Semantic Web 1 (2010) 3–10
DOI 10.3233/SW-2010-0015
IOS Press

Theoretical foundations and engineering tools for building ontologies as reference conceptual models

Editors: Krzysztof Janowicz, Pennsylvania State University, USA; Pascal Hitzler, Wright State University, USA
Solicited Reviews: Oscar Corcho, Universidad Politécnica de Madrid, Spain; Pascal Hitzler, Wright State University, USA

Giancarlo Guizzardi
Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo (UFES), Vitória, Espírito Santo, Brazil
E-mail: gguizzardi@inf.ufes.br

Abstract. Perhaps the most fundamental notion underlying the desiderata for a successful Semantic Web is *Semantic Interoperability*. In this context, ontologies have been more and more recognized as one of the enabling technologies. This paper defends the view that an approach which neglects the role of ontologies as *reference conceptual models* cannot meet the requirements for full semantic interoperability. The paper starts by offering an engineering view on ontology engineering, discussing the relation between *ontologies as conceptual models* and *ontologies as codification artifacts*. Furthermore, it discusses the importance of foundational theories and principles to the design of ontology (conceptual) modeling languages and models, emphasizing the fundamental role played by *true ontological notions* in this process. Finally, it elaborates on the need for proper tools to handle the complexity of ontology engineering in industrial scenarios and complex domains. These tools include *ontological design patterns* as well as well-founded computational environments to support ontology creation, verification and validation (via model simulation).

Keywords: Conceptual modeling, foundational ontology, methodological and computational tools for ontology engineering

1. Introduction

Perhaps the most fundamental notion underlying the desiderata for a successful Semantic Web is *Semantic Interoperability*. To a large extent, the Semantic Web is about offering support for complex information services by combining information sources that have been designed in a concurrent and distributed manner. In this context, ontologies have been more and more recognized as one of the enabling technologies.

In general, in computer science, ontologies have been used either as a *reference model of consensus* to support semantic interoperability, or as an explicit, declarative and machine processable artifact coding a domain model to enable automated reasoning. This duality, however, points to different (and even conflicting) sets of quality criteria that should be met by the representation languages employed to construct these ontologies.

On one hand, ontologies considered as reference conceptual models for semantic interoperability should be constructed in manners that maximize, on one hand, the expressivity in capturing fundamental aspects of the underlying domain and in making explicit the underlying *ontological commitments*. On the other hand, they should also be designed to maximize conceptual clarity (or pragmatic efficiency) to afford the tasks of communication, domain understanding, problem-solving and meaning negotiation among human users. In contrast, ontologies as reasoning artifacts for the semantic web, should be built in a way that supports decidable and computationally tractable automated reasoning.

The first idea defended in this article can be summarized in the following manner. If Ontology Engineering is to become a mature engineering discipline, able to construct and manage artifacts in a range of complex domains, it must incorporate a number of lessons learned from other closely-related engineering disciplines. This starts with the acknowledgement that *there is no Silver Bullet*! From a language point of view, this means that we should not attempt to produce one single representation system (with associated methodological tools). In contrast, we should recognize that different representation systems of different nature are needed in different phases of an ontology engineering process. This idea is articulated in Section 2.

The second point I want to make is that, in order to meet the quality criteria outlined above for producing ontologies as reference conceptual models (i.e., ontological expressivity and conceptual clarity), we cannot eschew *truly ontological* questions. In other words, we need an ontology conceptual modeling language that assists modelers in: (i) making explicit the ontological commitment assumed in that conceptualization; (ii) producing representation structures that do justice to the nature of the underlying reality. The design of such a language can greatly benefit from theories produced in disciplines such as Formal Ontology in Philosophy, Cognitive Science, Linguistics and Philosophical Logics. This point is elaborated in Section 3.

Finally, for us to be able to count on a systematic engineering discipline that can be used to establish full and successful semantic interoperability in heterogeneous and complex real-world scenarios, we need proper methodological and computational tools to handle that complexity. This is the third topic of this paper which is discussed in Section 4.

These three points are non-orthogonal, in the sense that, in the way I have presented in this article, the solutions outlined in Section 4 dependent on the acceptance and implementation of the views advocated in Sections 2 and 3. For this reason, the latter sections can also be understood as background knowledge for the former. Furthermore, this accounts for a certain unbalance in length between these sections.

