STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

CONCEPTUAL MODELING EVOLUTION. A FORMAL APPROACH

MARIA-CAMELIA CHISĂLIŢĂ-CREŢU

Abstract. The aim of this paper is to investigate the possible types of conceptual variability and to propose shifting strategies, like refactoring, forward conceptual abstraction, and conceptual specialization to switch between conceptual models. A biological evolution-based model is proposed to describe the changes within the structure of the studied models. The transformations that highlight the conceptual modeling variability in ontogenic and phylogenic processes are formalized.

1. Introduction

Software systems continually change as they evolve to reflect new requirements, but their internal structure tends to decay. Refactoring is a commonly accepted technique to improve the structure of object oriented software. Its aim is to reverse the decaying process in software quality by applying a series of small and behaviour-preserving transformations, each improving a certain aspect of the system [9]. The *variability* [19] in the context of conceptual modeling means the possibility to build distinct and still correct conceptual models for the same set of requirements from the real world (Universe of Discourse, UoD) problems. Such conceptual model is called *variant*. A non-exhaustive framework of three types of variability was proposed, based on a literature survey and empirical evidence [19].

Within conceptual modeling variability, refactoring has proved to be a feasible technique to switch between variants. In order to emphasize refactoring transformations, the artifact may be represented as graph [14]. As refactoring was initially applied at the implementation level, conceptual models became a potential target for application of refactorings.

Research within conceptual modeling reveals the possibility to integrate the refactoring process in the analysis development phase too. A biological

Received by the editors: March 20, 2011.
2010 *Mathematics Subject Classification.* 68N30, 92B10.
1998 *CR Categories and Descriptors.* D.2.1 [**Reusable Software**]: – *Reusable Software*; I.6.5 [**Model Development**]: – *Modeling Methodologies*.
Key words and phrases. conceptual modeling, refactoring, biological evolution.

evolution model is proposed in order to cope with different types of variability that were identified. Specific refactorings are suggested to shift between *ontogenic* conceptual models, while forward conceptual abstraction and conceptual specialization are advanced to achieve *phylogenic* conceptual models.

The rest of the paper is organized as follows. Section 2 investigates the three types of conceptual modeling variability identified in the literature. Three types of transformations are advanced in order to cope with the variability among variants. A biological evolution-based model is proposed in Section 3, while Section 3.2 formalizes the identified transformations within the three types of conceptual modeling variability studied as ontogenic and phylogenic processes. Section 4 contains some conclusions of the paper and suggestions of future work.

2. Conceptual Modeling Variability Types

A non-exhaustive framework of three types of variability was proposed, based on a literature survey and empirical evidence: construct, vertical abstraction, and horizontal abstraction variability [19].

2.1. **Construct Variability.**

Definition 2.1. *Construct Variability ([19])*

Construct variability represents the possibility of modeling concepts in the UoD using different constructs in the same modeling language.

Within construct variability, the concepts within UoD have the same semantics in all variants. They are represented by a class (entity), an attribute, a relationship.

2.1.1. *Types of Equivalent Construct Variants.* There are many types of construct variability. In [3] the possible refactorings that may be applied to the conceptual modeling variability are studied. In Figure 1 a frequently case of construct variability within object-oriented analysis and design is presented. The `Price` concept may be both modeled as a class (*cmA* variant) with a single attribute `amount` or as an attribute of type integer (*cmB* variant). The semantic definition for the `price` and the `Product` are identical in both variants, though different language constructs to represent it are used, i.e., an attribute (*cmB* variant) instead of a concept (*cmA* variant).

Another type of construct variability is similar to normalization of a database definition, by removing all redundant data elements from the class definitions. Figure 2 emphasizes such a normalization in the *cmA* variant, where the product `value` aspect is modeled as an attribute. In the *cmB* variant, the product `value` is modeled as a method, which multiplies the *quantity* by the *price* to obtain the correct `value` in Figure 2. A motivation to consider the

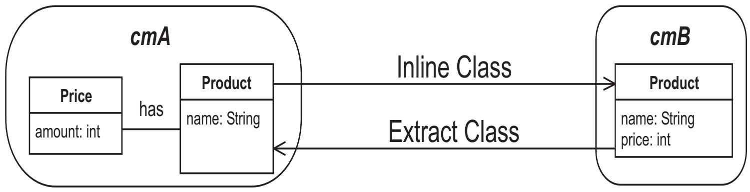


FIGURE 1. Construct variability for the concept `Price`. *cmA* variant is an entity-based model; *cmB* variant is an attribute-based model.

value a method is because it represents a calculation done predominantly at design level. Within the analysis phase, the semantic definition of the *value* is given by the same formula within different conceptual models.

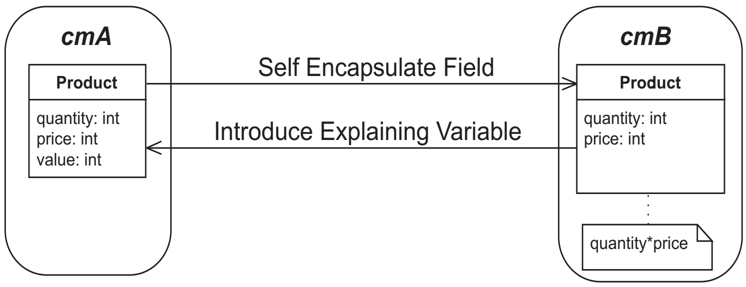


FIGURE 2. Construct variability for the attribute `value`. *cmA* variant is an attribute-based model; *cmB* variant is a method-based model.

A third type of construct variability is represented by the multiple types definition. It consists of introducing or removing specific codes for all the types that are a direct specialization of a generic type. Figure 3 shows a general `Product` that is refined to another ones, more specific: `BOY` and `GIRL`. The *cmA* variant defines specific codes for each concrete product as for *boy* or *girl*, adding a short description for the derived types. On the other hand, the *cmB* variant creates different classes for *boy* or *girl* `products`, providing flexibility for improvements directed to a specific type. Therefore, the definition for the specialized types has not the same semantics for both variants.

2.1.2. *Suggested Refactorings.* In order to switch between variants, there are several types of refactorings that may be applied.

Refactoring a class (entity) to a set of attributes (properties)

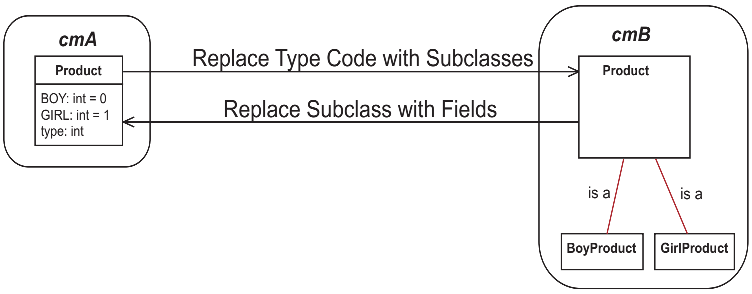


FIGURE 3. Construct variability for multiple types. *cmA* variant is an attribute-based model; *cmB* variant is a entity-based model.

