An Automated Transformation from OntoUML to OWL and SWRL

Pedro Paulo F. Barcelos[1], Victor Amorim dos Santos[2], Freddy Brasileiro Silva[2], Maxwell E. Monteiro[3], Anilton Salles Garcia[1]

[1]Electrical Engineering and [2]Computer Science Departments
Federal University of Espírito Santo - UFES
Vitória – ES – Brazil

[3]Federal Institute of Espírito Santo - IFES
Serra – ES – Brazil

{pedropaulofb, victor.amsantos, freddybrasileiro}@gmail.com,
maxmonte@ifes.edu.br, anilton@inf.ufes.br

Abstract. *OntoUML and OWL are ontology languages appropriated to different knowledge representation levels. In order to have better knowledge representation and reasoning capabilities in OWL ontologies, an Ontology Engineering should be used – which corresponds to the transformation of a conceptual model ontology language, such as OntoUML, to a computational ontology language, such as OWL. This paper aims to bridge the expressivity gap between these languages through a Model Driven Architecture automated transformation from OntoUML to OWL with SWRL rules that contributes to (i) make easier the OWL creation from OntoUML, (ii) eliminate the human errors in this process, (iii) improve the resultant OWL ontology semantics.*

1. Introduction

In order to have better knowledge representation and reasoning capabilities in computational ontologies, like the ones represented with the Web Ontology Language (OWL), an Ontology Engineering with well-defined phases is defended in [Guizzardi 2007]. In a conceptual modeling phase, highly-expressive languages should be used to create strongly axiomatized ontologies that approximate as well as possible to the ideal ontology of the domain. The focus of these languages is on representation adequacy, since the resulting specifications are intended to be used by humans in tasks such as communication, domain analysis and problem-solving [Guizzardi 2007]. Guizzardi proposed in [Guizzardi 2005] an ontologically well-founded profile of the Unified Modeling Language (UML), later named OntoUML, to be a language used in this step. OntoUML provides stereotypes based on the Unified Foundational Ontology (UFO) to capture domain knowledge and has been successfully applied in different domains like electrophysiology [Gonçalves et al. 2007], telecommunications [Barcelos et al. 2011] and oil and gas [Guizzardi et al. 2010].

Once users have already agreed on a common conceptualization, versions of a reference ontology can be created as the objective of the Ontology Engineering (its last phase). These versions have been named in the literature lightweight ontologies. Contrary to reference ontologies, lightweight ontologies are not focused on

representation adequacy but are designed with the focus on guaranteeing desirable computational properties [Guizzardi 2007]. An Example of a language suitable for lightweight ontologies is the Web Ontology Language (OWL). OWL is the standard language for knowledge representation and reasoning in the semantic web and in computational applications. The addition of rules written in Semantic Web Rule Language (SWRL), a Horn-like rule language, in OWL ontology improves its representation expressivity.

In order to achieve this objective, an intermediate phase is necessary in the Ontology Engineering: a phase to bridge the gap between the conceptual modeling of references ontologies and the coding of these ontologies in terms of specific lightweight ontology languages. Issues that should be addressed in such a phase are, for instance, determining how to deal with the difference in expressivity of the languages that should be used in each of these phases [Guizzardi 2007]. This paper aims to present an automated transformation from an OntoUML model to OWL ontology with SWRL rules, here named OntoUML2OWL+SWRL, which is inserted into this Ontology Engineering phase.

The OntoUML2OWL+SWRL is a Model Driven Architecture (MDA) transformation that contributes to the creation of OWL files with improved semantics to be used for knowledge representation and reasoning on computational applications. Two different OntoUML to OWL transformations already exists; however, OntoUML2OWL+SWRL differ from them in scope and complexity.

This paper is structured as follows: Section 2 presents the OntoUML2OWL+SWRL, including all conceptual considerations and limitations, and all the implementation technologies used. As related works, Section 3 presents the other OntoUML to OWL transformations and their relations to our transformation. Section 4 presents some conclusions as well as future works. Background information about OntoUML and OWL concepts is provided during the paper's sections.

2. The OntoUML2OWL+SWRL Transformation

The OntoUML2OWL+SWRL transformation was created as a Model Driven Architecture (MDA) transformation [Miller and Mukerji 2003]. This transformation is done in the M2 level (the metamodel level), which makes it reusable, as each specific transformation in the M1 level (the domain model level) is an instance of the generic M2 transformations. The conceptual ontology model can be seen as a Computational Independent Model (CIM), while the OWL with SWRL rules model can be seen as a Platform Independent Model (PIM). Further transformations can be created from the PIM (the OWL) to code - a possible Platform Specific Model (PSM). OntoUML metamodel is presented in [Guizzardi 2005], and a MOF-Based OWL metamodel can be found in http://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel.