2. An engineering view to ontology engineering

A domain ontology is a special kind of conceptual model, i.e. an engineering artifact with the additional requirement to represent a model of consensus within a community. This model is designed to facilitate individuals to share information about that domain by conforming to some standard set of constructs. For this reason, this activity should be structured in process paths that are analogous to the ones practiced in other disciplines that also support the transition from a representation of a conceptualization to some coding artifact. In this transition path, the process must take into account that the produced coding artifact should preserve not only the real-world semantics of the original representation but it should also typically comply with a number of non-functional requirements particular to a specific computational environment.

In disciplines such as Software and Information Systems Engineering, there is a clear distinction between Conceptual Modeling, Design and Implementation. In Conceptual Modeling, a solution-independent specification is produced with the aim to make a clear and precise description of the domain elements. In the Design phase, this conceptual specification is transformed into a logical design specification (e.g. a relational database schema or an object class model) by taking into consideration a number of issues ranging from architectural styles, non-functional quality criteria to be maximized (e.g., performance, adaptability), target implementation environment, etc. The same conceptual specification can potentially be used to produce a number of (even radically) different logical designs. Finally, in the Implementation phase, a physical design is coded in one or more target languages to be then deployed in a computational environment. Again, from the same logical design, a number of different implementations can be produced. Design, thus, bridges Conceptual Modeling and Implementation.

The same reasoning should be applied to the discipline of Ontology Engineering [8]. Firstly, in a conceptual modeling phase in Ontology Engineering, the main requirements for the resultant models (and, hence, for the modeling languages) are *domain appropriateness* and *comprehensibility appropriateness* [7]. These requirements mean that on the one hand, the models should be truthful to the phenomena being represented. And on the other hand, it should be clear for users of the language to understand what elements of the universe of discourse are represented by elements of the model, as well as what problem-solving operations are to be performed on these elements. Consequently, the features of a modeling language that maximize these quality attributes should not be sacrificed in favor of issues such as decidability and computational efficiency for automatic reasoning (which are design concerns, not conceptual ones).

Secondly, as a conceptual model of reference, an ontology can then be used to produce several different alternative implementations in different codification languages (e.g., OWL DL, RDF, F-Logic, DLR_{US}, Haskell [1], Relational Database languages, CASL, among many others). The choice of each of these languages should be made to favor a specific set of non-functional requirements. Moreover, within the solution space defined by these codification languages, we have a multitude of choices regarding, for instance, decidability, completeness, computational complexity, reasoning paradigm (e.g., closed versus open world, adoption of a unique name assumption or not), expressivity (e.g., regarding the need for representing modal constraints, higher-order types, relations of a higher arity), verification of finite satisfiability, among many others. The point here is that the choice of a particular codification language can only be justified as a *design choice*. To put it baldly, the question is not whether, for instance, OWL is good or not for representing ontologies. The question is whether OWL is *justifiable as an adequate design choice in a specific design scenario*. At this point, I would like to echo the historical report of Janis Bubenko regarding an analogous discussion taking place in the conceptual modeling community in the 70's between supporters of *Conceptual Data Models* (e.g. ER diagrams) and those of the *Relational Data Model* [3]. As summarized by Bubenko, "*[t]oday the battle is settled: conceptual data models are generally used as high-level problem oriented descriptions. Relational models are seen as implementation oriented descriptions*".

To complete the view outlined above, between the phases of *Ontology Conceptual Modeling* and *Ontology Codification*, we need a phase of *Ontology Design* that provides methodological supports for: (i) systematic exploration of the solution space, hence, supporting reasoning with possible choices of codification technology as well as their ability to satisfy a specific set of non-functional requirements; (ii) mapping from the conceptual to a selected codification language with the goal of preserving as much as possible the real-world semantics of the original model while still attempting at *satisficing* the non-functional requirements at hand.

This rationale has received much attention in the context of the OMG's MDA (Model-Driven Architecture) initiative which aims at improving model-reuse via separation of concerns. In that scenario, due to recognition of the elevated costs of producing high-quality domain representations, there is a clear understanding that these representations should be independent of computational concerns (hence the term *Computational Independent Model*). The idea is to prevent these models from becoming deprecated due to changes which are purely related to technological choices.

Finally, there is an important additional aspect which I would like to draw attention to and which directly comes to mind when thinking about reference models. The role of a domain reference model is to provide a *frame of reference*, i.e., to serve as a conceptual tool for mastering the complexity and harmonizing possibly heterogeneous viewpoints and terminologies regarding a domain. Such a reference model is commonly used as a frame for producing implementations (including ones with automated reasoning). However, it can also be used in an off-line manner in a multitude of other meaning negotiation tasks. In other words, a Reference Conceptual Model has a value in itself, independent of the implementations that can be derived from it.