The first recommended transformation is the *InlineClass* refactoring [9]. It is used as a weight reducer since it diminishes the number of classes (entities) by redistributing the responsabilities among the remaining classes (entities). The aspect of moving the attributes and methods to other classes is realized through refactorings like *MoveMethod* or *MoveField*. Figure 1 illustrates how the `price` attribute in the `Product` class is replaced by the attribute contained within the `Price` class, i.e., the `amount` attribute. In order to reverse the effect of the *InlineClass* refactoring, the *ExtractClass* refactoring may be applied to switch between variants. As an immediate effect of the latter application is a raise of the abstraction level, while the overall weight of the UoD increases due to the new class (entity) addition.

Refactoring an attribute to a method

There are many cases where an attribute may be computed from other existing attributes, while the attribute continues to exist. This results in redundancy, a particular case of bad smell [9]. The initialization of the attribute may consist of a formula of which the interpretation gives the correct value of the attribute. Subsequently, the attribute is updated by the corresponding formula. Thus, it may be extracted to a method and the calls to this method will replace the access to the redundant attribute. The latter one being no longer referenced, will be removed. In order to remove this redundancy problem, the *SelfEncapsulateField* refactoring is suggested by Figure 2, that will replace all accesses and updates to the attribute with calls to a newly introduced accessor (*getter*) and modifier (*setter*) method. The *IntroduceExplainingVariable* refactoring allows to return to initial variant by adding an attribute that will be initialized with the corresponding expression.

Refactoring type codes to a set of derived classes

Type codes may be used to model different specializations of the same class. This situation is usually indicated by the presence of `case`-like conditional statements, i.e., `switch`, `if-then-else` constructs. They test the value of the type code and then execute the appropriate code, depending on the value of the type code. A variant to this conceptual model is to expand the class into a class hierarchy in order to emphasize specific types of the base class.

Different refactorings are recommended to be used subsequently. Conditionals that affect the behavior need to be transformed by the *ReplaceConditionalwithPolymorphism* refactoring, that allows to use polymorphism to handle the different behavior in the inherited classes. In order to switch the type codes within a context where the behaviour is not affected the *ReplaceTypeCodewithSubclass* refactoring may be applied. In each cases, the type codes will be replaced with a subclass for each distinct one. Furthermore, there are cases where some features that are relevant only to objects with certain type codes.

Creating such a class hierarchy through this refactoring, then the *PushDownMethod* and the *PushDownField* refactorings may be applied to clarify to which subclass these features are relevant. An important advantage of this switch between variants is the possibility to move the particular behaviour from a client of a class to the class itself. This refactoring ensures a large flexibility of the variant within a continuous changing UoD, through polymorphism.

The reverse process allows to transform subclasses into attributes within a single class, following the *ReplaceSubclasswithFields* refactoring. Figure 3 depicts the way the type codes are changed by the appropriate refactorings. This refactoring situation represents a special case of forward conceptual abstraction. There are cases where the type code may or may not affect the behavior.

2.1.3. *Core Ideas.* The three representative examples demonstrate that refactoring between construct variants is feasible. The effort required to switch between the variants is reduced by the application of a limited number of small refactorings.

There are few, rather limited differences between the variants within the construct variability. Though, the literature retained some work that suggests the evolvability aspect of conceptual models within this type of variability. This would be the case of the third example discussed here, where the flexibility to improvements of the *cmA* variant is reduced. In [1] it is claimed that an entity should be preferred over an attribute if it is likely that the modeled concept in the UoD will take benefit of additional properties in the future.

Though, this claim suggests that is useful to be able to switch between these two variants.

This type of variability is exploited in the shift from object-oriented analysis to design. As a consequence, it is expected that *construct variability* had been already used refactoring in current modeling activities. Furthermore, other relevant results in refactoring conceptual models show that is unlikely that hard obstacles for this transformations between construct variants will be found [6, 18, 7].

2.2. **Vertical Abstraction Variability.**

Definition 2.2. *Vertical Abstraction Variability ([19])*

Vertical abstraction variability refers to the possibility of modeling concepts in the UoD in a more or less generic (abstract) way.

2.2.1. *Types of Vertical Abstract Variants.* There are two ways to navigate over the vertical abstraction variability. The first one refers to the possibility to switch from a general conceptual model to a concrete model, while the other one increases the abstraction level by removing concrete aspects, or by adding various parameters. In [2] refactoring categories needed to switch between models are identified and described.

An example of an abstract vertical variability that may be navigated in both ways, from a generic to a specific conceptual model and vice versa is presented for the `Loan` concept. It cannot be considered like in the construct variability, because its definitions are different within studied variants. The *cmA* variant illustrated by Figure 4 consists of a concrete conceptual model, where a `Loan` is associated with the `Client` to whom it was given to.

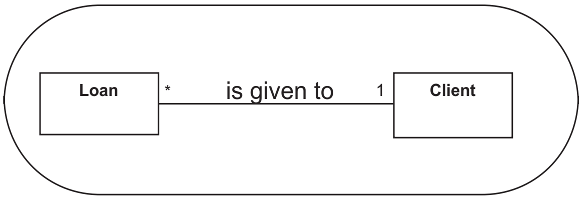


FIGURE 4. *cmA* variant for the concept `Loan`: a *concrete model* within a vertical abstraction.

In the Figure 5, the *cmB* variant defines the `Loan` given to a `Client` when a specific `Action` is achieved, e.g., the client meets some eligibility criteria.

Figure 6 depicts the *cmC* variant, where the `Loan` is given to `Client`, that may be an `Institution` or a `Person`. Moreover, the `Loan` has a type and it is given to the `Client` when some `Action` is fulfilled.

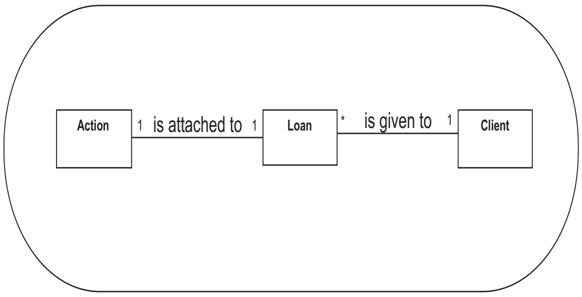


FIGURE 5. *cmB* variant for the concept `Loan`: a *general model* within a vertical abstraction.

