Design Patterns and Inductive Modeling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML

Giancarlo Guizzardi, Alex Pinheiro das Graças, Renata S.S. Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO)
Computer Science Department,
Federal University of Espírito Santo (UFES), Brazil
{gguizzardi,agracas,rguizzardi}@inf.ufes.br

Abstract. In recent years, there has a growing interest in the use of Ontologically Well-Founded Conceptual Modeling languages to support the domain analysis phase in Information Systems Engineering. OntoUML is an example of a conceptual modeling language whose metamodel has been designed to comply with the ontological distinctions and axiomatic theories put forth by a theoretically well-grounded Foundational Ontology. However, despite its growing adoption, OntoUML has been deemed to pose a significant complexity to novice modelers. This paper presents a number of theoretical and methodological contributions aimed at assisting these modelers. Firstly, the paper explores a number of design patterns which are derived from the ontological foundations of this language. Secondly, these patterns are then used to derive a number of model construction rule sets. The chained execution of these rule sets assists the modeler in the instantiation of these patterns, i.e., in the use of OntoUML as pattern-language. Thirdly, the article demonstrates how these rule sets can be materialized as a set of methodological guidelines which can be directly implemented in a tool support in the form of an automated dialogue with the novice modeler.

1 Introduction

In recent years, there has been a growing interest in the use of Ontologically Well-Founded Conceptual Modeling languages to support the domain analysis phase in Information Systems Engineering. OntoUML is an example of a conceptual modeling language whose metamodel has been designed to comply with the ontological distinctions and axiomatic theories put forth by a theoretically well-grounded Foundational Ontology [1]. This language has been successfully employed in a number of projects in several different domains, ranging from Petroleum and Gas [2] to Bioinformatics [3]. Moreover, the ontological distinctions underlying this language has been experimenting increasing adoption [4,5]. However, despite its growing popularity, OntoUML has been deemed to pose a significant complexity to novice modelers.

In order to assist these novice modelers, a number of efforts have been undertaken [6,7]. In particular, [6] proposes an automated model-checking editor that takes advantage of a well-behaved set of ontological constraints that govern the metamodel of this language. The editor constrains the users to only produce models which are consistent with these ontological constraints, i.e., if a user attempts to produce a model which does not adhere to these rules, the editor can automatically interfere in the process, identifying and explaining the model violation.

This editor, however, works in a *reactive* manner: in principle, the user can build any model using the primitives of the language; if the user attempts to build an ontologically inconsistent model, the editor prompts a proper action. In this paper, we pursue a different direction on providing tool support, i.e., we attempt to explore an *inductive* strategy in the construction of OntoUML models. As we illustrate in this paper, the ontological constraints underlying the language restrict its primitives to be combined in specific manners. To put it in a different way, in contrast with ontologically neutral languages such as UML, EER or OWL, OntoUML is a *Pattern Language*. When a model is built in OntoUML, the language induces the user to construct the resulting models via the combination of existing ontologically motivated design patterns. These patterns constitute modeling primitives of a higher granularity when compared to usual primitives of conceptual modeling such as *Class*, *Association*, *Specialization*, among others. Moreover, these higher-granularity modeling elements can only be combined to each other in a restricted set of ways. Thus, in each modeling step, the design space is reduced. The hypothesis of this work is that this strategy reduces the complexity of the modeling process for the novice modeler.

This paper, thus, presents a number of theoretical and methodological contributions aimed at assisting these novice modelers. In section 2, we briefly present the ontological foundations underlying a fragment of OntoUML. In section 3, we explore a number of design patterns which emerge from the ontological constraints underlying this language. After that, these patterns are used to derive a number of model construction rule sets. Furthermore, we illustrate how the chained execution of these rules assists the modeler in the instantiation of these patterns. Finally, we demonstrate how these rule sets can be materialized as a set of methodological guidelines which can be directly implemented in a tool support in the form of an automated dialogue with the novice modeler. Section 4 presents a brief discussion on how these ideas have been implemented in a Model-based OntoUML editor. To conclude the paper, section 5 elaborates on final considerations and directions for future work.

2 Background: OntoUML and its Underlying Ontology

OntoUML has been proposed as an extension of UML that incorporates in the UML 2.0 original metamodel a number of ontological distinctions and axioms put forth by the Unified Foundation Ontology (UFO) [1]. In this work, we focus our discussion on a small fragment of this metamodel. This fragment discusses an extension of the *Class* meta-construct in UML to capture a number of ontological distinctions among the Object Types categories proposed in UFO. In fact, for the sake of space, we limit our discussion here to an even more focused fragment of the theory, namely, those categories of Object Types which extend the ontological notion of *Sortal types* [1]. The choice for this specific fragment for this particular paper is justified by the fact that this fragment comprises the ontological notions which are believed to be the more recurrent in the practice of conceptual modeling for information systems [4,5]. Besides incorporating modeling primitives that represent these ontological distinctions, the extended OntoUML metamodel includes a number of logical constraints that govern how these primitives can be combined to form consistent conceptual models. In what follows, we briefly elaborate on the aforementioned distinctions and their related constraints for the fragment discussed in this article. For a fuller discussion and formal characterization of the language, the reader is referred to [1].