3. Revisiting the ontological level

In this section, I focus on ontologies as Reference Conceptual Models. Given the nature of possible applications of an ontology in this sense, a conceptual modeling language for producing high-quality ontologies should be able to: (i) allow the conceptual modelers and domain experts to be explicit regarding their ontological commitments, which in turn enables them to expose subtle distinctions between models to be integrated and to minimize the chances of running into a *False Agreement Problem* [5]; (ii) support the user in justifying their modeling choices and providing a sound design rationale for choosing how the elements in the universe of discourse should be modeled in terms of language elements.

Regarding (i), in order for a conceptual modeling language to be able to produce truthful specifications of a domain conceptualization, it must offer modeling primitives which are able to capture the nuances and subtleties involving the very *essence* of the elements constituting that domain. As recognized in the Harvard Business Review report of October 2001:

[1] See the paper "Modeling vs. Encoding for the Semantic Web" by Werner Kuhn in this inaugural issue.

"*one of the main reasons that so many online market makers have foundered [is that]the transactions they had viewed as simple and routine actually involved many subtle distinctions in terminology and meaning*"[2]. Corroborating this point, [7] demonstrates a number of semantic interoperability problems that can arise when integrating even simple lightweight ontologies. Additionally, [6] elaborates on cases of semantic overload involving concepts which are central to a domain (e.g., the concept of Petroleum for a Petroleum company!) that pass undetected even within the same organization. In both these cases, the problems are related to the inability of the modeling approach used in giving support for establishing precise meaning agreements.

Regarding (ii), I would like to revisit a classification put forth by Nicola Guarino is his seminal paper "*The Ontological Level*" [4]. As discussed there, *Logical-Level* languages (e.g., FOL) are "flat" in the sense that they put all predicative terms (e.g., Apple and Red) in the same footing; *Epistemological-Level* languages (e.g., UML, ER, OWL) provide ways for elaborating structures which differentiate these terms. For instance, in UML: (a) we can define a Class of Apples with an attribute *color=red*; or (b) we can define a Class of Red with an attribute *type=apple*. What an Epistemological-Level language does not give us is a precise criterion for explaining why structure (a) is better than (b). As discussed in that paper, structuring decisions, such as this one, should not be the result from heuristic considerations, but they should rather reflect important *ontological distinctions* that should be motivated and explained. For instance, in this case, the choice between these modeling alternatives reflects a choice between sorts of object types of completely different nature, and which entails radically different consequences both in theoretical and practical terms [7].

In summary, in order to meet the *desiderata* in (i) and (ii), we need the support of a system of truly *Ontological Categories.* This system should comprise a body of formal (i.e., domain independent) theories postulating ontological distinctions, as well as a rich axiomatization prescribing how these distinctions can be related. Moreover, this system of categories should be embedded in a language system, i.e., we need a modeling language with a set of constructs that honor these ontological distinctions.

A language designed with the specific purpose of addressing these issues for the case of *Structural Conceptual Models* is the version of UML 2.0 proposed in [7] and latter dubbed OntoUML. This language reflects a system of categories postulated by an underlying reference ontology of endurants (objects), based on a number of theories from Formal Ontology, Philosophical Logics, Philosophy of Language, Linguistics and Cognitive Psychology. As a result, the language offers a rich set of primitives capturing fine-grained distinctions among, for example: (i) part-whole relations; (ii) object types, (iii) properties; (iv) forms of ontological dependence, etc. In the next section, we refer to OntoUML to illustrate some of the points discussed there.

[2] I thank Nicola Guarino for bringing this text to my attention.

4. The humble ontologist

In his ACM Turing Award Lecture entitled "*The Humble Programmer*" [11], E.W. Dijkstra discusses the sheer complexity one has to deal with when programming large computer systems. His article represented an open call for an acknowledgement of the complexity at hand and for the need of more sophisticated techniques to master this complexity.

I believe that we are now in an analogous situation with respect to conceptual modeling, in general, and ontology construction, in particular. We will experience an increasing demand for building and using reference ontologies in subject domains in reality for which sophisticated ontological distinctions are demanded. As discussed in the previous section, we need ontologically sound representation languages. However, for the sake of scalability and separation of concerns, the ontology engineering practitioner should not be required to deal with all the intricacies of the theories underlying the language. In other words, on the one hand, we need to offer to the working ontologist, theories and modeling distinctions as expressive as possible. On the other hand, we need as much as possible to shield this practitioner from the complexity of these conceptual tools.