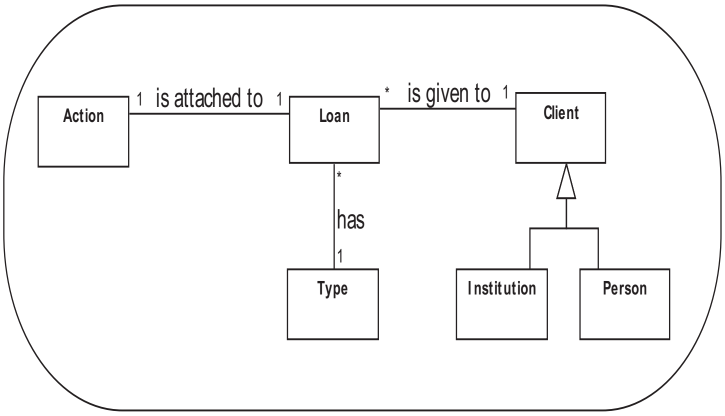


FIGURE 6. *cmC* variant for the concept `Loan`: a *very general model* within a vertical abstraction.

The three variants may be navigated from the most concrete, to the most general and conversely. The *cmC* variant is the most general model, where the `Loan` definition is available for a wide number of situations compared to the first two variants.

2.2.2. *Suggested Refactorings.* In order to switch between the presented variants, the two ways to navigate over the vertical abstraction and instantiation variability were studied and appropriate solutions were provided.

Transforming to a more generic variant

For the `Loan` concept, switching from the most concrete variant (*cmA* variant) to the most general (*cmC* variant) means to increase its flexibility degree. First, by introducing some `Action` concept that allows to offer a `Loan` to a `Client` (*cmB* variant). The abstraction level is increased by switching from the *cmB* variant to the *cmC* variant, by adding a `Type` to the `Loan` and differentiating a `Client` as `Institution` or `Person`. In order to do that, a more detailed analysis is needed. The forward conceptual abstraction follows to discover new concepts and new associations that make the model more flexible. The refactoring techniques that may be applied are not immediately identified.

Transforming to a more specific variant

In order to obtain the *cmA* variant starting from the *cmC* variant, the flexibility level of the more generic variant has to be reduced by removing all unrequired concepts, associations, and attributes. The refactorings that can be applied in this situation are those specific to inheritance or generality category [9]. They may consists of refactorings like: *PullUpField, PullUpMethod, PushDownMethod, PushDownField, ExtractSubclass, ExtractSuperclass,* or *ExtractInterface*. It is important to underline that from this large refactoring category only the refactorings that work on concepts, associations, or attributes will be applied. This transformations allow to switch from a flexible model to a thin, clear, concrete model, by removing the superfluous information.

2.2.3. *Core Ideas.* The software *genericity* is defined by Parnas [16] as the possibility to use it "without change, in a variety of situations". In [11] *abstraction* is defined as "a view of an object that focuses on the information relevant to a particular focus and ignores the remainder of the information".

In order to identify and implement new concepts and specialized behaviour that transform the concrete models to more generic ones, the forward conceptual abstraction process is needed. Refactoring techniques have limited usage in this navigation way of the vertical abstraction variability. The new variants have the advantage of a raised adaptability and flexibility.

The shift from a more generic to a more specific conceptual model consists of applying refactoring techniques that remove the redundant modeling elements and achieve a lighter variant of the initial model. This type of variability was observed in the process of simplifying the design of the over engineered systems [16, 10, 8, 6]. Changes to the system are made more easily if the conceptual model is more general and consequently, more difficult if the conceptual model is too simple or too concrete.

2.3. Horizontal Abstraction Variability.

Definition 2.3. *Horizontal Abstraction Variability ([19])*

Horizontal abstraction variability refers to the possibility of modeling concepts in the UoD based on different properties.

2.3.1. *Types of Horizontal Abstract Variants.* The horizontal abstraction is emphasized within a particular UoD that contains the concepts of an academic management system, like students and teachers. A *student* has a specialty that follows, while a *teacher* has a didactic position. Each of them has a certain *civil status*. The solutions proposed to achieve horizontal variability are presented in [4].

A first direction within the research is represented by the one depicted in Figure 7 where the *person type*, i.e., *student* or *teacher*, is emphasized. The possibility to make visible the person types means to add it as a *primary dimension* [19], that allows to isolate all instances of a certain type of person. In the *cmA* variant, the person type instances are separated in two categories, defined as the `Student` or `Teacher` concepts. `Civil Status` property is shared by both `Student` and `Teacher` concepts. In [19] such a property forms a *secondary dimension* in the concept modeling. Its instances, like *married person* or *single person* are scattered (made not visible) over all instances of the `Student` and `Teacher` types.

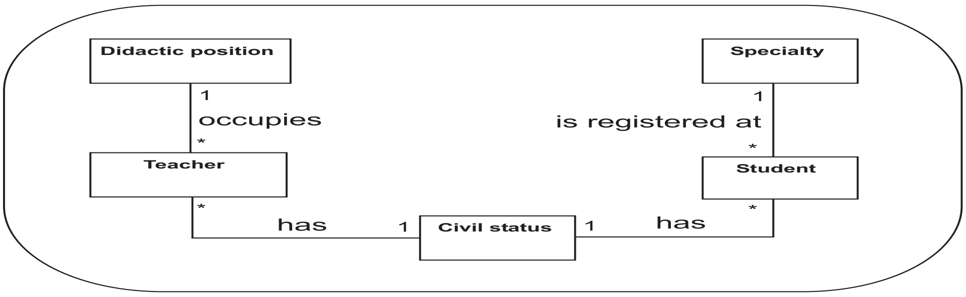


FIGURE 7. *cmA* variant : The `Person` types form a primary dimension, through the `Student` and `Teacher` concepts.

The second approach consists of highlighting the `Civil Status` property by isolating it as a primary dimension. Therefore, instances of persons are divided in two categories: `Married person` and `Single person`. The person type (*student* or *teacher*) remains as a secondary dimension, being spread over all instances of type `Married Person` and `Single Person`. Figure 8 presents the *cmB* variant where the person type is not visible, but shared between different instances of `Civil Status` types.