We start by making a basic distinction between categories of Object Types considering a formal meta-property named Rigidity [1]. In short, a type T is rigid iff for every instance x of that type, x is necessarily an instance of that type. In contrast, a type T' is anti-rigid iff for every instance y of T', there is always a possible world in which y is not instance of T'. In other words, it is possible for every instance y of T' to cease to be so without ceasing to exist (without losing its identity) [1]. A stereotypical example of this distinction can be found by contrasting the rigid type Person with the anti-rigid type Student.

Rigid types can be related in a chain of taxonomic relations. For instance, the rigid types Man and Person are related such that the former is a specialization of the latter. The type in the root of a chain of specializations among rigid types is termed a Kind (e.g., Person) and the remaining rigid types in this chain are named Subkinds (e.g., Man, Woman). As formally demonstrated in [1], we have the following constraints involving Kinds and Subkinds: (i) every object in a conceptual model must be an instance of exactly one Kind; (ii) as consequence, we have that for every object type T in OntoUML, T is either a Kind or it is the specialization of exactly one ultimate Kind; Subkinds of a Kind K typically appear in a construction named *Subkind Partition*. (iii) A Subkind Partition $\langle SK_1 ... SK_n \rangle$ defines an actual partition of type K , i.e., (iii.a) in every situation, every instance of SK_i is an instance of K; Moreover, (iii.b) in every situation, every instance of K is an instance of exactly one SK_i. In UML, and consequently also in OntoUML, partitions are represented by a disjoint and complete Generalization Set (see figure 1.a and 1.c).

Among the anti-rigid Sortal types, we have again two subcatetories: Phases and Roles. In both cases, we have cases of dynamic classification, i.e., the instances can move in and out of the extension of these types without any effect on their identity. However, while in the case of Phase these changes occur due to a change in the intrinsic properties of these instances, in the cases of Role, they occur due to a change in their relational properties. We contrast the types Child, Adolescent, Adult as phases of a Person with the Roles Student, Husband or Wife. In the former case, it is a change in the intrinsic property *age* of Person which causes an instance to move in and out of the extension of these phases. In contrast, a Student is a role that a Person plays when related to an Education Institution and it is the establishment (or termination) of this relation that alters the instantiation relation between an instance of Person and the type Student. Analogously, a Husband is a role played by a Person when married to (a Person playing the role of) Wife. Thus, besides being Anti-rigid, Role possesses another meta-property (absent in Phases) named Relational Dependence [1]. As a consequence, we have that the following constraints must apply to Roles: every Role in an OntoUML conceptual model must be connected to an association representing this relational dependence condition. Moreover, the association end connected to the depended type (e.g., Education Institution for the case of Student, Wife for the case of Husband) in this relation must have a minimum cardinality ≥ 1 [1].

Furthermore, as discussed in [1], Phases always occur in a so-called *Phase Partition* of a type T. For this reason, *mutatis mutandis*, constraints identical to (iii.a and iii.b) defined for Subkind Partitions are also defined for the case of Phase Partitions. However, for the case of Phase Partitions, we have an additional constraint: for every instance of type T and for every phase P_i in a Phase Partition specializing T, there is a possible world w in which x is not an instance of Pi. This implies that, in w, x is an instance of another Phase P_j in the same partition.

Finally, as formally proved in [1], rigid types cannot specialize anti-rigid types.

3. Ontological Design Patterns and Inductive Process Models

In this section, we present a number of Design Patterns which are derived from the ontological constraints underlying OntoUML as presented in the previous section. In other words, we limit ourselves here to the patterns which are related to the ontological constraints involving the three primitives previously discussed: Phases, Roles and Subkind. These patterns are depicted in figure 1 below.

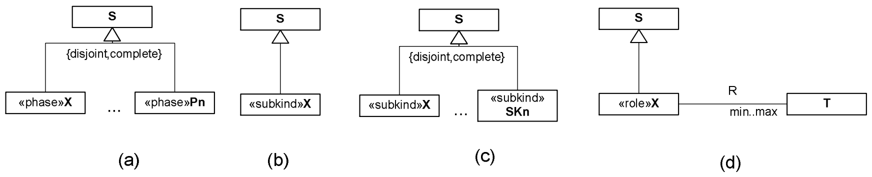


Fig.1. Design Patterns emergent from the Ontological Constraints underlying OntoUML: (a) the Phase Pattern; (b-c) the Subkind Patterns, and (d) the Role Modeling Design Pattern.