OntoUML2OWL+SWRL accomplish the following objectives: (i) make easier the OWL files creation from OntoUML models, (ii) eliminate the human errors in this process, and (iii) improve the resultant OWL ontology semantics.

The conceptual transformation' considerations are presented in section 2.1, while the implementation tools and languages are presented in section 2.2.

2.1. The Conceptual Transformation's Design

An intrinsic characteristic of transformation from high expressive modeling languages to computational ones (that must be decidable, tractable, etc.) is the loss of expressivity. These losses are presented as limitations during this section. We can cite, as a first example, the incapacity of this transformation to represent OntoUML's *existential dependencies* (specific instance dependence). Although OWL can represent existential dependencies, in order to allow this representation, the classes' instances must be known. As no instances are represented in OntoUML models, the transformation cannot create the resulting OWL with the existential restrictions.

The design considerations about OntoUML2OWL+SWRL transformation are described in this section. Our intention here is to hide as much as possible the resulting code and present just the ideas.

Classes: We have taken as a development premise the separation of the models' concepts with the metamodel's ones for class transformation. That is, in OntoUML2OWL+SWRL the generated OWL file contains only domain classes, for example, applying the transformation to a Genealogy OntoUML model, the resulting OWL will have just classes with Genealogy concepts, like Mother, Father and Offspring. It will not have OntoUML metamodel's concepts like Kind, Role, etc. This decision simplifies the generated OWL and makes it simpler to the users (humans or machines).

In classes' transformation, the OntoUML classes are directly translated to OWL classes. Even though the simplicity of this transformation, OntoUML's metamodel restrictions are considered in this step. Disjoint concepts and *Phases-partitions* (a special kind of generalization sets), explained hereafter, are examples of these considerations.

Disjoint Concepts: One of UFO's meta-properties is the *identity principle*, which is related to the nature of an object. For example, a Student is a Person, as they have the same identity principle, but they can never be a Horse, as these entities have different identity principle. The entities that provide identity principles are named Substance Sortals (stereotyped in OntoUML as Kinds, Quantities or Collectives). Mixins (Categories, Role Mixins or Mixins, in OntoUML) are the entities that aggregate objects of different identity principles. An example of Mixin is the concept "Animal", as it aggregate instances of the classes Person and Horse. In contrast with Sortals and Mixins, Moments (Modes and Relators) are entities that inhere in, and, therefore, are existentially dependent of, another entity. These entities' restrictions are considered in the OntoUML's metamodel, as can be seen in Figure 1 below.

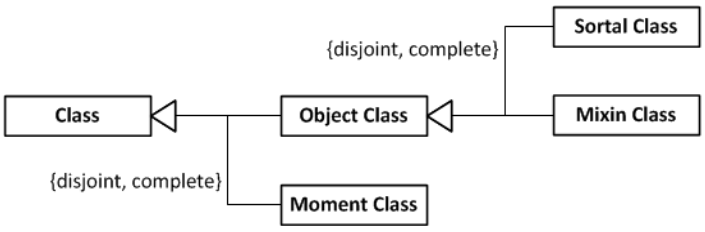

Figure 1 - Fragment of the OntoUML's metamodel

The disjoint entities are implemented in OntoUML2OWL+SWRL by the following considerations (top-level entities are entities that are not generalized by others): (a) all Substance Sortals are disjoint from each other; (b) all top-level Moments are disjoint from each other; (c) top-level Moments are disjoint from Substance Sortals and from top-level Mixin Class types.

Generalization Sets: OntoUML have two generalization sets' meta-properties: *isCovering* and *isDisjoint*, both of Boolean type. These meta-properties were considered in this transformation as follows:

- *isCovering* = *true*: the generalized class is equivalent to all complete set.
- *isDisjoint* = *true*: the generalizing classes are marked disjoint from each other.

Figure 2 presents as an example: (a) an OntoUML generalization set, (b) the resultant OWL class taxonomy, (c) the OWL Class' Person definition, and (d) the OWL Class Man's definition.