In order to shift between the two variants an intermediate *cmC* variant, presented by Figure 9, needs to be build. Therefore, the primary dimension of the person types from the first approach and the primary dimension of the

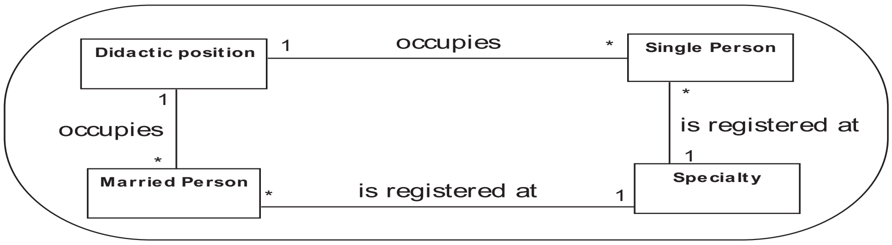


FIGURE 8. *cmB* variant : The `Civil Status` types form a primary dimension, through the `Married Person` and `Single Person` concepts.

civil status types from the second research direction are used. This means that both are isolated and visible.

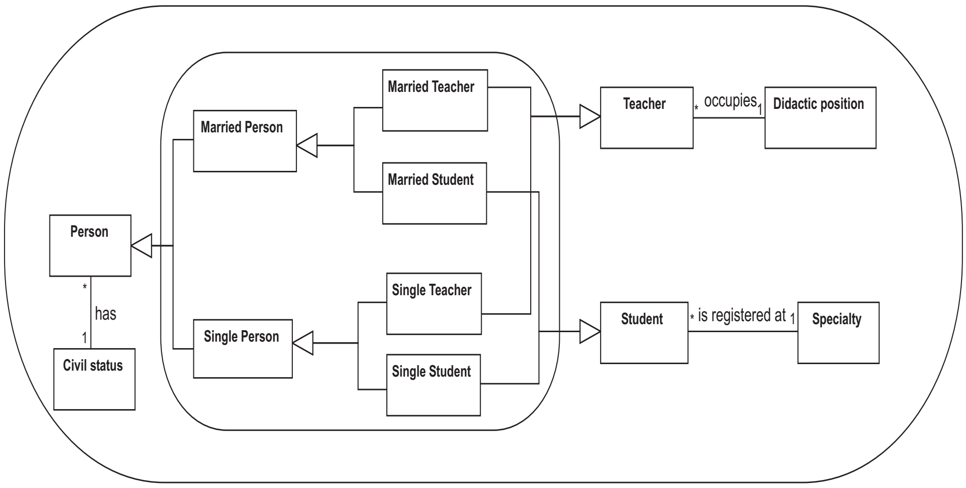


FIGURE 9. *cmC* variant : Both `Person` and `Civil Status` types are primary dimensions.

2.3.2. *Suggested Refactorings.* There are situations where someone may want to switch between variants developed within horizontal abstraction variability. The shift between the *cmA* variant and the *cmB* variant may be done using the *cmC* variant as an intermediate variant. In order to achieve it a two phases process has to be implemented:

- *Step 1:* establish an equivalency relationship between the two dimensions, by transforming the *cmA* variant to the *cmC* variant;

- *Step 2:* allow to keep the requested primary dimension only.

Refactoring to equivalent dimensions (*Step 1*)

In order to accomplish the equivalency between the *cmA* variant and the *cmC* variant, each entity type from the primary dimension of the *cmA* variant will receive the specialization from the primary dimension of the *cmB* variant. This means that a `Student` will become `Married Student` or `Single Student`, while a `Teacher` will be `Married Teacher` or `Single Teacher`. The total number of entity types is computed as cartesian product between the **n**umber **o**f entity types for the **P**rimary **D**imension (**noPD**) within the *cmA* variant (*student*, *teacher*) $noPD_{cmA} = 2$, and the *cmB* variant (*married*, *single*) $noPD_{cmB} = 2$. Therefore, as it is shown in Figure 9 the number of specialized types identified within the *cmC* variant is $noPD_{cmC} = noPD_{cmA} \times noPD_{cmB} = 2 \times 2 = 4$. Transformations required to switch between the *cmA* variant and the *cmC* variant include the following aspects:

(1) *add new classes to specialize.* For each entity type of the primary dimension within the *cmA* variant, i.e., `Student` and `Teacher`, new subclasses that emphasize the primary dimension within the *cmB* variant, i.e., `Married` and `Single`, are added. This means the new variant will be enhanced with the following classes (entities): `Married Student`, `Single Student`, `Married Teacher`, and `Single Teacher`.

(2) *responsibility reassignment.* Specialization for the new added classes (entities) is made by moving or pushing down data (and behaviour) from the initial primary dimension to the new subclasses, using refactorings like: *PushDownMethod* and *PushDownField* [9]. This new variant introduces redundancies, like where a `Single Student` has already a typed property `Civil Status` as `Single`.

(3) *add new class to generalize.* Generalization for the new created subclasses is added from the primary dimension of the *cmB* variant.

Refactoring to a primary dimension (*Step 2*)

To obtain the simplified *cmB* variant starting from the *cmC* variant, the transformations previously applied have to be semantically reversed. This means that primary dimension from the *cmB* variant has to be emphasized by collapsing the existing subclasses in the *cmC* variant. Thus, the entity types related to a `Person` (`Student` and `Teacher`) will be spread within the model through the new primary dimension `Civil Status` (`Married` and `Single`) of the *cmB* variant. The suggested refactorings to obtain a lighter model, concentrated on a specific primary dimension, include the following transformations that collapse the hierarchy:

(1) *responsibility reassignment.* Generalization is realized by moving or pulling up data (and behaviour) from one initial primary dimension to a new class, using refactorings like: *PullUpMethod* and *PullUpField.* This transformations prepare the model to safely remove the superfluous information from the *cmC* variant.
(2) *safely removal of the specialized classes.* Only the primary dimension of the *cmB* variant, represented by the typed property `Civil Status`, through `Married Person` and `Single Person` are kept. Specialized classes related to the primary dimension of the *cmA* variant, as `Married Student`, `Single Student`, `Married Teacher`, and `Single Teacher` may be safely removed from the new model.

Refactorings applied to reduce the model to a single primary dimension, does not ensure the equivalency between the *cmB* variant and the *cmA* variant. There are two possibilities to check that a `Person` is a `Student` or a `Teacher`.

- *implicit relationship usage.* There is an implicit relationship between
  - `Student` and `Specialty` – *only* the `Student` is registered at a faculty `Specialty`;
  - `Teacher` and `Didactic Position` – *only* the `Teacher` occupies a `Didactic Position`;
- *type variable usage.* A new type variable may be introduced to distinguish between the two `Person` types (`Student` and `Teacher`).

2.3.3. *Core Ideas.* Similar to the vertical abstraction variability, in horizontal abstraction variability the concepts in the UoD are modeled using different semantic definitions. But, within the former one, the difference between variants concerns the different levels of generality, while in latter one, the difference between variants bears upon concepts that are modeled based on different properties.

In the horizontal abstraction variability, the properties may be or not *visible* and *isolated.* They are classified as *primary dimension properties* (that can be visible and isolated from others) and *secondary dimension properties* (that are not visible and cannot be isolated from others)[19].