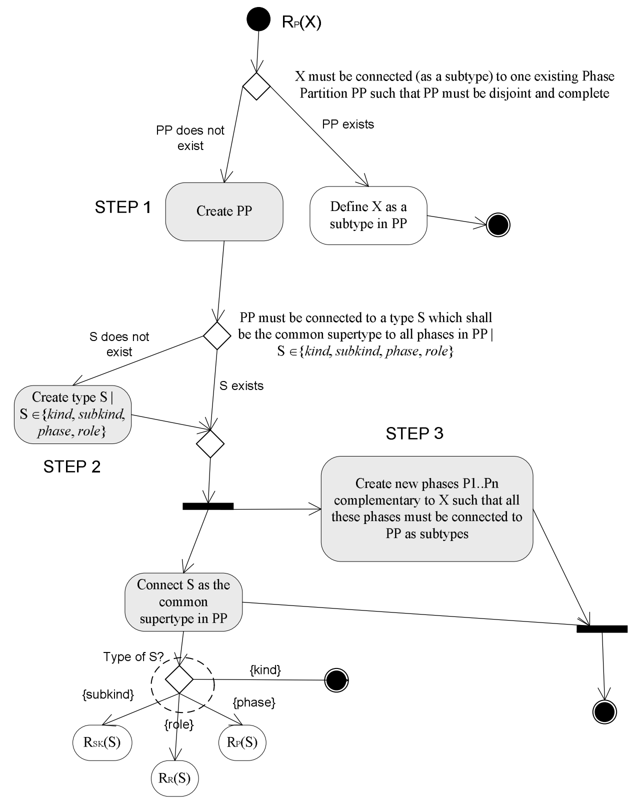
As a second objective of this section, we elaborate on a number of process models (representing inductive rule sets for model construction) which can be directly derived from these patterns. The hypothesis considered and illustrated here is the following: in each step of the modeling activity (i.e., each execution step of these process models), the solution space which characterizes the possible choices of modeling primitives to be adopted is reduced. This strategy, in turn, reduces the cognitive load of the modeler and, consequently, the complexity of model building using this language. Finally, this section demonstrates how these process models can be materialized through an interactive dialogue between the modeler and an automated tool running these rule sets. This idea is presented here via a running example and, in the following subsections, we will exemplify how the modeler may gradually build the ontology model of figure 5. For that, the design tool executes these process models and engages in dialogues with the user, guiding the development of the model from 5(1) to 5(11)

3.1 The Phase Design Pattern

Phases are always manifested as part of a Phase Partition (PP). In a PP, there is always one unique root common supertype which is necessarily a Sortal S. This pattern is depicted in figure 1.a above. By analyzing that pattern, we can describe a modeling rule set R_P which is to be executed every time a Phase P is instantiated in the model (an OntoUML class is stereotyped as phase). The rule set R_P is represented in the form of an UML activity diagram in figure 2 below. In the sequel, we exemplify the execution of this rule. In figure 2, the activities in grey represent the ones fired in this illustrative execution.

Example Execution: Let us suppose that the first type included by the user in the model is Child and that this type is stereotyped as ⟨⟨Phase⟩⟩. Thus, the rule set R_P (Child) is executed following the steps described in figure 2. Since there is no other modeling element in the model, a Phase Partition is created (step 1). Notice that, at this point, the following dialogue can be established with the user so that the supertype S of Child can be created (step 2) as well as the remaining subtypes of S complementary to Child (step 3). Each underlined term in the following dialogues can be automatically inferred in the execution of R_P:

1. **Modeling Tool:** Child is a phase for what kind of type?
2. **User:** Person
3. **Modeling Tool:** What should be the other phases of Person which are complementary to Child?
4. **User:** Teenager and Adult



RP(X)
X must be connected (as a subtype) to one existing Phase Partition PP such that PP must be disjoint and complete
PP does not exist
PP exists
STEP 1
Create PP
Define X as a subtype in PP
PP must be connected to a type S which shall be the common supertype to all phases in PP | S ∈{kind, subkind, phase, role}
S does not exist
S exists
Create type S | S ∈{kind, subkind, phase, role}
STEP 2
STEP 3
Create new phases P1..Pn complementary to X such that all these phases must be connected to PP as subtypes
Connect S as the common supertype in PP
Type of S?
{kind}
{subkind}
{phase}
{role}
RSK(S)
RP(S)
RR(S)

Fig.2. Activity Diagram representing the Inductive Rule Set RP associated to the formation of an instance of the *Phase* Design pattern

At this point, the process execution is at the decision element circumscribed by the dashed oval. Once more, a dialogue with the user can be used here to determine whether Person is a kind (in which case the process reaches a final state) or not (in which case a subsequent rule is fired in a recursive manner).