In the sequel, I discuss three of the possible kinds of tools that can be used to master the inherent complexity of this process.

4.1. Ontological design patterns

In software engineering, design patterns have become a way to capture in a standard form a solution to a recurrent problem. As recognized by the community of pattern languages, patterns are actually not only a means for reusing expert's knowledge. More than that, they define a language to talk about design, having become part of the area's jargon. In other words, people exchange patterns as signs with specific and shared semantics within that community as opposed to having to repeatedly explain the situation that motivated their creation.

In ontological engineering, there are obvious opportunities to take advantage of a similar approach. Due to space limitations, I will comment here on just two classes of these patterns, namely, *modeling patterns* and *transformation patterns.* For an example of *analysis patterns* proposed to identify the scope of transitivity of parthood, one can refer to [7].

Firstly, we need patterns that can be used to represent domain-independent solutions to modeling problems that can be manifested in several domains. These patterns shall be motivated by formal ontological reasons and (also because of that) I predict that they will hardly be identified in an approach that neglects formal ontological categories. Examples of patterns in this class have been proposed, for instance in [7], to address modeling problems such as: (i) role modeling with disjoint admissible types; (ii) modeling of material relations and their truth-makers (relational properties); (iii) separating entities from their constitutions; (iv) representation of qualities with alternative associated quality spaces; (v) harmonizing alternative notions of roles, among others. It is important to highlight that, in the case of all these patterns, the modeling solutions proposed result from an ontological analysis of the problems at hand in terms of the categories and theories of an underlying foundational ontology.

More than collecting a number of useful Modeling Patterns, we should pursue the construction of ontology modeling languages which are *pattern languages*. OntoUML is a language which has such a feature to a large extent. In that language, it is common that the choice of modeling a domain element using a particular construct causes a whole pattern to be manifested [7]. This opens the possibility for an editor that supports the user in modeling with elements of higher-granularity and cohesion, i.e., instead of simply using isolated primitives such as classes, associations and attributes, the models would be constructed with pattern blocks instantiating formal relations from a foundational theory.

Secondly, we need *transformation patterns* capturing standard solutions to problems of mapping ontologically rich models to languages which are less expressive or have specific characteristics. A number of examples of patterns in this category which aim at supplanting the limitations of OWL have been collected in ODP Portal[3]. However, other patterns in this class have also been proposed considering radically different paradigms. For instance, [10] proposes a pattern which captures a solution to the problem of preserving the basic semantics of mereological relations in traditional Object-Oriented implementations. In fact, there are many opportunities for employing standard OO Patterns such as Composite, Delegation, State and Observer to propose standard solutions for implementing ontology-related issues such as transitive propagation of properties, multiple and anti-rigid classification, and existential dependency. Having the source model represented in an ontologically rich language provides a direct guidance for when and how to apply these patterns.

4.2. Model-driven editors

As previously discussed, the OntoUML metamodel contains: (i) elements that represent ontological distinctions prescribed by an underlying foundational ontology; (ii) constrains that govern the possible relations that can be established between these elements. Let us illustrate these points by using the distinction between the object type *Kind* and *Roles*. In a simplified view we can state that: a Kind is a type that congregates all the essential properties of its instances and, for that reason, all instances of a Kind cannot cease to instantiate it without ceasing to exist; a Role, in contrast, represents a number of properties that instances of a Kind have contingently and in a relational context. A stereotypical example of this distinction can be appreciated when contrasting the Kind Person and the Role Student [7]. Regarding (i), OntoUML incorporates constructs that represent both of these ontological categories. Regarding (ii), the metamodel embeds constraints such as: a role must be a subtype of exactly one ultimate Kind; a role cannot be a supertype of a Kind.

Because these distinctions and constraints are explicitly and declaratively defined in the metamodel, they can be directly implemented using metamodeling architectures such as the OMG's MOF (Meta Object Facility). Following this strategy, [1] reports on an implementation of OntoUML graphical editor by employing a number of basic Eclipse-based frameworks such as the ECore (for metamodeling purposes), MDT (for the purpose of having automatic verification of OCL constraints) and GMF (for the purpose of building a model-based graphical interface). An interesting aspect of this strategy is that, by

[3] http://ontologydesignpatterns.org/

incorporating ontological and semantic constraints in the metamodel (i.e., the abstract syntax) of the language, it mimics a process which also takes place in natural language.