The literature records claims that, as vertical abstraction variability, the horizontal abstraction variability affects evolvability [17]. Within the latter one, the aspect responsible for the evolvability disadvantages is represented by the information hiding highlighted within the secondary dimension.

In order to refactor from the *cmA* variant to the *cmB* variant a new intermediate with equivalent dimensions *cmC* variant is used. The transformation process from a variant with equivalent dimensions to a regular variant implies more resources than the previous step, when there were many constraints and relationships that became implicit in the resulting model.

The transformations number applied through refactoring when switching between variants depends on the number of entity types in the primary dimensions of both variants ($noPD_{cmC} = noPD_{cmA} \times noPD_{cmB}$) and the extent to which these types are used. Thus, it is expected that some range may be taken over, above which the refactoring cost weights to much over its advantages.

Refactoring literature does not record real world application of horizontal abstraction variability, though active research has been developed [15].

3. A Model for the Evolution in Conceptual Modeling Variability

Variability within conceptual modeling outlines an evolutionary process among different models of a specific variability type. This process is similar to the biological evolutionary process presented by Maturana and Varela in [13]. According to them, changes are determined by the structure of an organism and a perturbation. A perturbation itself does not determine how the organism evolves, but it triggers the organism to change its structure. The evolved organism with its new structure affects the outer environment and produces another perturbation. This iterative process of the interaction between the organisms structure and the environment through a perturbation is a driving force of evolution [13].

For a software product, customers may require new functionalities to be implemented. This results in changes that serve as perturbation in the software product evolution. In order to achieve variability within conceptual modeling, changes provided by refactoring, forward conceptual abstraction, conceptual specialization or other evolutionary changes have to be applied. There are two types of evolution in biology: *phylogeny* and *ontogeny* [13]. The former refers to the evolution as species while the latter refers to the evolution of individual living beings.

Two types of evolutions have been identified for the conceptual modeling variability within our research. A first type of evolution is similar to an *ontogenic process* where an individual living being grows. This corresponds to small changes that does not substantially affect the overall conceptual model. Major modifications on the conceptual models that have effect on the entire future development represent a second type of evolution. This is represented by a *phylogenic process* that fundamentally affects every development stage forward. Within a phylogenic evolution, the before and after conceptual models belong to different development stages and have different development approaches.

For the already studied conceptual modeling variability with its three different types, i.e., construct, vertical abstraction, and horizontal abstraction, an evolution model may be developed. Figure 10, Figure 11, and Figure 12 illustrate how the two ontogenic and phylogenic biological evolution may be modeled in the conceptual modeling variability context.

3.1. Conceptual Modeling Variability as Biological Evolution.

Evolution in Construct Variability

Figure 10(a) depicts the ontogenic evolution for conceptual models that are perturbated by small changes that does not fundamentally affect the developed model. These changes correspond to switches between *attributes and entities* ($cm_{i,j} \rightarrow cm_{i+1,j}$, $cm_{i+1,j} \rightarrow cm_{i,j}$) or *attributes and methods* ($cm_{i+1,j} \rightarrow cm_{i+2,j}$, $cm_{i+2,j} \rightarrow cm_{i+1,j}$) approaches (see Section 2.1.2). They consists of refactorings applied to the conceptual models such that their overall organization remains fundamentally unchanged.

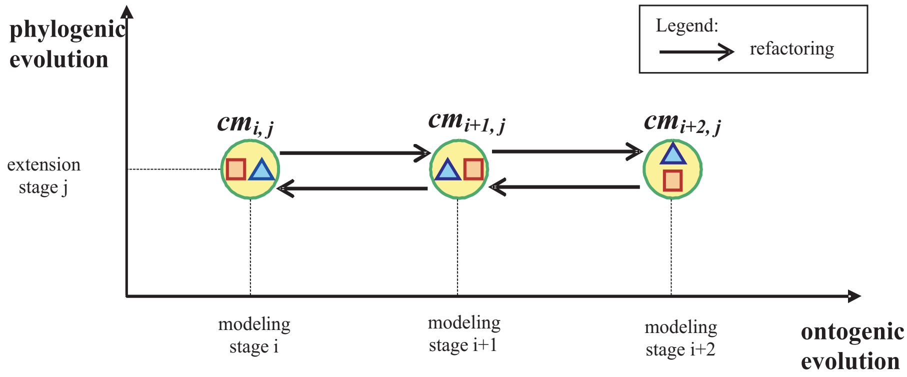
The *multiple types definition* construct variability presented in Section 2.1, outlines a phylogenic evolution by the addition of new types within the conceptual model. Figure 10(b) shows that changes within the model are reflected by a forward conceptual abstraction process, denoted by $- \rightarrow$, for *addition* of new types ($cm_{i,j} - \rightarrow cm_{i+1,j+1}$). The existing types *removal* and their replacement by type codes is achieved by conceptual specialization, relation denoted by $-\bullet$, where $cm_{i+1,j+1} - \bullet\, cm_{i,j}$. They represent small refactorings that decrease the complexity. They may be interpreted as a special case of forward conceptual abstraction inducing a different generality level between the source and the target models.

Evolution in Vertical Abstraction Variability

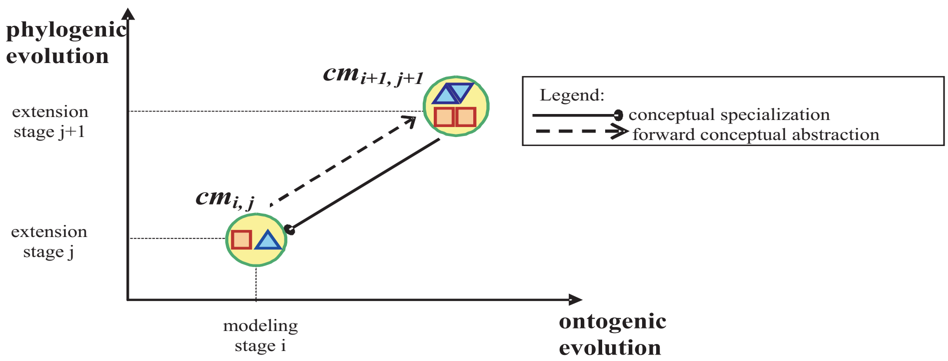
Reducing the abstraction level for a conceptual model means to remove superfluous information in order to shape a more concrete conceptual model. Figure 11 suggests that shifting from a more general to a more concrete model ($cm_{i+1,j+2} - \bullet\, cm_{i,j}, cm_{i+3,j+3} - \bullet\, cm_{i+1,j+2}, cm_{i+1,j+2} - \bullet\, cm_{i+2,j+1}$) results in changes applied to a single model, i.e., the more concrete one, which corresponds to a conceptual specialization, by reducing the model generality. In this way, the simplifying process consists of refactorings that remove the irrelevant information in the target model. On the contrary, raising the abstraction level requires additional information gathered by forward conceptual abstraction. Figure 11 shows that moving to a generic model new extension stages are added ($cm_{i,j} - \rightarrow cm_{i+1,j+2}$, $cm_{i+1,j+2} - \rightarrow cm_{i+3,j+3}$, $cm_{i+2,j+1} - \rightarrow cm_{i+1,j+2}$).