1. **Modeling Tool:** Does exist another type which is a subtype of Person?
2. **User:** No

Thus, since in this example, Person must be kind, this execution of R_P (Child) reaches a final state. In figure 5, we illustrate the user's model after his choice of stereotyping Child as a Phase (figure 5.1) and the resulting design pattern which is configured by the execution of R_P(Child) (figure 5.2).

3.2 The Subkind Design Pattern

Subkinds can be manifested in two different manners: (i) as a specialization of a type S; (ii) as a part of a *generalization set* which has as a common superclass a type S. These two patterns are illustrated in figure 1.b and 1.c, respectively. The inductive formation rule set associated with the stereotype *subkind* is captured in figure 3. In the sequel, we exemplify the execution of this rule set.

Example Execution: Let us suppose that the next type included in the model is the type School which (in this example) is stereotyped as a subkind. The rule set R_{SK}(School) is then fired. The following dialogue can be used to discover whether School is a subkind of a type S already existent in the model. Since in this case, the answer is negative, the dialogue proceeds such that the proper supertype of School can be created (Step 1). Furthermore, after connecting Organization as a supertype of School, the tool dialogue attempts to uncover whether there are additional subkinds of Organization complementary to School forming a generalization set (DP1):

1. **Modeling Tool:** Is School a subkind of Person[1]?
2. **User:** No
3. **Modeling Tool:** Of which type is School a subkind then?
4. **User:** Organization
5. **Modeling Tool:** Is there another subkind of Organization complementary to School?
6. **User:** No

Finally, notice that in parallel (DP2), the tool attempts to discover whether Organization is itself a kind or a subkind (in which case the rule set R_{SK} (Organization)) should be recursively fired). Since, in this example, the former case obtains, then this execution of rule R_{SK} terminates. In fig. 5, we illustrate the user's model after his choice of stereotyping School as a subkind (fig. 5.3) and the resulting design pattern which is configured by the execution or R_{SK}(School) (fig. 5.4).

[1]If multiple kinds and subkinds already exist in a given model presenting themselves as alternatives for the supertype S, then the modeling tool's interface should present the user with such alternatives accordingly.

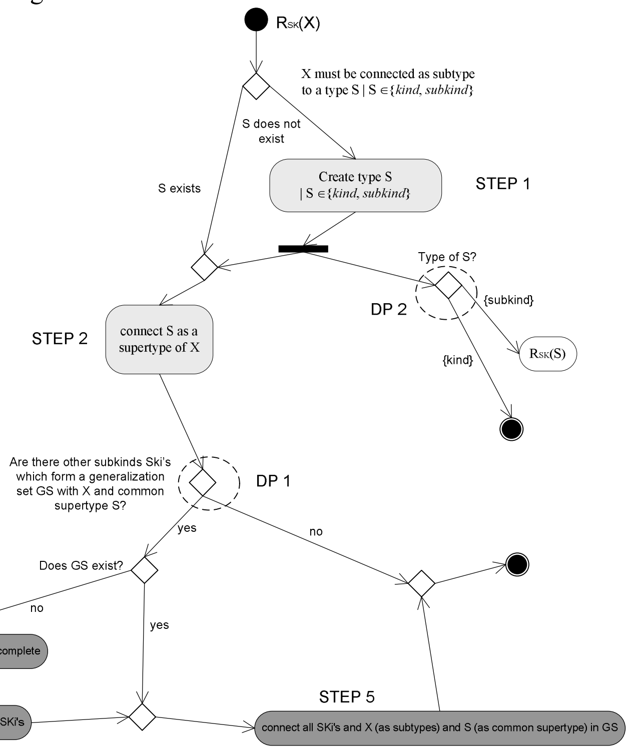
We can offer a second example execution which supposes that the next element included in the model of figure 5.5 is a type Woman, also stereotyped as subkind. The rule R_{SK}(Woman) is then fired and the step 1 of figure 4 is executed:

1. **Modeling Tool:** Is Woman a subkind of Person or Organization?
2. **User:** Person

In this case, the process goes straight to Step 2 and then once more to the decision point DP1. Since there is no other subkind of Person already in this model, the dialogue with the user is verbalized in the following manner:

1. **Modeling Tool:** Is there another subkind of Person complementary to Woman?
2. **User:** Man

We then have the execution of steps 3-5 in figure 3, resulting in the model of figure 5.6. Activities which are executed only in this second example execution of R_{SK}, are depicted in dark grey in figure 3.



