A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML

Alessander Botti Benevides and Giancarlo Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo (UFES), Av. Fernando Ferrari, nº 514, Goiabeiras, Vitória (ES), Brazil
{abbenevides, gguizzardi}@inf.ufes.br

Abstract. This paper presents a Model-Based graphical editor for supporting the creation of conceptual models and domain ontologies in a philosophically and cognitively well-founded modeling language named OntoUML. The Editor is designed in a way that, on one hand, it shields the user from the complexity of the ontological principles underlying this language. On the other hand, it reinforces these principles in the produced models by providing a mechanism for automatic formal constraint verification.

Keywords. Ontology Engineering, Conceptual Modeling.

1 Introduction

Throughout the last years, ontologies have increasingly been applied in Computer Science. They have been a topic of research in Artificial Intelligence (AI) since the late seventies and more recently, in Software Engineering (SE). On the one hand, in the former, ontologies have been used as a knowledge representation technique to convey domain terminologies (*e.g.*, Description Logic T-Boxes) to be particularized as facts (*e.g.*, Description Logic A-Boxes) for serving to reasoning purposes. In the latter, on the other hand, ontologies have been mainly recognized as comprising a technique for developing enhanced domain-specific conceptual models.

There are two common trends in the use of ontologies in these two areas: (i) firstly, ontologies are always regarded as an explicit representation of a shared conceptualization, *i.e.*, a concrete artifact representing a model of consensus within a community and a universe of discourse. Moreover, in this sense of a reference model, an ontology is primarily aimed at supporting semantic interoperability in its various forms (*e.g.*, model integration, service interoperability, knowledge harmonization); (ii) secondly, the discussion regarding representation mechanisms for the construction of domain ontologies is, in both cases, centered on computational issues, not truly ontological ones. On one side, the AI community values representation languages which prime for computational tractability [1]. On the other side, the SE community is mostly concerned with committing to the use of standardized languages such as the Unified Modeling Language (UML) [2], and with producing ontology representations that facilitates the mapping to specific implementation environments (*e.g.*, Object-Oriented Frameworks [3]). Now, an important aspect to be highlighted is the

incongruence between (i) and (ii). As shown, for instance in Guizzardi [4], in order for an ontology to be able to adequately support (i), it should be constructed using an approach that explicitly takes into account a dimension which is neglected in (ii), namely, the use of foundational concepts that take truly ontological issues seriously.

In pace with Degen *et al.* [5], we argue that "every domain-specific ontology must use as framework some upper-level ontology". This claim for an upper-level (or foundational) ontology underlying a domain-specific ontology is based on the need for fundamental ontological structures, such as theory of parts, theory of wholes, types and instantiation, identity, dependence, unity, etc, in order to properly represent reality. From an ontology representation language perspective, this principle advocates that, in order for a modeling language to meet the requirements of expressiveness, clarity and truthfulness in representing the subject domain at hand, it must be an ontologically well-founded language in a strong ontological sense, *i.e.*, it must be a language whose modeling primitives are derived from a proper foundational ontology [6], [7].

An example of a general conceptual modeling and ontology representation language that has been designed following these principles is the version of UML proposed in [8]. This language (later termed OntoUML) has been constructed in a manner that its metamodel reflects the ontological distinctions prescribed by UFO (Unified Foundational Ontology). Moreover, formal constraints have been incorporated in this language's metamodel in order to incorporate the formal axiomatization in UFO. Therefore a UML model that is ontologically misconceived taking UFO into account is syntactically invalid when written in OntoUML.

Although this approach has been able to provide mechanisms for addressing a number of classical conceptual modeling problems [9], and the language has been successfully employed in application domains [10], [11], there was still no tool support for building and validating conceptual models and domain ontologies constructed using OntoUML. The main contribution of this paper is thus to present a Model-Based OntoUML Graphical Editor with support for automatic model checking in face of ontological constraints. The binaries and source code files for the editor are available at http://code.google.com/p/ontouml.

A snapshot of the main section of the editor is shown in Fig. 1 below. As one can see, there is a tool bar on the right side, where the user can drag and drop model elements on the left panel.