As an example of the latter point, take the two sentences: *(i) (exactly) five mice were in the kitchen last night; (ii) the mouse which has eaten the cheese has been in turn eaten by the cat.* If we have the patterns *(exactly) five X...* and *the Y which is Z...*, only the substitution of X,Y,Z by common nouns will produce sentences which are grammatical. To see that, one can try the replacement by the adjective Red in the sentence *(i): (exactly) five red were in the kitchen last night.* Now, the reason for why this is the case is an ontological one [7]. The interesting aspect here is that the competent user of this natural language does not need to know that! In other words, one can (as most language speakers do) abstract from the ontological reasons behind a grammatical constraint.

We should pursue the same ideal in ontology conceptual modeling languages. For example, one does not need to be fully aware of the reason why *a Role cannot be a supertype of Kind*. Actually, following the strategy adopted for the OntoUML tool editor, the user does not even have to be aware of the syntactic rule either: if one tries to produce a model violating this rule, this will be identified by the embedded OCL constraint checker of the tool, and the modeler will be promptly notified about the forbidden action.

Another important advantage of having an ontology language with an explicitly defined metamodel is the possibility of implementing multiple transformations from an ontology conceptual model to different codification schemes. Again, metamodel transformation is a widespread practice by the followers of OMG's MDA initiative (refer to Section 2). In this spirit, once we have a transformation model defined between, for example, the OntoUML and the OWL metamodels, every model in the first language can be automatically transformed into a specification in the second one. For example, [2] implements a transformation model from OntoUML to the constraint language Alloy by using the ATL language (a popular implementation of the OMG's QVT model). This mapping enables the creation of the model simulator discussed in the next section.

4.3. Model simulators

Having a modeling language whose metamodel incorporates the ontological constraints of a foundational theory directly eliminates the representation of *ontologically non-admissible states of affair*. However, it cannot guarantee that only *intended states of affair* are represented by the domain model at hand [21]. This is because the admissibility of domain-specific states of affair is a matter of factual knowledge (regarding the world being the way it happens to be), not a matter of *consistent possibility*.

To illustrate this point, suppose a medical domain ontology representing the procedure of a transplant. In this domain, we have concepts such as Person, Transplant Surgeon, Transplant, Transplanted Organ, Organ Donor, Organ Donee, etc. A *transplant conceptual model* which places Organ Donor (role) as a supertype of Person (kind), or one that represents the possibility of a Transplant (event) without participants clearly violates ontological rules. However, these two cases can be easily detected and proscribed by an editor such as the one discussed in the Section 4.2. The issue here is that in this case one can still produce a model which does not violate any of these rules but which still admits unintended states of affair as valid instances. One example is a state of affair in which the Donor, the Donee and the Transplant Surgeon are one and the same Person. Please note that this state of affair is only considered inadmissible due to domain-specific knowledge of social and natural laws. Consequently, it cannot be ruled out *a priori* by a domain independent system of categories.

Guaranteeing the exclusion of unintended states of affair without a computational support is a practically impossible task for any relevant domain. In particular, since many fundamental ontological distinctions are modal in nature [7], in order to validate a model, one would have to take into consideration the possible valid instances of that model in all possible worlds.

In [2], an extension to the OntoUML editor was presented that offers a contribution to this problem. On the one hand, it aims at proving the satisfiability of a given ontology by presenting a valid instance (logical model) of that ontology. On the other hand, it attempts to exhaustively generate instances of the ontology in a branching-time temporal structure, thus, serving as a visual simulator for the possible dynamics of entity creation, classification, association and destruction. The snapshots in this world structure confront a modeler with states of affair that are deemed admissible by the ontology's current axiomatization. This enables modelers to detect unintended states of affair and to take the proper measures to rectify the model. The assumption is that the example world structures support a modeler in this validation process, especially since it reveals how states of affair change in time and how they may eventually evolve in counterfactual scenarios.

5. Final considerations

As we argue throughout this paper, one of the key aspects of the Semantic vision is Semantic Interoperability. If conceptual modeling is about "*the construction of models of reality that promote a common understanding of that reality among their human users*" [11], then successful semantic interoperability is about harmonizing different viewpoints reflected in different conceptualizations of that same reality. In any case, reality cannot be left out of the loop. As a consequence, an approach which neglects the role of ontologies as reference conceptual models cannot meet the requirements for semantic interoperability.