RSK(X)
X must be connected as subtype to a type S | S ∈{kind, subkind}
S does not exist
S exists
Create type S | S ∈{kind, subkind}
STEP 1
Type of S?
{subkind}
DP 2
RSK(S)
{kind}
STEP 2
connect S as a supertype of X
Are there other subkinds Ski's which form a generalization set GS with X and common supertype S?
DP 1
yes
no
Does GS exist?
STEP 3
no
yes
create generalization set GS disjoint and complete
STEP 4
create and/or select all the appropriate SKi's
STEP 5
connect all SKi's and X (as subtypes) and S (as common supertype) in GS

Fig.3. Activity Diagram representing the Inductive Rule Set RP associated to the formation of instances of the *Subkind* Design patterns

3.3 The Role Design Pattern

Roles represent (possibly successive) specializations of a Kind by using a relational specialization condition. Figure 4 captures the modeling rules that induces the creation of the role design pattern depicted in figure 1.d. In the sequel, we present the dialogue the tool may engage with the reader, following this rule set.

Example Execution: Let us suppose that the next type included in the model of figure 5.7 is the type Student which (in this example) is stereotyped as a role. The rule R_R(Student) is then fired. The following dialogue can be used to discover whether Student is a subkind of a type S already existent in the model. Since in this case the answer is affirmative for Person, the dialogue proceeds to discover whether Student is a direct subtype of Person or of one of its subtypes (Child, Adolescent, Adult, Man, Woman). Since Student is a direct subtype of Person, step 1 in figure 4 is executed. Moreover, in decision point DP 1, a dialogue is established to discover if the type Student is *relationally dependent of* is already present in the model (line 5 in the dialogue below). Afterwards, in decision point 2 (DP2), since there is no relation connected to student, the tool enquires the user regarding the relation R representing the relational specialization condition for Student (line 7). After that, the *studies in* relation is created (step 2) and its cardinality constraints are assessed (step 3, see also lines 9-16 below). Finally, the *studies in* relation is connected to both Student and School. With the execution of R_R(Student) we then obtain the model of figure 5.8.

1. **Modeling Tool:** Is Student a subtype of Person or Organization?
2. **User:** Person
3. **Modeling Tool:** Is it a direct subtype of Person?
4. **User:** Yes
5. **Modeling Tool:** In relation to which type of entity does a Person plays the role of Student? School, Organization, Child, Adolescent, Adult, Man or Woman?
6. **User:** School
7. **Modeling Tool:** How would you name this relation between Student and School?
8. **User:** studies in
9. **Modeling Tool:** A Student is related to a minimum of how many different School(s) via the *studies in* relation?
10. **User:** 1
11. **Modeling Tool:** Is there a maximum?
12. **User:** No
13. **Modeling Tool:** A School is related to a minimum of how many different Student(s) via the *studies in* relation?
14. **User:** Zero
15. **Modeling Tool:** Is there a maximum?
16. **User:** No

Once more, we can offer a second example execution for the Role design pattern and its associated rule. This example supposes the next element included in the model of figure 5.9 to be the type Husband also stereotyped as Role. The rule R_R(Husband) is then fired and step 1 is executed. However, this time, as a subsequent step of decision point DP 1, step 5 of the process is executed instead (activity marked in dark grey in figure 5). This is due to the fact that (following lines 7 and 8 below), the type T to which Husband should be connected is absent in the model.

1. **Modeling Tool:** Is Husband a subtype of Person or Organization?

2. **User:** Person
3. **Modeling Tool:** Is it a direct subtype of <u>Person</u>?
4. **User:** No
5. **Modeling Tool:** Is Husband a direct subtype of <u>Man</u>, <u>Woman</u>, <u>Child</u>, <u>Teenager</u>, or <u>Adult</u>?
6. **User:** Man
7. **Modeling Tool:** In relation to which type of entity does a <u>Person</u> plays the role of <u>Husband</u>? <u>School</u>, <u>Organization</u>, <u>Child</u>, <u>Adolescent</u>, <u>Adult</u>, <u>Man</u> or <u>Student</u>?
8. **User:** None of these. The correct type is Wife

After this, steps 2-4 of figure 4 are executed and the relation *married to* between Husband and Wife is properly instantiated. The dialogue corresponding to these steps is omitted below for the sake of space economy. Finally, the recursive execution of rule R_R(Wife) is triggered (DP 3). This time the following steps of the process in figure 4 are executed: (i) Step 1 – in which the specialization between Wife and Woman is created; (ii) DP 1 – at this point since type T already exists (Husband), execution goes straight to DP 2; (ii) DP 2 - at this point since relation R already exists (*married to*), step 4 is executed and the process terminates. As a result, the final model of figure 5.10 is obtained.

