# Object Morphology – A Protean Generalization of Object-Oriented Paradigm

Zbyněk Šlajchrt

Department of Information Technologies,
Faculty of Informatics and Statistics,
University of Economics, Prague, Czech Republic
zbynek.slajchrt@vse.cz

**Abstract.** Modeling protean objects, i.e. objects adapting their structure and behavior dynamically with respect to a changeable environment, is often challenging in traditional object-oriented languages. According to the author, the root cause of this problem lies in the class-based conceptual framework embedded in the foundation of the object-oriented paradigm. The proposed paradigm Object Morphology (OM) is greatly influenced by prototype theory developed in the field of cognitive psychology. OM abandons the notion of *class* and suggests, instead, that the abstractions of protean objects should be established through the construction of morph *models* describing the possible forms of those objects. This paper provides a condensed introduction to the theoretical foundations of OM accompanied by a series of examples. In separate sections, this paper gives a brief description of prototypical analysis, the reference implementation and OM applications.

**Keywords:** Object-orientation, modeling, conceptual framework, abstraction mechanism, prototype theory, metamorphism, Aristotelian logic, Scala.

## 1 Introduction

Object-orientation (OO) in software development has proven successful in a wide area of applications and has played a major part in software development over the past few decades [1][2][3]. Nevertheless, as shown as presented in [4], OO modeling is not able to capture well the tacit knowledge in the form of experience, heuristics and human competencies, which constitutes the major part of design knowledge today. Furthermore, according to [5], no object-oriented notation is rich enough to be able to model all key aspects of the real world.

Pivotal in OO modeling is the Aristotelian conceptual framework, in which an object represents a physical *phenomenon*, while a class may be used to represent a *concept* from the real world. Such a notion is expected to make it easier to model entities from real domains and the relationships among them [5][6].

The paper presents an alternative and "non-aristotelian" object-oriented paradigm called *object morphology* (OM), whose domain is primarily the modeling of the so-called *protean objects*. A *protean object* is a term referring to Proteus – a sea god famous of his metamorphic feats. Hence the term protean object denotes a phenomenon occurring in a multitude of forms and defying the traditional Aristotelian

class-based categorization [5]. The concepts (abstractions) of such objects may often be only loosely defined, e.g. by means of family resemblance rather than by specifying strict rules for class membership.

Examples are fetal development, insect metamorphosis, phase transitions, autopoietic (self-maintaining and self-reproducing) systems such as cells, roles in society, crisis and other biological, social or economic phenomena.

Instead of building type or class hierarchies, protean objects are modeled through the construction of *morph models* describing the forms that the protean objects may assume. The individual forms are called *morph alternatives*.

The goals of OM are:

- Establishing theoretical foundations of MO
- Formulating basic tenets of protean analysis
- Implementing a proof-of-concept application platform

## 2 Research

OM is based on prototype theory formulated by Eleonor Rosch in the 1970s [7]. That theory is used in the field of cognitive psychology to describe how people classify things. It claims that some members of a class are more central than others, and those "more central" members are called *prototypes*. In his discussion of open issues in object-oriented programming, Madsen proposes an adoption of the prototypical view of concepts into object-oriented methodologies and languages [5]. Additional information on the relationship between prototypical concepts and prototype-based language can be found in [8][9][10].

As far as rather practical effort made in this field, one must mention an UML extension called OntoUML developed by Giancarlo Guizzardi [11]. Another representative of such effort is Data-Context-Interaction paradigm (DCI) proposed by Reenskaug and Coplien [12]. Sections 0 and **Chyba! Nenalezen zdroj odkazů.** deal with those approaches in more detail and in the context of OM. Other related approaches include Aspect-Oriented Programming [13], Subject-Oriented Programming [14], Role-Oriented Programming and mixins. In [15] the author paper analyses the problem of class generalization from the non-traditional point of view – class life cycles. The paper also shows that this problem is closely connected with the "problem of conflicting identities" in generalization trees discussed on the border of the conceptual modeling and ontology engineering fields.

## 3 Methods

Chosen methods reflect the character of the problem, which lies on the border between basic and applied science:

1. Case studies (one structural, one behavioral)
2. Evaluation of three OOP languages (Java, Scala, Groovy)
3. Analysis of problematic aspects identified in the case studies

4. Generalization of the existing paradigms and methods (OOP, UML, Liskov principle) and formulation of theoretical foundations of Object Morphology
5. Verification through development of a proof-of-concept implementation and its application on a set of problematic scenarios

# 4 Theoretical Foundations of OM

This section presents the theoretical foundations of Object Morphology. With respect to its limited scope, this paper introduces the OM concepts in a rather brief manner with emphasis on intuition. Wherever it is possible, the explanations are accompanied by simple illustrations.