Evolution in Horizontal Abstraction Variability

Switching between models developed under horizontal abstraction variability may be done using an intermediate variant. The phylogenic evolution within this type of variability appears at the first shifting step. Figure 12 shows that addition of a new visibility dimension to the model, which drives the complexity of the development process to a higher level through forward conceptual



(a) Ontogenic evolution in construct variability, through refactoring



(b) Phylogenic evolution in construct variability, through forward conceptual abstraction and conceptual specialization

FIGURE 10. Construct variability as ontogenic and phylogenic evolution processes, through refactoring, forward conceptual abstraction, and conceptual specialization

abstraction ($cm_{i,j} - \rightarrow cm_{i+1,j+1}$, $cm_{i+2,j} - \rightarrow cm_{i+1,j+1}$). In order to reduce the number of visible dimensions, refactoring may be applied to a model within a conceptual specialization ($cm_{i+1,j+1} - \bullet\, cm_{i,j}$, $cm_{i+1,j+1} - \bullet\, cm_{i+2,j}$). This process is depicted by Figure 12 where the intermediate model $cm_{i+1,j+1}$ is used to achieve the specialization for a single conceptual model.

3.2. **Formal Approach.** In order to formalize the conceptual modeling variability as an ontogenic and phylogenic evolution some definitions are needed.

Definition 3.1. *Conceptual Model* ([5])

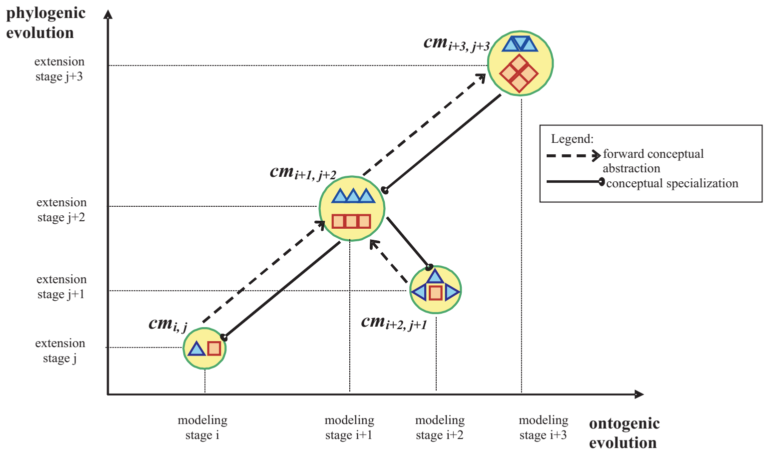


FIGURE 11. Vertical abstraction as phylogenic evolution through forward conceptual abstraction and conceptual specialization

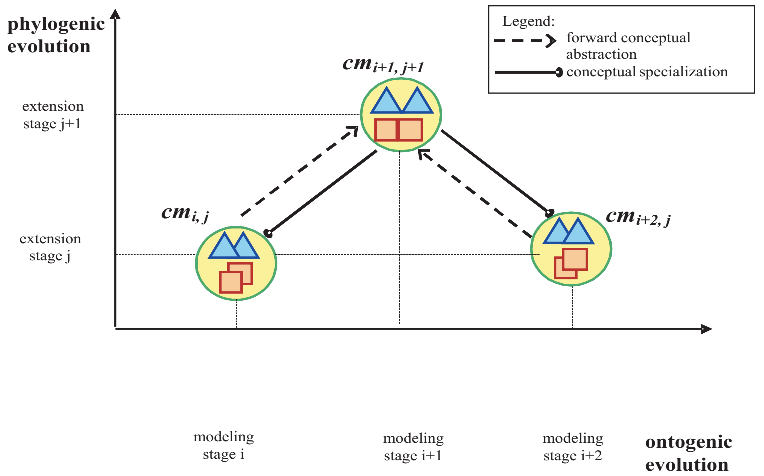


FIGURE 12. Horizontal abstraction as phylogenic evolution through forward conceptual abstraction and conceptual specialization

*A **conceptual model** is a triple $M = (E, S, A)$, where:*

(i) $E = \{e_1, \ldots, e_m\}$ *is the set of entities or concepts within the model;*

(ii) $S = \{s_{ij} | e_i, e_j \in E,\ \exists e_i\ s_{ij}\ e_j, i, j \in \{1, \ldots, m\}\}$ *is the set of associations between the entities within the model;*

(iii) $Attr_i = \{a_{i_1}, \ldots, a_{i_k}\}$ *is the set of attributes for the entity* $e_i, i = \overline{1,m}$, *and*
$A = \cup_{i=1}^{m} Attr_i$ *is the set of all attributes within the model.*

In what follows, by $\mathcal{P}(X)$ is denoted the power set of X.

Definition 3.2. *Refactoring ([5])*
Let $M_1 = (E_1, S_1, A_1)$ *and* $M_2 = (E_2, S_2, A_2)$ *be two conceptual models. A* ***refactoring*** *is a triple* $r = (e_r, s_r, a_r)$ *that transforms* M_1 *to* M_2. *Furthermore, the following constraints are met:*

(i) $e_r : E_{1_r} \to \mathcal{P}(E_{2_r})$ *maps the affected set of entities of the two conceptual models, where* $E_{1_r} \subseteq E_1, E_{2_r} \subseteq E_2$;

(ii) $s_r : S_{1_r} \to \mathcal{P}(S_{2_r})$ *maps association relationship changes between the two conceptual models, where* $S_{1_r} \subseteq S_1, S_{2_r} \subseteq S_2$;

(iii) $a_r : A_{1_r} \to \mathcal{P}(A_{2_r})$ *maps the affected set of attributes of the two conceptual models, where* $A_{1_r} \subseteq A_1, A_{2_r} \subseteq A_2$.

This is denoted by $M_1 \xrightarrow{r} M_2$.