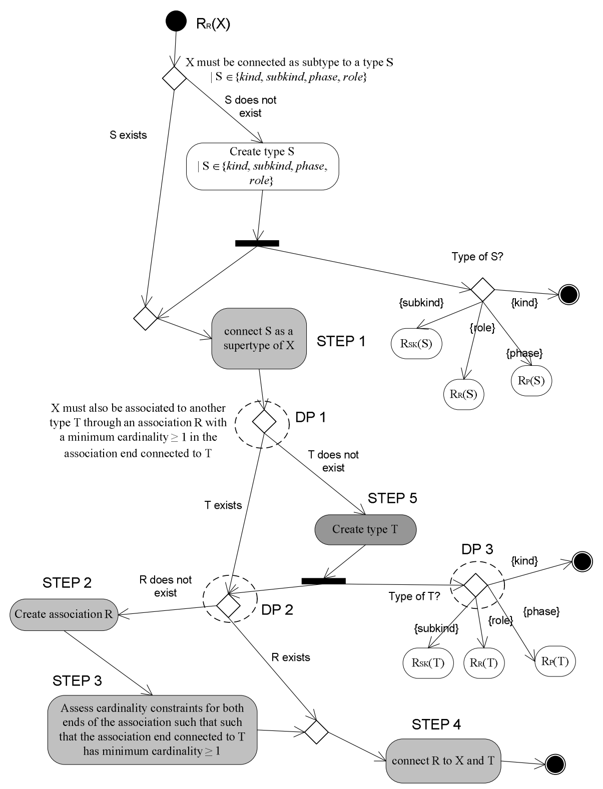

R_R(X)
X must be connected as subtype to a type S | S ∈ {kind, subkind, phase, role}
S does not exist
S exists
Create type S | S ∈ {kind, subkind, phase, role}
Type of S?
{subkind}
{kind}
{role}
{phase}
R_SK(S)
R_R(S)
R_P(S)
connect S as a supertype of X
STEP 1
X must also be associated to another type T through an association R with a minimum cardinality ≥ 1 in the association end connected to T
DP 1
T does not exist
T exists
STEP 5
Create type T
DP 3
{kind}
Type of T?
{subkind}
{role}
{phase}
R_SK(T)
R_R(T)
R_P(T)
STEP 2
R does not exist
Create association R
DP 2
R exists
STEP 3
Assess cardinality constraints for both ends of the association such that such that the association end connected to T has minimum cardinality ≥ 1
STEP 4
connect R to X and T

Fig.4. Activity Diagram representing the Inductive Rule R_R associated to the formation of an instance of the *Role* Design pattern in OntoUML