The remainder of this paper is structured as follows. Section 2 presents models that illustrates main concepts of UFO, represented using OntoUML, and validated on the

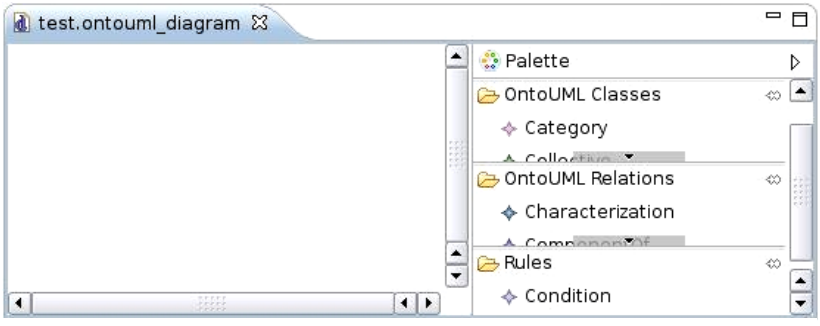

Fig. 1. Snapshot of the editor.

editor by automatically processing integrity and derivation rules that represent ontological constraints over the produced models. Section 3 briefly elaborates on the metamodeling and implementation technologies used in the construction of the editor. Section 4 presents some related work. Section 5 presents some final considerations.

2 Presentation of the Editor

In this section, we illustrate the support provided by the editor for automatically checking integrity constraints and deriving information in models. Integrity constraints are inspected via two different mechanisms named *Live Validation* and *Batch Validation*. In order to illustrate these features, let us make use of a simple domain model of car dealing. This simple universe of discourse is comprised of concepts such as Person, Car, CarCustomer, CarSupplier, Organization, Purchase and car parts (*e.g.*, Engine, Chassis). In the following we briefly exemplify how the editor can assist the user in the construction of a simple conceptual model in this domain.

2.1 Live Validation

In this conceptualization, Person would typically be modeled in OntoUML as a class with a <<kind>> stereotype, and a CarCustomer would be modeled as a class with a <<role>> stereotype, as is shown in Fig. 2. In OntoUML, the <<kind>> stereotype is used to represent the UFO Kind category, and the <<role>> stereotype represents the UFO Role category. In order to explain these UFO categories, we have to describe some more categories, which are represented on the excerpt of the UFO taxonomy in Fig. 3.

As we can see, Kinds and Roles are Entities, where Entity is the higher UFO category. Entity can be distinguished in Universal and Individual, where Individuals are entities that exist in reality possessing a unique identity, and Universals, conversely, are space-time independent pattern of features, which can be realized in a number of different individuals. In its turn, Universals can be distinguished in Monadic Universal and Relation (entities which glue together other entities). Within the category of Monadic Universal, in order to show the differences between Substance Universal and Relator Universal, we need to explicate what are Substances and Moments.

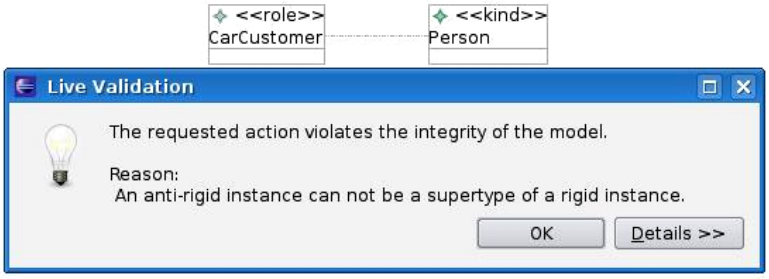

Fig. 2. Live validation example.

