slot-values>
        <name>year-of-birth</name>
        <value>1961</value>
      </slot-values>
      <slot-values>
        <name>father-of</name>
        <value>John</value>
      </slot-values>
      <slot-values>
        <name>life-history</name>
        <value>Carl worked hard all his life.</value>
      </slot-values>
    </individual>
  </module>
```

References

[1] Welty C. and Guarino N. Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering*, 2001.

[2] Dieter Fensel. Relating ontology languages and web standards. pages 5–7, April 2000.

[3] Guarino N. Gangemi A. and Masolo C. Restructuring wordnet's top-level: The ontoclean approach oltramari. *Proceedings of LREC2002 (OntoLex workshop)*, 2002.

[4] Guarino N. Gangemi A., Oltramari A. Masolo C., and Schneider L. Sweetening ontologies with dolce. *Proceedings of EKAW 2002*, 2002.

[5] Pisanelli D. M. Gangemi A. and Steve G. A formal ontology framework to represent norm dynamics. *Second International Workshop on Legal Ontologies*, 2000.

[6] Masolo C. Guarino N. and Oltramari A. Understanding top-level ontological distinctions gangemi. *Proc. of IJCAI 2001 workshop on Ontologies and Information Sharing*, 2001.

[7] Welty C. Guarino N., C. A. Bean R. Green, and S. Hyon Myaeng(eds.). Identity and subsumption. *The Semantics of Relationships: An Interdisciplinary Perspective*, pages 111–126, 2001.

[8] Frank Mittelbach Michel Goossens and Alexander Samarin. *The Latex Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts U.S.A., April 1994.

[9] Guarino N. and Welty C. A formal ontology of properties. *R. Dieng and O. Corby (eds.), Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference, EKAW2000*, pages 97–112, 2000.

[10] Guarino N. and Welty C. Evaluating ontological decisions with ontoclean. *Communications of the ACM, 45(2)*, pages 61–65, 2002.

[11] Vinay K. Chaudhri Peter D. Karp and Jerome Thomere. Xml-based ontology exchange language. July 1999.

[12] Erik T. Ray. *Learning XML: Guide to Creating Self-Describing Data*. O'Reilly, U.S.A., Canada and U.K., January 2001.

[13] Heather Williamson. *XML: The Complete Reference*. Corel VENTURA, Berkkeley California USA, 2001.

Index