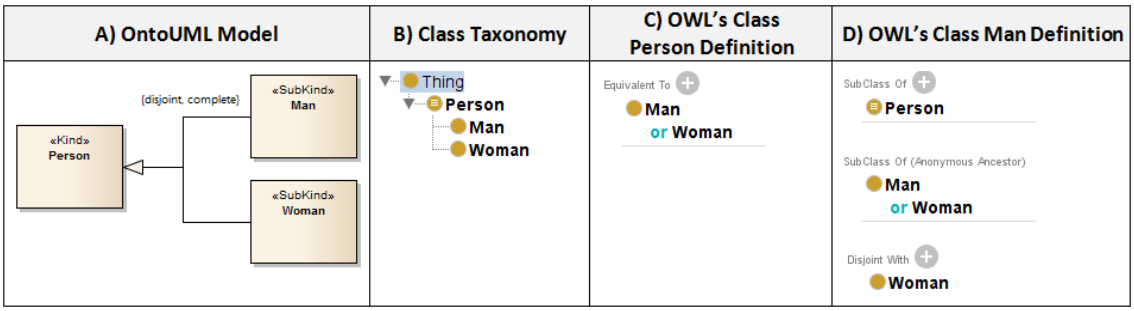

Figure 2 – Transformation of Generalization Sets

In OntoUML, *Phases-partitions* are a special type of generalization sets composed of classes stereotyped as Phases. As a particularity, they have always the true value for *isDisjoint* and *isCovering*. This particularity is considered in OntoUML2OWL+SWRL transformation.

Associations: OWL distinguishes between two main categories of associations, called *properties*: *Object properties*, that link individuals to individuals, and *DataType properties*, that link individuals to data values [Hitzler et al. 2012]. In OntoUML2OWL+SWRL, OntoUML associations are mapped to Object properties (here discussed), while DataTypes are mapped to Data properties (discussed later in this section).

Differently from OntoUML, which do not have directed associations, OWL properties are directed binary relations. This implies the necessity to create two object properties for each OntoUML association: a direct one and its inverse. As a design choice, we have named the inverse relation with the same direct relation's name prefixed with "INV.". This decision was taken because the generation of improved inverse names (for example: "drives" and "is driven by") would require language processing and it would be different in every natural language (English, French, etc.).

OntoUML associations always have a source class and a target class. Source and Target classes are considered in the transformation in order to create, respectively, the domain and range of an OWL object property. The nomenclature of generated OWL

object property is also related to these classes, as the reading direction is not a feature of OntoUML metamodel, i.e., it is just a visual resource and cannot be read from the OntoUML model to the OWL ontology. In order to produce the desired OWL object property name, the name of the OntoUML model must be given from the source class to the target one. Figure 3 illustrate the results of correct and incorrect associations.

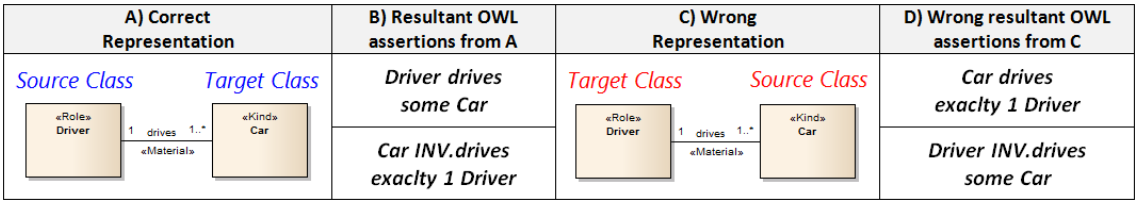

Figure 3 - Association representation

When no name is assigned to an association, the association's OntoUML stereotype is used to create its name using the following nomenclature: "AssociationStereotype.SourceClassName.TargetClassName". An example of a relation named this way can be found in the SWRL rule found in Figure 4.

Every object property is asserted as Equivalent Class of the class that it is related, except in the case when the cardinality's lower bound is zero (explained in *Cardinalities*). Disjointness of object properties is also considered as relations with different stereotypes are set as disjoint from each other (associations with the same stereotype are not set as disjoint from each other, as one can be a specialization of other). OntoUML's Material and Part-whole relations are separately explained as their transformations have particularities.

Material Relations: In OntoUML, Material relations are the ones that depend on a Relator to exists, i.e., the Material relations are derived relations that need a truth maker to exist. Figure 4 (A) presents a Material relation ("drives") that is derived from the existence of the Relator License.