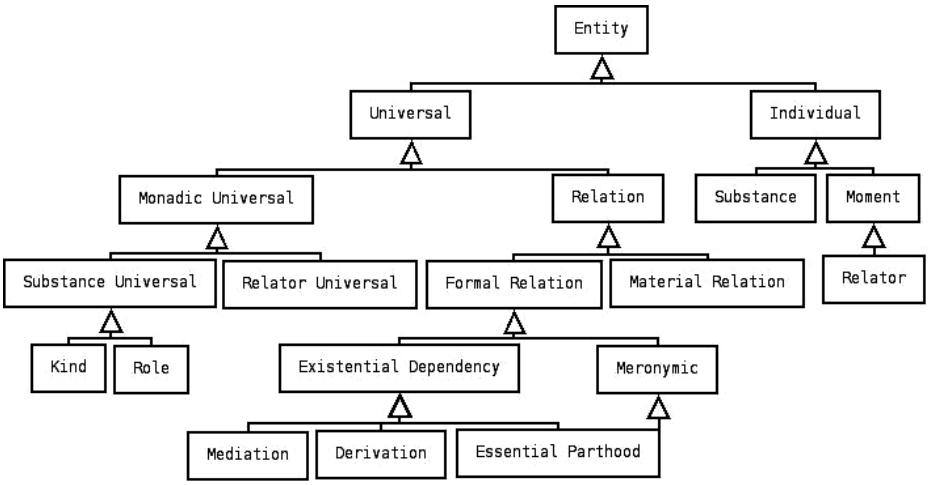

Fig. 3. Excerpt of UFO taxonomy [8].

Substances are existentially independent individuals, *i.e.*, there is no Entity x that must exist whenever a Substance y exists. Examples of Substances include ordinary mesoscopic objects such as an individual person, a house, a hammer, a car, but also the so-called *Fiat Objects* such as the North-Sea and its proper-parts, postal districts and a non-smoking area of a restaurant. Conversely, a Moment is an individual which can only exist in other individuals. Typical examples of moments are: a color, a connection, a purchase order.

So, a Substantial Universal is a universal whose instances are Substances (*e.g.*, the universal Person or the universal Apple). While, a Relator Universal is a universal whose instances are individual relational moments (*e.g.*, the particular enrollment connecting John and a certain University is an instance of the universal Enrollment).

We need to define some formal notions (rigidity and anti-rigidity) to be able to make further distinctions within Substance Universal. Definition 1 (Rigidity): A universal U is rigid if for every instance x of U, x is necessarily (in the modal sense) an instance of U. In other words, if x instantiates U in a given world w, then x must instantiate U in every possible world w'. ■ Definition 2 (Anti-rigidity): A universal U is anti-rigid if for every instance x of U, x is possibly (in the modal sense) not an instance of U. In other words, if x instantiates U in a given world w, then there must be a possible world w' in which x does not instantiate U. ■ [8].

A Substantial Universal which is rigid is named here a Kind. In contrast, an anti-rigid substantial universal is termed a Role. The prototypical example highlighting the modal distinction between these two categories is the difference between the universal (Kind) Person and the (Role) universal CarCustomer, both instantiated by the individual John in a given circumstance. Whilst John can cease to be a CarCustomer (and there were circumstances in which John was not one), he cannot cease to be a Person. In other words, in a conceptualization that models Person as a Kind and CarCustomer as a Role, while the instantiation of the role CarCustomer has no impact on the identity of an individual, if an individual ceases to instantiate the Kind Person, then it ceases to exist as the same individual. Moreover, [9] formally proves that a

rigid universal cannot have as its superclass an anti-rigid one. Consequently, a Role cannot subsume a Kind in our theory.

Now, as discussed in [9], a common mistake in conceptual modeling is the use of subtyping to represent alternative allowed types, *i.e.*, alternative types that supply players for a given role. In this particular case, suppose that the user attempts to represent that instances of Person are possible players of the role CarCustomer, by using subtyping. In other words, the user tries to model a Kind Person as a subtype of the Role CarCustomer. If allowed, this would not be an ontologically correct model, since it is not the case that every instance of Person is a CarCustomer, and since a Person cannot cease to be a Person but it can cease to be a CarCustomer. When attempting to create this ontologically incorrect model with the editor presented here, an integrity constraint is violated. As consequence, the editor ignores the corresponding model updating action and prompts a live validation pop-up that alerts the user of his attempt of creating an invalid model. The validation pop-up resulting from this example is shown in Fig. 2.