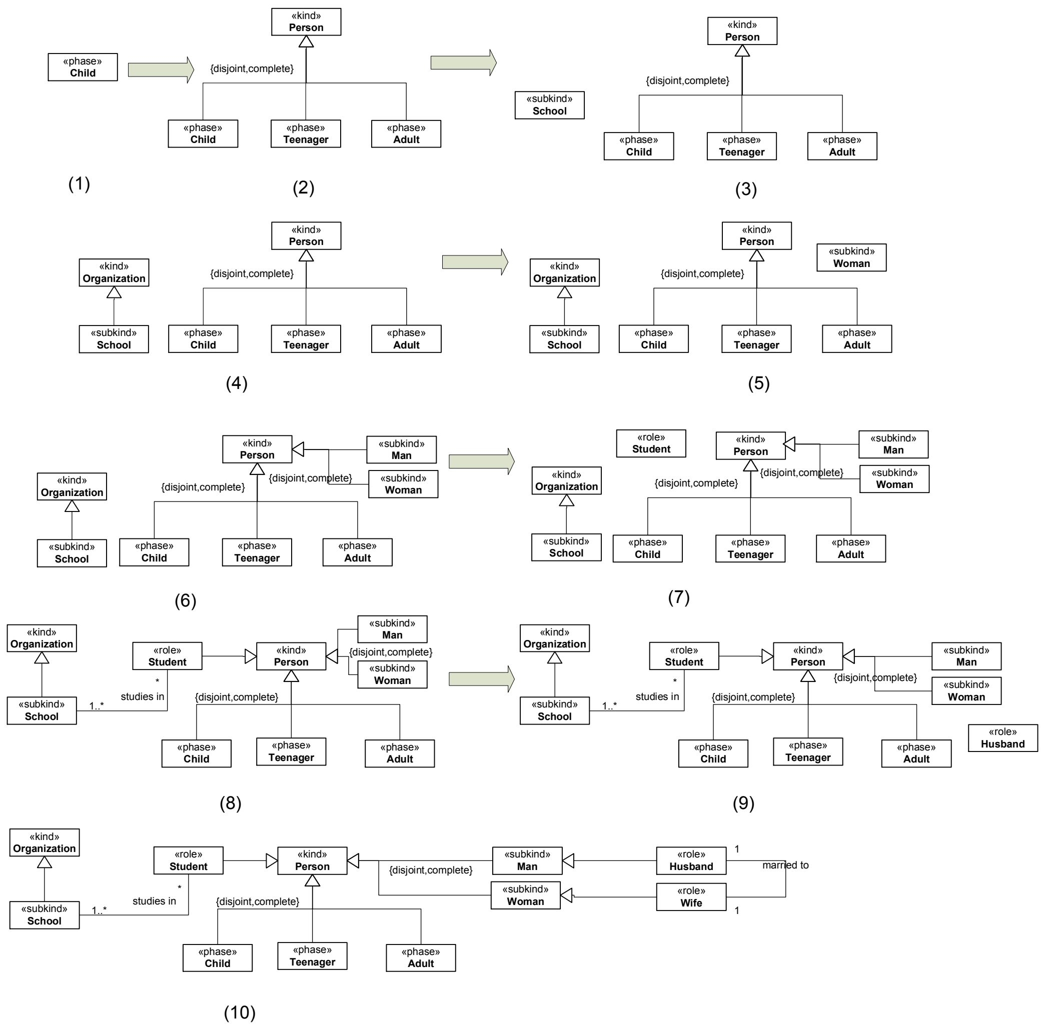
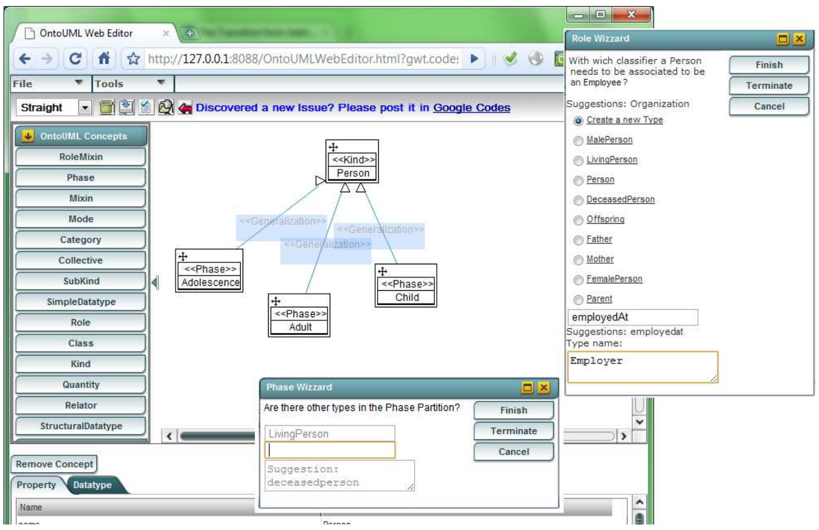


Fig.5. Valid OntoUML model inductively constructed via the chained execution of Rule Sets

4 Tool Support

This work applies some design patterns and its associated rule sets to create an interactive dialogue with the modeler to support model development in a kind of Model-Based Web Wizard for novice modelers. The architecture of this editor has been conceived to follow a Model Driven Approach. In particular, we have adopted the OMG MOF (Meta-Object Facility) metamodeling architecture with an explicit representation of the OntoUML metamodel. The rule sets presented here are implemented in a business logic layer which explicitly manipulates this metamodel. The interactive dialog with the modeler (which occurs via the execution of these rule sets) is implemented using a graphical interface as illustrated in figure 6 below.



With wich classifier a Person needs to be associated to be an Employee ?

Suggestions: Organization

- [✓ Create a new Type]
- MalePerson
- LivingPerson
- Person
- DeceasedPerson
- Offspring
- Father
- Mother
- FemalePerson
- Parent

employedAt

Suggestions: employedat

Type name:

Employer

Are there other types in the Phase Partition?