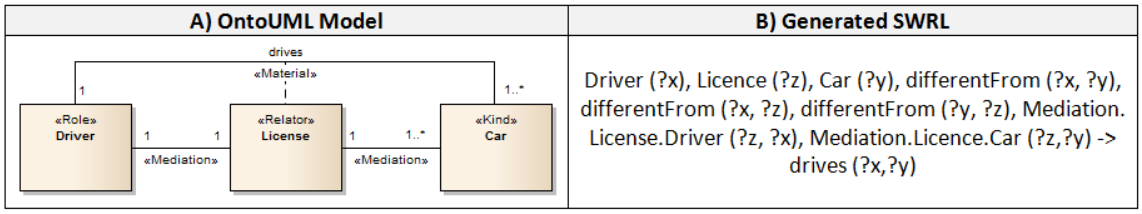

Figure 4 - SWRL resultant from Material relations

To each Material relation that exists in an OntoUML model a SWRL rule is created. This rule aims to represent the Material relation's derivation from the Relator.

Every SWRL rule created in this transformation is in accordance with Description Logic (DL) safe-rules [Motik et al. 2005], guaranteeing reasoning decidability.

Part-whole Relations: Differently to other associations, Part-whole relations are transformed to OWL sub-object properties of an object property with the name of its stereotype. This is done in order to better represent its meta-properties (called characteristics in OWL).

According to UFO, part-whole relations (stereotyped as *componentOf*, *memberOf*, *subCollectionOf* and *subQuantityOf* in OntoUML) are always irreflexive and asymmetric – a characteristic that is considered in OntoUML2OWL+SWRL.

Part-whole relations' different types have different transitivity relations, as can be seen in [Guizzardi 2005]. *subCollectionOf* and *subQuantityOf* are transitive; *memberOf* is intransitivity (it is never transitive); *componentOf* is non-transitivity, i.e., there are cases when it is transitive and other cases when it is not. Figure 5 represent the transitivity cases considered in OntoUML2OWL+SWRL (empty stereotypes are left to indicate that the pattern can occur with the following functional complex stereotypes: Kind, Subkind, Role or Phase).

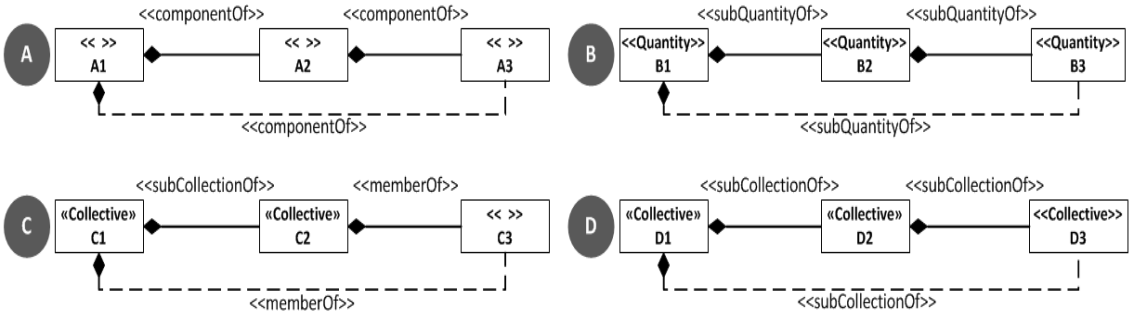

Figure 5 - Transitivity cases considered in OntoUML2OWL+SWRL

Four different generic SWRL rules can be created to represent the transitivity cases from Figure 5. These rules are added to the resultant OWL ontology when its specific case occurs. For example, every time the transitivity case (A) from Figure 5 can occur (the sum of *componentOf* is greater than 1), the following SWRL rule is created: *componentOf (?x, ?y), componentOf (?y, ?z), differentFrom (?x, ?y), differentFrom (?x, ?z), differentFrom (?y, ?z) -> componentOf (?x, ?z)*.

It is important to note that SWRL rules acts over instances, while the object properties' characteristics are defined in a higher level in OWL. If we just mark, for example, *subCollectionOf* as irreflexive, asymmetric and transitive, this will result in an error. As in the SWRL rules we are stating that the transitivity occurs only in different elements (by using the *differentFrom* operator), this error does not occurs.

An important limitation on OntoUML part-whole relations representation is about its metaproperties *isEssential* and *isInseparable*, which cannot be represented in OWL as they represent the existential dependence between parts and wholes.