2.2 Deriving Model Information

In order to represent the relation between CarCustomer and Person, one should model CarCustomer as a Role played by Person in a certain context, where he buys a Car from a CarSupplier. Analogously, one should model CarSupplier as a Role played by an Organization when selling a Car to a CarCustomer. This context is materialized by the Material Relation *purchases* (represented as the <<material>> stereotype in OntoUML), which is in turn, derived from the existence of the Relator Universal Purchase (<<relator>>). In other words, we can say that a particular customer x purchases a particular car y from a particular supplier z iff there is a Purchase which mediates x, y and z. This situation is illustrated in Fig. 4. The mediation formal relations between the Relator Purchase and the Roles CarCustomer and CarSupplier are responsible for the existence of the derived Material Relation *purchases* that hold between CarCustomer and CarSupplier. Thus, the cardinality restrictions of the purchases relation can be systematically calculated from these associations as

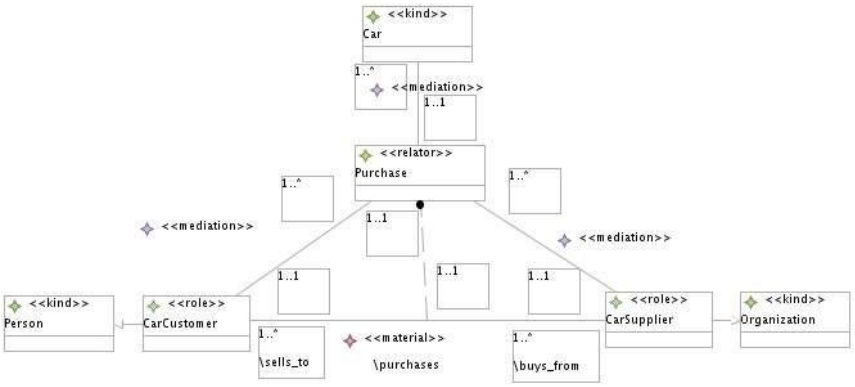

Fig. 4. Example of derivation of information.

exemplified in Fig. 5 below. The derivation of purchases from the mediation relations is represented by a Derivation association (pictured as a dashed line association between purchases and Purchase, where there is a black circle), which also have its cardinalities systematically calculated.

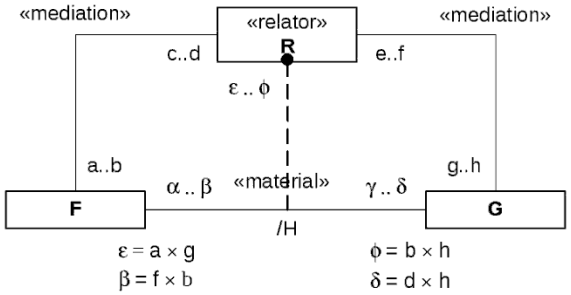

Fig. 5. Cardinality derivation.

In order to better explain what is a Material Relation, a Mediation relation (<<mediation>>) and a Derivation relation, we need to describe more categories represented in Fig. 3. The Relation category is differentiated in *Formal Relation* and *Material Relation*, where Formal Relations are relations that hold between two or more entities directly, without any further intervening individual, and Material Relations, conversely, do need an intervening individual. Examples of formal and material relations are older_than and purchases, respectively. The notion Formal Relation is further differentiated here in Existencial Dependency and Meronymic, where the former represents existentially dependent associations and the latter represents part-whole relations. For now, we can consider two types of existentially dependent Formal Relations: Mediation and Derivation. A Mediation relation is a relation that holds between a Substantial Universal and Relator Universal. Mediation and Relator Universal are the basis for defining Material Relations. In order to a Material Relation M_1 hold between two Substantial Universals S_1 and S_2, there must exists at least two Mediation relations (M_2 and M_3) and one Relator R, such that M_2 holds between S_1 and R and M_3 holds between S_2 and R. The Derivation relation is a relation between a Material Relation M_1 and the Relator Universals on which M_1 depends [8].

2.3 Batch Validation

A more complete version of a model in this domain is shown in Fig. 6, which represents some of the parts that compose a Car. In this figure, it is represented that a Car is composed of one CarEngine. However, part-whole relations must obey the so-called Weak Supplementation axiom, which, in simple words, states that in order to be a whole, an entity must have at least two disjoint parts. Therefore, to satisfy this axiom, if a Car is composed of one and only one Engine, it must also have another car component as a part. Now, differently from the Person-CarCustomer subtyping example discussed above, the lack of a second part represented in the model that would meet the requirement posed by the Weak Supplementation axiom can be due to

a momentary incompleteness of the model. In other words, after the part-whole relation between Car and CarEngine is represented, the user can still include information in the model that will prevent this model from being considered ontologically inconsistent. As this example shows, there are validation actions that should only be performed by the tool once the user deems suitable. Now, as illustrated in Fig. 6, if this model is validated with the presented information, the editor prompts to the user that, in that form, the model is considered incorrect. Furthermore, the editor informs the user by highlighting the source and reason of inconsistency in the model.