The Semantic Web vision puts forth an undoubtedly inspiring challenge. Moreover, it brought us a number of interesting results from serious and talented researchers working on the field. However, in that context, there has been an unbalanced focus on developing representation techniques to support efficient reasoning. In contrast, the very task of domain representation, i.e., the task of constructing principled conceptual structures that represent with truthfulness and clarity the underlying domain, has been left to the user. Another negative aspect that must be brought to attention regarding the Semantic Web is that, due to its popularity, the hype wave it has generated also brought us a lot of *noise*. I believe this seriously harmed the establishment of a clear view of the sheer complexity involved in the problem at hand.

Since ontology engineering is a young discipline, there are many lessons to be learned from closely related areas such as Software Engineering, Information Systems and Databases. One of these is that the quality of any implementation artifact based on a model is ultimately bound by the quality of that model. Another one is that the area must properly define its problem and solution spaces as well as to bridge them effectively. The latter bears strong ties with the topic of *Ontology Education*, a subject which has gained much interest in the international community recently[4]. I am afraid until we have a minimally agreed curriculum or body of knowledge[5] to guide the education of ontologists, many of our discussions will still be carried out by engineers that lack true *ontological* knowledge as well as formal ontologists that lack basic industrial experience and sensitivity to the need of engineering tools.

This paper elaborates on a research program that addresses exactly the conceptual modeling phase of Ontology Engineering, focusing on the development of foundational theories, modeling languages and methods, design patterns and supporting computational environments that aim at supporting the construction of ontologies as reference models. The paper reflects on a limited list of items and there are many other fundamentally important issues regarding ontology engineering which I did not deal with here.

There are still many challenges to be met before we can have a mature discipline of Ontology Engineering. The road ahead of us is both challenging and exciting and, once more paraphrasing Dijkstra, we should do a much better ontology engineering job in the future, "*provided that we approach the task with a full appreciation of its tremendous difficulty*".

Acknowledgements

This research is supported by FAPES (Grant# 45444080/09) and CNPq (Grant# 481906/2009-6).

References

[1] Benevides, A.B., Guizzardi, G., A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML, *Proceedings of the 11th ICEIS, Milan*, 2009.

[2] Benevides, A.B., Guizzardi, G., Braga, B.F.B., Almeida, J.P.A., Assessing Modal Aspects of OntoUML Conceptual Models in Alloy, *Proc. of the 1st ETheCom, Gramado*, 2009.

[3] Bubenko Jr., J.A., From Information Algebra to Enterprise Modelling and Ontologies – a Historical Perspective on Modelling for Information Systems, *Conceptual Modeling in Information Systems Engineering*, Springer-Verlag, 2007.

[4] Dijkstra, E.W., The Humble Programmer, *Communications of the ACM*, **15**:10, October 1972.

[5] Guarino, N., Formal Ontology and Information Systems, *Proceedings of the 1st FOIS, Trento, Italy, June 6–8*. IOS Press, Amsterdam: pp. 3–15, 1998.

[6] Guarino, N., The Ontological Level, In R. Casati, B. Smith and G. White (eds.), *Philosophy and the Cognitive Science*. Holder-Pivhler-Tempsky, Vienna: pp. 443–456, 1994.

[7] Guizzardi, G., *Ontological Foundations for Structural Conceptual Models*, Telematica Instituut Fundamental Research Series No. 15, ISBN 90-75176-81-3, The Netherlands, 2005.

[8] Guizzardi, G., Halpin, T., Ontological Foundations for Conceptual Modeling. *Applied Ontology*, v. 3, p. 91–110, 2008.

[9] Guizzardi, G., Lopes, M., Baião, F., Falbo, R., On the impor tance of truly ontological representation languages, *International Journal of Information Systems Modeling and Design (IJISMD)*, IGI-Global, 2010.

[4] See the discussions on the "Ontology Summit 2010: Creating the Ontologists of the Future" (http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2010).

[5] For a contrast, see the IEEE Guide to the Software Engineering Body of Knowledge (http://www.swebok.org/).

[10] Guizzardi, G., Falbo, R.A., Pereira Filho, J.G., Using objects and patterns to implement domain ontologies, *Journal of the Brazilian Computer Society*, ISSN 0104-6500, **8**:1, July 2002.

[11] Mylopoulos, J., Conceptual modeling and Telos. In P. Loucopoulos & R. Zicari (eds.), *Conceptual Modeling, Databases, and CASE* (Chapter 2, pp. 49–68). Wiley, 1992.