DataTypes: Direct and structured DataTypes, with and without asserted cardinality, are treated in our transformation, as presented in Figure 6. These DataTypes are mapped to OWL's DataType properties.

Figure 6 - Example of considered different representations of DataTypes

The transformation supports the following OWL DataTypes: unsigned int, unsigned byte, double, String, normalized string, Boolean, hex binary, Integer (int), short, byte, unsigned long. If the provided DataType is not one of these, the transformation creates it as a Literal. Hidden cardinality is mapped to "exaclty one" concept in OWL. Attributes from the same class are set as disjoint from each other.

Applying the OntoUML2OWL+SWRL transformation to the model presented in Figure 6 we have the following object properties presented in Figure 7.

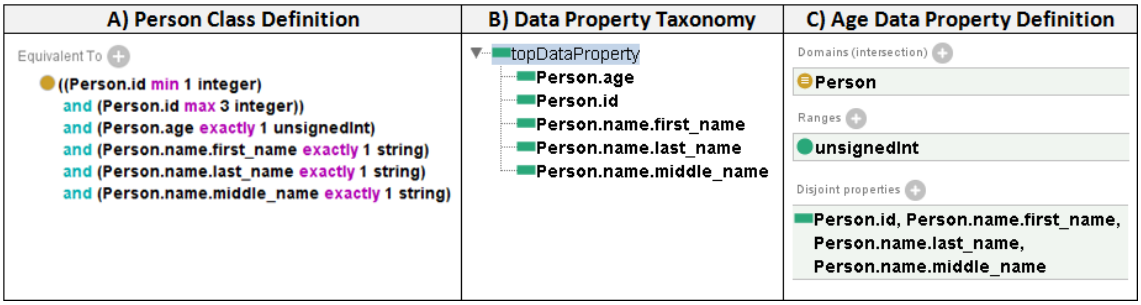

Figure 7 – OntoUML's DataTypes transformation to OWL Data Properties

DataTypes are created with the following nomenclature: "*Class.AttributeName*". In case of a structured DataType, it is created with the following nomenclature "*Class.AtributeName.StructuredDatatypeAtributeName*".

Cardinalities: Different cardinalities imply different transformations, as can be seen in Figure 8. This holds for object properties as well as to DataType properties.

OntoUML Cardinality	OWL Class Assertion	OWL Cardinality Restriction
0..*	Not treated	
0..N (0 < N < *)	Subclass Of	Max N
N (N ≠ 0)	Equivalent Class	Exactly N
1..*	Equivalent Class	Some
N..M (1 ≤ N < M, M ≠ *)	Equivalent Class	Min N and Max M

Figure 8 – Cardinality transformation

As can be seen in Figure 8, there's a transformation limitation to represent cardinalities with lower bound equal to zero, since the assertion "*has min 0*" would provoke an inconsistency. This happens because in OWL all elements "*have min 0*" properties with any other element, hence, OWL assumes that any instance of a class may have zero or more values for a particular property since a restriction was not added [Patel-Schneider et al. 2004].

In fact, properties (associations and attributes) with minimum cardinality 0 (optional properties) are not desirable in OntoUML models as they usually hide an entity's role. For example, an association "Person drives 0..* Car" hides the Person's role Driver. As stated in [Guizzardi 2005], the representation of optional cardinality constraints leads to unsound models with undesirable consequences in terms of clarity.

2.2. Transformation Implementation Technologies

The Ontology Lightweight Editor (OLED), currently in its version 0.8, is more than just an OntoUML editor - it is full framework for development of OntoUML ontologies. It provides: (a) a model editor , (b) a syntactical validation, (c) an OntoUML to OWL transformation, (d) a validation environment, which provides semantic validation realized as anti-pattern identification and treatment, and as a visual simulation through an Alloy transformation [Sales et al. 2012]. OLED is a free tool available for download at: https://code.google.com/p/ontouml-lightweight-editor/.

We have taken as a requisite to the development of the OntoUML2OWL+SWRL that the generated OWL file must open in Protégé 4.3. This decision was taken due to the fact that the Protégé is the most used tool for creation of OWL ontologies - it can be helpful to developers to view the OWL resultant from the transformation.

We have used as implementation language Java and, in order to do the translation, we have used The OWL API (http://owlapi.sourceforge.net/), an open source API that allows the developer to easily create OWL files.