Definition 3.3. *Forward Conceptual Abstraction ([5])*
Let $M_1 = (E_1, S_1, A_1)$ *and* $M_2 = (E_2, S_2, A_2)$ *be two conceptual models. A* ***forward conceptual abstraction*** *is a triple* $fca = (e_{fca}, s_{fca}, a_{fca})$ *that transforms* M_1 *to* M_2 *(denoted by* $M_1 \xrightarrow{fca} M_2$*). Furthermore, the following constraints are met:*

(i) $e_{fca} : E_{fca} \to \mathcal{P}(E_2)$ *maps a new set of entities* E_{fca} *to the* M_2 *conceptual model, where* $E_1 \cap E_{fca} = \varnothing, E_1 \cup E_{fca} = E_2$;

(ii) $s_{fca} : S_{fca} \to \mathcal{P}(S_2)$ *maps a new set of association relationships* S_{fca} *to the* M_2 *conceptual model, where* $S_1 \cap S_{fca} = \varnothing, S_1 \cup S_{fca} = S_2$;

(iii) $a_{fca} : A_{fca} \to \mathcal{P}(A_2)$ *maps a new set of attributes* A_{fca} *to the* M_2 *conceptual model, where* $A_1 \cap A_{fca} = \varnothing, A_1 \cup A_{fca} = A_2$.

Refactoring allows to switch between variants on the same abstraction level, while forward conceptual abstraction increases the generality of the target conceptual model, placing it on a higher extension stage. In order to reach a lower abstraction level, transformations that reduce complexity and generality are applied. Conceptual specialization decreases the abstraction level, being a special case of forward conceptual abstraction that achieves the transformation in the reverse order. Similar to this, conceptual specialization uses refactoring to move towards a lower extension stage. Therefore, a conceptual specialization is defined as a refactoring that acts as a reversed forward conceptual abstraction.

Definition 3.4. *Conceptual Specialization ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. A ***conceptual specialization*** *is a triple $r = (e_{cs}, s_{cs}, a_{cs})$ that transforms M_1 to M_2 (denoted by $M_1 \stackrel{cs}{\to} M_2$). Furthermore, the following constraints are met:*

(i) $e_{cs} : E_{1_{cs}} \to \mathcal{P}(E_2)$ *maps the set of entities $E_{1_{cs}}$ to the M_2 conceptual model, where $E_{1_{cs}} \subseteq E_1, E_2 \subseteq E_1$;*
(ii) $s_{cs} : S_{1_{cs}} \to \mathcal{P}(S_2)$ *maps the set of association relationships $S_{1_{cs}}$ to the M_2 conceptual model, where $S_{1_{cs}} \subseteq S_1, S_2 \subseteq S_1$;*
(iii) $a_{cs} : A_{1_{cs}} \to \mathcal{P}(A_2)$ *maps the set of attributes $A_{1_{cs}}$ to the M_2 conceptual model, where $A_{1_{cs}} \subseteq A_1, A_2 \subseteq A_1$.*

Following the notions previously introduced, the ontogenic and phylogenic processes are formally defined.

Definition 3.5. *Ontogenic Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models, where $E_2 - E_1 = \varnothing$, $S_2 - S_1 = \varnothing$ and $A_2 - A_1 = \varnothing$. An ***ontogenic evolution*** *is a transformation $t_o = (e_{t_o}, s_{t_o}, a_{t_o})$ that transforms M_1 to M_2 (denoted by $M_1 \stackrel{t_o}{\to} M_2$). The transformation t_o has the following properties:*

(i) *t_o consists of (small) changes that does not affect the semantics of the M_2 model;*
(ii) *t_o is a refactoring, i.e., $M_1 \stackrel{t_o}{\to} M_2$ is achieved by $M_1 \stackrel{r}{\to} M_2$, $t_o = r$.*

Definition 3.6. *Phylogenic Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models, where $E_2 - E_1 = E_p$, $S_2 - S_1 = S_p$ and $A_2 - A_1 = A_p$. A ***phylogenic evolution*** *is a transformation $t_p = (e_{t_p}, s_{t_p}, a_{t_p})$ that transforms M_1 to M_2 (denoted by $M_1 \stackrel{t_p}{\to} M_2$). The transformation t_p has the following properties:*

(i) *t_p consists of changes that affect the semantics and the abstraction level of the M_2 model;*
(ii) *if M_2 is a more general model than M_1 then t_p is a forward conceptual abstraction, i.e., $M_1 \stackrel{t_p}{\to} M_2$ is accomplished by $M_1 \stackrel{fca}{\to} M_2$, $t_p = fca$;*
(iii) *if M_1 is a more general model than M_2 then t_p is a conceptual specialization, i.e., $M_1 \stackrel{t_p}{\to} M_2$ is achieved by $M_1 \stackrel{cs}{\to} M_2$, $t_p = cs$.*

The three types of conceptual modeling variability are defined as biological evolution processes, following the ontogenic and phylogenic principles.

Definition 3.7. *Construct Variability Evolution ([5])*

*Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The **construct variability** is a transformation $cVar$ (denoted by $M_1 \stackrel{cVar}{\rightarrow} M_2$). The following statements stay within the construct variability:*

(i) *within the **ontogenic evolution** $M_1 \stackrel{t_o}{\rightarrow} M_2$:*
   (a) *a refactoring transformation is applied, i.e., $M_1 \stackrel{r}{\rightarrow} M_2$, and $cVar = t_o = r$, where M_1 and M_2 have the same abstraction modeling level;*

(ii) *within the **phylogenic evolution** $M_1 \stackrel{t_p}{\rightarrow} M_2$:*
   (a) *a forward conceptual abstraction may be applied, i.e., $M_1 \stackrel{fca}{\rightarrow} M_2$, and $cVar = t_p = fca$, where M_1 and M_2 have different abstraction modeling levels and M_2 is a more general model than M_1;*
   (b) *a conceptual specialization is applied, i.e., $M_1 \stackrel{cs}{\rightarrow} M_2$, and $cVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and M_1 is a more general model than M_2.*

Definition 3.8. *Vertical Abstraction Variability Evolution ([5])*
*Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The **vertical abstraction variability** is a transformation $vVar$ (denoted by $M_1 \stackrel{vVar}{\rightarrow} M_2$). The following statements stay within the vertical abstraction variability:*

(i) *within the **phylogenic evolution** $M_1 \stackrel{t_p}{\rightarrow} M_2$:*
   (a) *a forward conceptual abstraction may be applied, i.e., $M_1 \stackrel{fca}{\rightarrow} M_2$, and $vVar = t_p = fca$, where M_1 and M_2 have different abstraction modeling levels and M_1 is converted to a more general model M_2;*
   (b) *a conceptual specialization may be applied, i.e., $M_1 \stackrel{cs}{\rightarrow} M_2$, and $vVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and M_1 is converted to a more specific model M_2.*

Definition 3.9. *Horizontal Abstraction Variability Evolution ([5])*
*Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The **horizontal abstraction variability** is a transformation $hVar$ (denoted by $M_1 \stackrel{hVar}{\rightarrow} M_2$). The following statements stay within the horizontal abstraction variability:*

(i) *within the **phylogenic evolution** $M_1 \stackrel{t_p}{\rightarrow} M_2$:*
   (a) *a forward conceptual abstraction may be achieved, i.e., $M_1 \stackrel{fca}{\rightarrow} M_2$, and $hVar = t_p = fca$, where M_1 and M_2 have different modeling*

abstraction levels and the M_1 model is transformed to the M_2 model by adding a new primary dimension;

(b) *a conceptual specialization may be applied, i.e., $M_1 \stackrel{cs}{\to} M_2$, and $hVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and the M_1 model is transformed to the M_2 model by removing a primary dimension.*

The existing relations among various conceptual models within the same or different extension stages of the development process is formalized too.