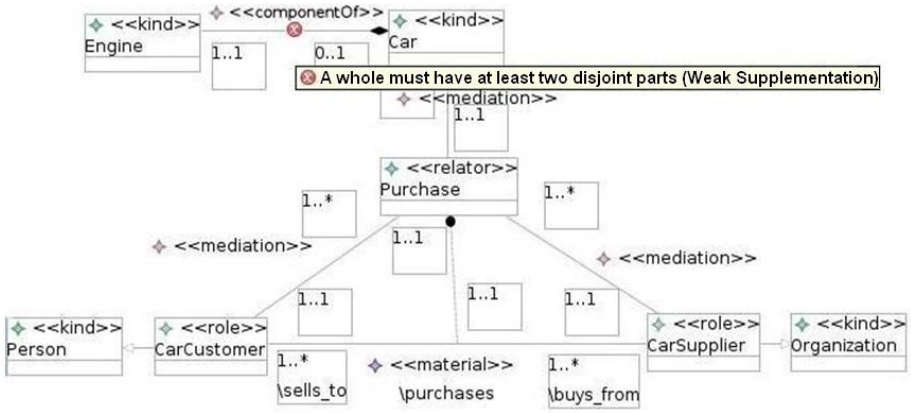

Fig. 6. Batch validation example.

A possible solution to this issue is to represent that a Car is composed of something more than an Engine, *e.g.*, a Chassis. Fig. 7 depicts this alternative representation where a car is composed of one and only one Engine and an essential unique Chassis, where the "essential" tag in this part-whole relations means that the whole is existentially dependent of the part [8].

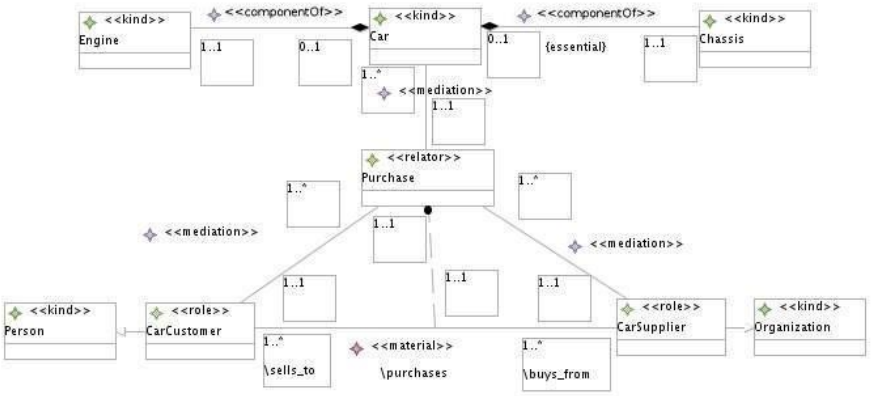

Fig. 7. A possible solution to correct the model pictured in Fig. 6 above.

3 The Architecture and Implementation of the Editor

The architecture of the editor presented here has been conceived to follow a Model-Driven Approach. In particular, we have adopted the OMG MOF (Meta-Object Facility) metamodeling architecture [12]. In order to describe constraints in UML/MOF (meta)models, the OMG also proposes the declarative formal language OCL (Object Constraint Language) [13]. On the formalization of the OntoUML profile we have used OCL expressions mainly to: define how derived attributes/associations get their values; define default values of attributes/associations, *i.e.*, define their initial values; specify query operations and specify invariants, *i.e.*, integrity constraints that determine a condition that must be true in all consistent system states.