As a usability issue, it is important to mention that, for OntoUML2OWL+SWRL, it is not obligatory that the OntoUML models to be created directly on OLED. The models can be created on professional tools like Sparx Systems Enterprise Architect or at Astah and exported as an XMI file and them imported in OLED.

OntoUML2OWL+SWRL code is open and can be found inside OLED's project.

3. Other OntoUML to OWL Transformations

The first identified initiative to create an OWL ontology with SWRL rules codification from an OntoUML model were made in [Zamborlini et al. 2008]. Although this transformation has been used to create an application based on an heart's electrophysiology ontology [Gonçalves et al. 2007], no automated transformation was created from the OntoUML model to the OWL, i.e., the OWL ontology was created manually.

Two other OntoUML to OWL transformations already exists (none of them considers SWRL rules), both implemented at the Ontology Lightweight Editor (OLED). In this section we are going to discuss the conceptual aspects of these two different transformations: the OLED's Simple Transformation (Section 3.1) and the Temporal Transformation (Section 3.2).

In order to exemplify the differences between the transformations, we are going to consider the following OntoUML model, presented in Figure 9. This simple OntoUML model does not intent to represent the world as it is: it is just a syntactical valid model with simple concepts in order to be used as a valid input for the transformations presented in this paper. This diagram states that every Person has Headache and that Persons can be Drivers. To be a Driver the Person has to be related with one License that is related with one or more Cars. The Protégé 4.3 software (http://protege.stanford.edu/) was used to visualize the generated OWL ontologies.

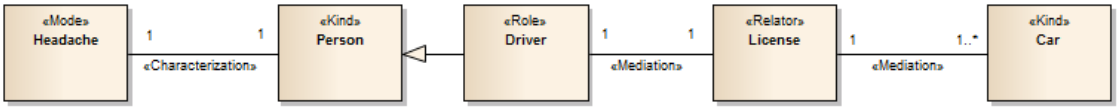

Figure 9 - Simple OntoUML model used as example

3.1. *OLED`s Simple Transformation*

The OLED's Simple Transformation, implemented by researcher Antognoni Albuquerque, was the first transformation from OntoUML to OWL that (similarly to the transformation proposed in this paper) did not included OntoUML stereotypes in the resultant OWL file. OLED's Simple Transformation treats the following cases in a different manner then we do: generalization set meta-properties and disjoint classes based on OntoUML stereotypes. It does not, however, treats: part-whole relations' meta-properties (nonetheless the user can create them using annotations in the OntoUML model) and transitivity, material relations' derivations, and structured DataTypes (it does treats simple DataTypes, creating OWL data properties). Another important design difference from this transformation to OntoUML2OWL+SWRL is the fact that it creates the OWL axioms as "subClassOf" instead of "EquivalentClasses". OLED's Simple Transformation does not consider temporal aspects.

Figure 10 represents, for the example model presented in Figure 9, the OLED's simple transformation for: (a) the class taxonomy, (b) the Object property taxonomy and (c) the License class description.

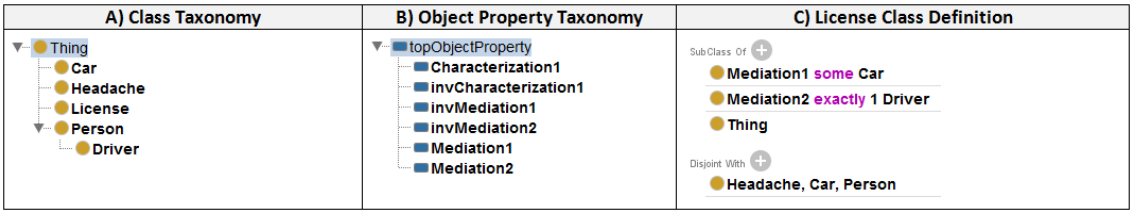

Figure 10 – OLED's Simple Transformation results

Considering the available transformations, the OLED's Simple Transformation is by far the most similar transformation to OntoUML2OWL+SWRL as both do not consider temporal aspects and as both do not intent to represent OntoUML or UFO (the foundational ontology which OntoUML is grounded) concepts in the generated OWL file. Yet, still comparing both transformations, OLED's Simple Transformation lacks in expressivity in comparison to OntoUML2OWL+SWRL as the latter considers more OntoUML restrictions when creating the OWL result.