Definition 3.10. *Ontogenic Equivalence ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. Then:

*M_1 is **ontogenically equivalent** to M_2 (denoted by $M_1 \equiv M_2$) if $\exists\, t_{o_r},\ t_{o_l}$ two ontogenic transformations such that $M_1 \stackrel{t_{o_r}}{\to} M_2$ and $M_2 \stackrel{t_{o_l}}{\to} M_1$.*

This means that M_1 and M_2 belong to the same extension stage, while t_{o_r} and t_{o_l} are *refactorings* that transform a conceptual model to another.

Definition 3.11. *Phylogenic Dominance ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. Then:

(i) *M_1 is **phylogenically dominated** by M_2 (denoted by $M_1 < M_2$) if $\exists\ t_{p_u}$ a phylogenic transformation such that $M_1 \stackrel{t_{p_u}}{\to} M_2$ and M_2 is a more general conceptual model than M_1;*

(ii) *M_1 **phylogenically dominates** M_2 (denoted by $M_1 > M_2$) if $\exists\ t_{p_d}$ a phylogenic transformation such that $M_1 \stackrel{t_{p_d}}{\to} M_2$ and M_1 is a more general conceptual model than M_2.*

This means that M_1 and M_2 belong to different extension stages, while t_{p_u} is a *forward conceptual abstraction* and t_{p_d} is a *conceptual specialization* that allows to shift between conceptual models.

4. Conclusions and Future Work

Variability occurs in almost every modeling activity and its exploitation may help modelers to switch between taken decisions and to validate the model equivalence. Refactoring techniques are dedicated to design and implementation phase, but the research shows that their applicability may be extended to the conceptual modeling level.

Even though refactoring was applied to some theoretical but representative conceptual model examples, there is a large confidence that refactoring is reliable when it is used on particular and large UoD. For each type of variability, the specific problems on refactoring between variants were studied.

There are several important ideas that emerge from this analysis:

- construct and vertical abstraction variability and the application of refactorings between them are already recognized in software evolution practice and research [9, 12];
- within vertical abstraction variability, the transformation to a more generic variant requires forward conceptual abstraction, similar to a forward engineering at modeling stage;
- within horizontal abstraction variability shifting between variants requires an intermediate conceptual model, with an inconsistent modeling state and reduced relevance.

A biological evolution model was proposed in order to cope with different types of variability previously identified. Three specific transformations were suggested to shift between *ontogenic* and *phylogenic* conceptual models: refactoring, forward conceptual abstraction, and conceptual specialization.

Furthermore, there are aspects that have to be analyzed in the near future, like: a thoroughly study of the switching process between horizontal abstraction variants and to estimate the refactoring effort between variants.

References

[1] C. Batini, S. Ceri, and S. Navathe. *Conceptual database design: an entity relationship approach*. Benjamin/Cummings, 1992.

[2] M.C. Chisăliţă-Creţu. Efecte ale refactorizării asupra structurii interne a codului. *"Analele Facultăţii", Seria Ştiinţe Economice, Universitatea Creştină "Dimitrie Cantemir" Bucureşti, Facultatea de Ştiinţe Economice Cluj-Napoca, ISSN:1584-5621*, 13(1):214–230, 2005.

[3] M.C. Chisăliţă-Creţu. General aspects of refactoring applicability to conceptual models. In *Proceedings of the Symposium "Colocviul Academic Clujean de INFORMATICĂ"(CACI2005)*, pages 99–104, 2005.

[4] M.C. Chisăliţă-Creţu. Describing low level problems as patterns and solving them via refactorings. *"Studii şi Cercetări Ştiinţifice", Seria Matematică, ISSN: 1224-2519*, (17):29–48, 2007.

[5] M.C. Chisăliţă-Creţu and A. Mihiş. A model for conceptual modeling evolution. In *The 7th International Conference on Applied Mathematics (ICAM 2010), September 1-4, 2010, Baia-Mare, Romania*, 2010.

[6] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns.* Morgan Kaufmann and DPunkt, 2002.

[7] B. Du Bois. Opportunities and challenges in deriving metric impacts from refactoring postconditions. In *In Proceedings of the Fifth International Workshop on Object Oriented Reengineering (WOOR2004), ECOOPworkshop*, 2004.

[8] M. Fayad and D. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[9] M. Fowler. *Refactoring Improving the Design of Existing Code*. Addison-Wesley, 1999.

[10] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274, 1995.

[11] IEEE. *Standard IEEE Std 610.12-1990: IEEE standard glossary of software engineering terminology, In IEEE standards collection: software engineering*. IEEE Press, 1990.

[12] J. Kerievsky. *Refactoring to Patterns*. Addison-Wesley Professional, 2004.

[13] H. R. Maturana and F.J. Varela. *The Tree of Knowledge: The Biological Roots of Human Understanding*. Shambhala Publiccations,Inc., Boston, MA., USA, 1998.

[14] T. Mens, S. Demeyer, and D. Janssens. Formalising behaviour preserving program transformations. *Graph Transformation of Lecture Notes in Computer Science*, 2505:286–301, 2002.

[15] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, and P. Van Gorp. Refactoring: Current research and future trends. *Electronic Notes in Theoretical Computer Science*, 82(3):483–499, 2003.

[16] D. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–128, 1979.

[17] H. Schmid. Systematic framework design by generalization. *Communications of the ACM*, 40(10):48–51, 1997.

[18] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz. A meta-model for language-independent refactoring. In *In Proceedings of the International Symposium on Principles of Software Evolution*, pages 157–169. IEEE Computer Society Press, 2000.

[19] J. Verelst. The influence of the level of abstraction on the evolvability of conceptual models of information systems. In *Proceedings of the International Symposium on Empirical Software Engineering (IIESE04), Los Angeles, IEEE CS Press*, pages 17–26, 2004.

Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: `cretu@cs.ubbcluj.ro`