The complete implementation of the OntoUML profile as a MOF metamodel is reported in [14]. The same reference also describes the full set of OCL expressions including: 8 OCL expressions to specify derivation rules; 145 OCL expressions to define default values; 13 OCL expressions to specify operations created to support some OCL derivation rules and invariants, and 69 invariants to model the constraints stated on the OntoUML profile [8]. An example of an OCL invariant representing the essential parthood axioms described in OntoUML is shown in the code below. One can notice that in this expression the modal existential dependence constraint of essential parthood from UFO is emulated via the existence condition (lower cardinality ≥ 1) plus the immutability constraint (isReadOnly = true).

```
context Meronymic

inv: if (self.isEssential = true) then self.target->
forAll(x | if x.oclIsKindOf(Property) then
((x.oclAsType(Property).isReadOnly = true) and
((x.oclAsType(Property).lower >= 1)) else false endif)
else true endif
```

In terms of implementation technology, the editor has been implemented using a number of plug-ins that supports graphical editor development in the context of the Eclipse IDE (Integrated Development Environment) [15]. For the creation of the OntoUML metamodel, we have used the Eclipse Modeling Framework (EMF) [16], [17] plug-in. This plug-in provides its own metamodeling language named ECore, which asides from few (mostly terminological) differences is equivalent to the EMOF (Essential MOF) language (a subset of the complete MOF 2.0 language) [12]. The EMF together with the Model Development Tools (MDT) [18] plug-in allows for the creation and validation of ECore models with embedded OCL constraints. Finally, to build the graphical interface of the editor, we have used the Graphical Modeling Framework (GMF) [19] plug-in. The GMF provides a high-level description of visual representations to support transformation to a set of java classes for the graphical editor using a Model-View-Controller (MVC) architecture. This process is schematically summarized in Fig. 8 below.

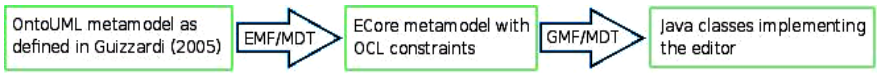

Fig. 8. Tool generation overview.

4 Related Work

As far as we know, there is no other tool for OntoUML. However, there are other editors that support philosophically well-founded languages and methodologies such as OntoClean [20], as well as tools based on upper-level ontologies as SUMO (Suggested Upper Merged Ontology) [21], SUO (Standard Upper Ontology) [22] and the Differential Semantics theory. For instance, Protégé [23] is a free open-source tool which supports OntoClean and SUMO. AEON (Automatic Evaluation of ONtologies) [24] is an open-source tool, which allows applying OntoClean to evaluate ontologies. Visual Ontology Modeler [25] is an editor that includes a library of ontologies that represent SUO. DOE (Differential Ontology Editor) [26] is a freeware ontology editor which allows the user to build ontologies according to the Differential Semantics theory. Sigma [27] is a free open-source knowledge engineering environment for theories in first order logic (FOL), which is optimized for SUMO.

5 Final Considerations

The need for using ontologically well-founded languages for conceptual modeling, in general, and domain ontologies, in particular, has increasingly been recognized in the literature. This is often a result of interoperability concerns and the unsuitability of lightweight representation languages in addressing these issues. Despite that, these languages are still not broadly adopted in practice. One of the main reasons is the need for high-level expertise in handling the philosophical concepts underlying them. Indeed, the dissemination of formal method techniques requires convincing industries and standardization bodies that such techniques in fact can improve development. In this way, design support tools are one of the key resources to foster their adoption in practice [28].

In this paper, we present an Eclipse-based graphical editor which aims at fulfilling the gap of tool support for one particular theoretically well-founded representation language, namely, OntoUML. Underlying this editor there is an implementation of the OntoUML metamodel proposed by Guizzardi [8] by using MDA (Model-Driven Architecture) technologies, in particular, the OMG MOF (Meta-Object Facility) and OCL (Object Constraint Language). Moreover, by representing UFO categories and axiomatization in the language metamodel, the complexity of these foundational issues is hidden from the user while still constraining him to produce ontologically sound models.

As a final remark, the promotion of a language such as OntoUML for domain engineering does not eliminate the need for codification languages such as OWL, DLR_{us}, Alloy or F-Logic, to cite just a few examples. In pace with the meaning independence principle defended by Guizzardi and Halpin [29], we adopt the view that these classes of languages are (and should be) meant to be used for different purposes and in different phases on an ontology engineering process.