3.2. *Temporal Transformation*

Similarly to this paper, [Zamborlini 2011] proposes alternatives for an OntoUML to OWL transformation concerned in to represent ontologies in an epistemological level language representing all ontological distinctions in order to guarantee the model quality. However, [Zamborlini 2011] has focus on temporal questions in the transformation process, while only static world is considered in the OntoUML2OWL+SWRL Transformation. Zamborlini's transformation is called here *Temporal Transformation* and it is also available in OLED.

The Temporal Transformation has two different approaches to consider temporal aspects of OntoUML in OWL. These approaches are (a) the Reification and (b) the Worm View.

a) Reification Transformation

Reification can be understood as the objectification of something so one can refer to it, qualify it and quantify it.

The focus in this transformation is the ontological difference between Objects and Moments, in which mutable information of individuals are reified. Then, the reification covers different types of moments. In this way, every others entities are mapped as Objects.

Applying the Reification Transformation to the OntoUML example model presented in Figure 9 we can see (a) the class taxonomy, (b) the Object property taxonomy and (c) the License class description in Figure 11.

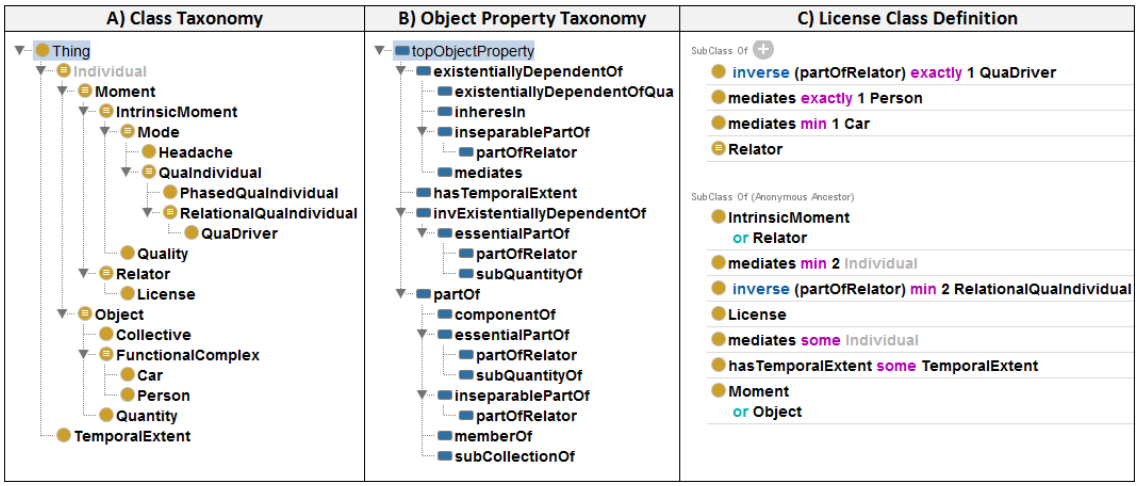

Figure 11 - Temporal Reification Transformation results

b) Worm Views Transformations

The OLED tool implements three different Worm View temporal considerations over OWL, they are called A0, A1 and A2.

In this approach individuals are considered spatiotemporal worms whose temporal parts are worm slices, in a way that individuals are composed by temporal parts and individual concept that maps him. The OWL base structure to represent this approach is divided in two different levels, the static one, called the *Individual Concept Level* (ICL), and the dynamic one, called the *Time Slice Level* (TSL). These three implementations are:

A0: Rigid concepts are represented in ICL; other concepts, relations, attributes in TSL.

A1: Rigid concepts, necessary and immutable attributes, and relations that implies in mutual existential dependency are represented on the ICL, and concepts, relations that not implies in mutual existential dependency and attributes not necessary and immutable simultaneously, on the TSL.

A2: Rigid concepts, necessary and immutable attributes, and relations that implies in existential dependency are represented on the ICL, and concepts, relations that not implies in existential dependency and attributes not necessary and immutable simultaneously, on the TSL.

As can be noticed, the Temporal Transformation has huge different considerations from OntoUML2OWL+SWRL because it is focused in the representation of temporal aspects. These transformations present a more expressive OWL as a result, but to do this it mix domain concepts with OntoUML and UFO concepts (see Figure 11) which demands that the OWL user (a person or a computational application) has this previous knowledge in order to understand and manipulate the output of the transformation. OntoUML2OWL+SWRL have as premise that just domain concepts are created in the OWL, resulting in a comprehensive OWL file.