LivingPerson

Suggestion:
deceasedperson

Fig.6. Screeshots for the Web Implementation of an OntoUML editor using Design Patterns and Inductive Rule Sets

5 Final Considerations and Future Work

The work presented in this paper lies in the identification of Ontological Design Patterns underlying the OntoUML language. In [8], three major categories of Design Patterns for Ontology-Driven Conceptual Modeling are discussed, namely, *Modeling Patterns*, *Analysis Patterns* and *Transformation Patterns*. Design Patterns which are most frequently found in the Ontology Engineering literature fall in the third category. An example is the catalog of patterns proposed in [9] which aims at helping the user in codifying common modeling structures in the Semantic Web languages. Analysis Patterns in the sense mentioned in [8] differs from the most frequent use of the term in Conceptual Modeling [10]. While the latter refers to conceptual structures which are recurrent in different domain models, the former refers to patterns derived from ontological theories which can be use to analyze properties of conceptual models. The Analysis Patterns in the sense of [10] would actually be akin to the ones explored in this article, which fall in the category of Modeling Patterns [8]. However, while the former captures domain-level recurrent conceptual structures, the latter represents domain-independent modeling solutions originated from ontological theories.

In future works, the theoretical contributions presented here shall be expanded by exploring a complete set of patterns which contemplate the complete set of modeling primitives of OntoUML. Moreover, we intend to conduct some empirical research initiatives to verify the hypothesis underlying this article, namely: (i) that the inductive methodological strategy pursued here in fact contributes to diminish the complexity of the process of ontology-driven conceptual modeling; (ii) that it positively contributes to increase the quality of the resulting models when produced by novice modelers.

Our research agenda for the future also includes the investigation of how the tool can be augmented in terms of providing extra cognitive support [11] for conceptual model development using OntoUML. This includes issues related to the scalability of the models using the present tool. Suppose, for instance, that the model under development already has a hundred *kinds*. It would be quite cumbersome if the tool offered all of them as options to generalize a new *role* being created (see subsection 3.3). It is wise to consider, for instance, techniques for modularizing the model, so that the tool could reduce this set of possible *kinds*. Other visualization and navigation techniques could also be applied to provide further cognitive support on model creation [12].

The idea is also to advance with this work to enable the reuse of knowledge that was previously only tacit in the mind of the user when conceiving a model. In other words, we intend to provide support to capture the *design rationale* behind the development of a conceptual model in order to reuse these previous insights as hints for upcoming modelers, who can build new models on the basis of such knowledge. Particularly related to knowledge reuse, we could explore some intelligent support (e.g., recommendation techniques) to enable search within the tool knowledge base (i.e. previously designed models) as well as in external sources (e.g. other conceptual model repositories or even other types of general knowledge sources), seeking to find new contributions to model particular domains. Regarding the use of external sources, formal ontology excerpts could be adopted as fragments of a new conceptual model, based on similarity of concepts and concept patterns. And besides, the use of results extracted from unstructured content could be applied as input to suggest specific terms applied in connection to the concepts which are already part of the model under development.

Acknowledgments. This research has been partially supported by FAPES (Grant #45444080/09) and CNPq (Grant #481906/2009-6 and Productivity Grant #309382/2008).

References

1. Guizzardi, G. Ontological Foundations for Structural Conceptual Models, Universal Press, Holanda, 2005. ISBN 1381-3617.
2. Guizzardi, G.; Lopes, M.; Baião, F.; Falbo, R. On the importance of truly ontological representation languages, IJISMD, 2010. ISSN: 1947-8186.
3. Gonçalves, B. ; Guizzardi, G. ; Pereira Filho, J. G. . Using an ECG reference ontology for semantic interoperability of ECG data. Journal of Biomedical Informatics, v. 43, 2010.
4. Bauman, B. T., Prying Apart Semantics and Implementation: Generating XML Schemata directly from ontologically sound conceptual models, Balisage Markup Conf., 2009.
5. Halpin, T., Morgan, T., Information Modeling and Relational Dababases, Morgan Kaufman, 2008. ISBN 1558606726
6. Benevides, A.B.; Guizzardi, G. A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML, LNBPI, Vol. 24, 2009. ISSN 1865-1356.
7. Castro, L. et al., A Linguistic Approach to Conceptual Modeling with Semantic Types and OntoUML, Proceedings of the IEEE 5th VORTE/MOST 2010, Vitoria, Brazil, 2010.
8. Guizzardi, G., Theoretical Foundations and Engineering Tools for Building Ontologies as Reference Conceptual Models, Semantic Web Journal, 1:1, IOS Press, 2010.
9. Ontology Design Patterns, online: http://ontologydesignpatterns.org/wiki/Main_Page.
10. Fowler, M., Analysis Patterns: Reusable Object Models, Addison-Wesley, 1996.
11. Walenstein, A.. Cognitive support in software engineering tools: A distributed cognition framework. PhD thesis, Simon Fraser University, Vancouver, BC, 2002.
12. Ernst, N.A.; Storey, M.; Allen, P. Cognitive support for ontology modeling. International Journal of Human-Computer Studies, 62:5 (May 2005), ISSN: 1071-5819. pp. 553-577.