References

1. Levesque, H.J., Brachman, R.J.: Expressiveness and Tractability in Knowledge Representation and Reasoning. In: Computational Intelligence, vol. 3, pp. 78--93 (1985)
2. Object Management Group (OMG): UML 2.0 Superstructure Specification, Doc.# ptc/03-08-02 (2003)
3. Falbo, R.A., Guizzardi, G., Duarte, K.C.: An Ontological Approach to Domain Engineering. In: ACM XIV International Conference on Software Engineering and Knowledge Engineering (SEKE-2002). Ischia, Italy (2002)
4. Guizzardi, G.: The Role of Foundational Ontology for Conceptual Modeling and Domain Ontology Representation. In: 7th International Baltic Conference on Databases and Information Systems. Vilnius, Lithuania (2006)
5. Degen, W., Heller, B., Herre, H., and Smith, B.: GOL: Toward an Axiomatized Upper-level Ontology. In: Proceedings of the international Conference on Formal ontology in information Systems - Volume 2001 (Ogunquit, Maine, USA, October 17 - 19, 2001). FOIS '01. ACM, New York, NY, pp. 34--46 (2001)
6. Guarino, N., Guizzardi, G.: In the Defense of Ontological Foundations for Conceptual Modeling. Invited Paper. In: Scandinavian Journal of Information Systems. Vol.18, No. 1, ISSN 0905-0167 (2006)
7. Guizzardi, G.: On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. In: Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV. Olegas Vasilecas, Johan Edler, Albertas Caplinskas (Editors), ISBN 978-1-58603-640-8, IOS Press, Amsterdam (2007a)
8. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models, Ph.D. Thesis, University of Twente, The Netherlands (2005)
9. Guizzardi, G., Wagner, G., Guarino, N., van Sinderen, M.: An Ontologically Well-Founded Profile for UML Conceptual Models. In: 16th International Conference on Advances in Information Systems Engineering (CAiSE), Latvia. Springer-Verlag, Berlin, Lecture Notes in Computer Science 3084, ISBN 3-540-22151-4 (2004)
10. Nunes, B.G., Guizzardi, G., Filho, J.G.P.: An Electrocardiogram (ECG) Domain Ontology. In: Proceedings of the Second Brazilian Workshop on Ontologies and Metamodels for Software and Data Engineering (WOMSDE'07), 22nd Brazilian Symposium on Databases (SBBD)/21st Brazilian Symposium on Software Engineering (SBES). João Pessoa, Brazil (2007)
11. Oliveira, F., Antunes, J., Guizzardi, R.S.S.: Towards a Collaboration Ontology. In: Proceedings of the Second Brazilian Workshop on Ontologies and Metamodels for Software and Data Engineering (WOMSDE'07), 22nd Brazilian Symposium on Databases (SBBD)/21st Brazilian Symposium on Software Engineering (SBES). João Pessoa, Brazil (2007)
12. Object Management Group (OMG): Meta Object Facility MOF Core Specification, v2.0, Doc.# ptc/06-01-01 (2006)
13. Object Management Group (OMG): Object Constraint Language, v2.0, Doc.# ptc/06-05-01 (2006)
14. Benevides, A.B.: A Model-Based Tool for Well-Founded Conceptual Modeling (in portuguese), Computer Engineering Monograph, Computer Science Department, Federal University of Espirito Santo (2007)
15. Eclipse, http://www.eclipse.org
16. Dean, D., Gerber, A., Moore, B., Vanderheyden, P., Wagenknecht, G.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, IBM Redbooks (2004)
17. Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf
18. Model Development Tools (MDT), http://www.eclipse.org/modeling/mdt

19. Graphical Modeling Framework (GMF), http://www.eclipse.org/modeling/gmf
20. OntoClean, http://www.ontoclean.org
21. SUMO, http://www.ontologyportal.org
22. IEEE, http://suo.ieee.org
23. Protégé, http://protege.stanford.edu
24. AEON, http://ontoware.org/projects/aeon
25. Sandpiper Software, http://www.sandsoft.com
26. DOE, http://homepages.cwi.nl/~troncy/DOE
27. Sigma, http://sigmakee.sourceforge.net
28. Vissers, C., van Sinderen, M., Pires, L.F.: What Makes Industries Believe in Formal Methods. In: Proceedings of the 13th International Symposium on Protocol Specification, Testing, and Verification (PSTV XIII). Elsevier Science Publishers, pp. 3--26 (1993)
29. Guizzardi, G., Halpin, T.: Ontological Foundations for Conceptual Modeling, Applied Ontology, Volume 3, pp. 91--110, Number 1-2, ISSN 1570-5838 (2008)