4. Conclusions

This paper presents a Model Driven Architecture automated transformation from OntoUML to OWL with SWRL rules, named OntoUML2OWL+SWRL, that contributes to (i) make easier the OWL creation from OntoUML, (ii) eliminate the human errors in this process, (iii) improve the resultant OWL ontology semantics. This transformation is placed between two phases of an Ontology Engineering as it bridges the gap between two classes of languages with different purposes: (i) OntoUML, on one hand, is a well-founded ontology representation language focused on representation adequacy regardless of the consequent computational costs, which is not actually a problem since OntoUML models are targeted at human users; and (ii) on the other hand, OWL, a lightweight representation language with adequate computational properties.

Although two other OntoUML to OWL transformation exists (namely, the Simple OLED's transformation and the Temporal Transformation) OntoUML2OWL+SWRL have different transformation scope and it is placed between them in complexity. Differently from the other existent transformation, OntoUML2OWL+SWRL also create SWRL rules for representation of Mediations and Part-whole relations.

The conceptual transformation's design was presented with limitations and other implications inherent to these kinds of transformations. OntoUML2OWL+SWRL is implemented in OLED, a framework for OntoUML, and its code is open and fully available.

As the required OWL expressivity can be different depending on its application, the implementation of a parameterized transformation, where the user can choose which features the resulting OWL can have, is a future work. Also, transformation for specific OWL profiles can be created. As visual diagramming languages (including here OntoUML) are not always able to capture all relevant restrictions of a domain, they are usually incremented with restriction rules in Object Constraint Language (OCL). The coupling of an OCL to SWRL transformation to OntoUML2OWL+SWRL is desired. The DataType transformation can be improved in the future considering extensions to UFO presented in [Albuquerque and Guizzardi 2013], where the notion Semantic Reference Spaces to are employed to improve the ontological foundations concerning value spaces.

Acknowledgements. This research has been funded by FAPES/CNPq (PRONEX 52272362/11).

References

Albuquerque, A. and Guizzardi, G. (2013). An Ontological Foundation for Conceptual Modeling Datatypes based on Semantic Reference Spaces. In 7th IEEE International Conference on Research Challenges in Information Science (RCIS 2013).

Barcelos, P. P. F., Guizzardi, G., Garcia, A. S. and Monteiro, M. E. (may 2011). Ontological Evaluation of the ITU-T Recommendation G.805. In 2011 18th International Conference on Telecommunications. IEEE.

Gonçalves, B., Guizzardi, G. and Filho, J. G. P. (2007). An electrocardiogram (ECG) domain ontology. In 2nd Workshop on Ontologies and Metamodels for Software and Data Engineering.

Guizzardi, G. (2005). Ontological Foundations for Structural Conceptual Models. Enschede: Centre for Telematics and Information Technology University of Twente.

Guizzardi, G. (2007). On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. Proceedings of the 2007 conference on Databases and Information Systems IV: 7th International Baltic Conference, p. 18–39.

Guizzardi, G., Baião, F., Lopes, M. and Falbo, R. (2010). The Role of Foundational Ontologies for Domain Ontology Engineering: An Industrial Case Study in the Domain of Oil and Gas Exploration and Production. International Journal of Information System Modeling and Design, v. 1, n. 2, p. 1–22.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F. and Rudolph, S. (2012). OWL 2 Web Ontology Language Primer (Second Edition). http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

Miller, J. and Mukerji, J. (2003). MDA Guide Version 1.0.1. Object Management Group, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf.

Motik, B., Sattler, U. and Studer, R. (2005). Query Answering for OWL-DL With Rules. Web Semantics: Science, Services and Agents on the World Wide Web 3.1, v. 3, n. 1, p. 41–60.

Patel-Schneider, P. F., Hayes, P. and Horrocks, I. (2004). OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/.

Sales, T. P., Barcelos, P. P. F. and Guizzardi, G. (2012). Identification of Semantic Anti-Patterns in Ontology-Driven Conceptual Modeling via Visual Simulation. 4th International Workshop on Ontology-Driven Information Systems (ODISE 2012).

Zamborlini, V. C. (2011). Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML Para OWL: Abordagens Para Representação de Informação Temporal. Federal University of Espírito Santo. Available only in Portuguese.

Zamborlini, V. C., Gonçalves, B. and Guizzardi, G. (2008). Codification and Application of a Well-Founded Heart-ECG Ontology. In Third Workshop on Ontologies and Metamodeling in Software and Data Engineering - WOMSDE 2008.