## 4.1 Morph Models

When modeling a protean phenomenon in the outer world, the modeler focuses on the description of the features exhibited by individual instances of the phenomenon. The individual instances may or may not share a set of common features, and even a single instance may not exhibit the same set of features during its existence. A typical example of such a phenomenon is a butterfly and its development; there is little or nothing in common between individual developmental stages, except the butterfly's identity.

In Object Morphology, the concept of a *fragment* is used to describe a feature of a modeled phenomenon. A fragment can be likened to a potentially complex attribute with possible behavior. In contrast to fragments, attributes hold data only and do not define any behavior, nor do they possess identity. On the other hand, a fragment should not be mistaken for a part of a phenomenon. The reason is that a fragment, in contrast to a part, may constitute or contribute to the identity of its phenomenon, while a part possesses its own identity distinct from its owner's identity.

A collection of fragments describing a given phenomenon at a certain moment is called *alternative*. The individual fragments in the alternative may be *passive* or *active*. A passive fragment's behavior cannot be influenced by the presence of other fragments, while the behavior of an active fragment can be overridden by other fragments (it is an analogy to final/non-final classes). An instance of an alternative is called a *morph*, which is a representation of a modeled phenomena instance (an analogy to an object).

The collection of all alternatives describing a given phenomenon is called a *morph model*. A morph model is in fact an expression of the concept of the phenomenon. A morph model may describe a phenomenon whose instances are either *immutable* or *mutable*. The immutable instances exhibit the same features until they cease to exist. The individual immutable instances differ in the composition of the fixed feature set, i.e. the alternatives of the morph model. On the other hand, the mutable instances may undergo many mutations during their lifetime and the morph model dictates the possible alternative forms.

There may exist one or more disjoint groups of alternatives sharing a certain collection of fragments. These common fragments are called *prototypes* of the morph model and the group sharing a given prototype is called the prototype's *attractor*. It

should be remarked that there may exist more ways to partition a morph model to attractors. It is on the modeler to select the best partitioning.

### 4.2 Examples

This section presents a couple of examples illustrating concepts introduced in the preceding section. The models in the examples are expressed using the R-Algebra, a mathematical formalism developed in the author's thesis [16], used to construct morph model expressions.

The *Person* model in (**1**) represents a person and consists of a single fragment – *Individual*. Such singular models consisting of only one alternative with one fragment virtually correspond to the traditional concept of class.

$$\text{Person} = \text{Individal} \qquad (\mathbf{1})$$

The notation of fragments follows the notation of classes (**Figure** 1). To distinguish the two concepts the fragment notation applies the *fragment* stereotype. It should be reminded that despite the notation similarity, the two concepts are fundamentally different, as a fragment should be seen rather as an attribute on steroids.

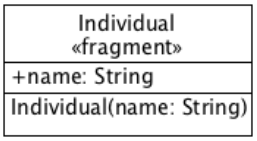


**Figure** 1: Fragment *Person*

A morph creation can be schematically expressed as shown in the pseudo-code[1] (**2**). The *p* morph can be used in the same way as if it were an instance of a class having the same structure as fragment *Individual*.

```
p: Person = Individual("Peter")
```
(**2**)

The model in formula (**3**) models the six basic human emotions. The vertical bar is the *union* operator delimiting mutually exclusive emotions (in terms of this model).

$$\text{Emotion} = \text{Joy} \mid \text{Surprise} \mid \text{Fear} \mid \text{Sadness} \mid \text{Disgust} \mid \text{Anger} \qquad (\mathbf{3})$$

[1] The pseudo-code in this section is inspired by Scala, which is the language on top of which the proof-of-concept implementation of OM is developed.

**Figure** 2 shows the structure of the *Joy* fragment. The structure of the remaining emotion fragments is analogous.

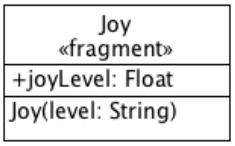


**Figure** 2: Fragment *Joy*

The listing (**4**) sketches two instantiations of emotion morphs.

```
e1: Emotion = Joy(0.7)                                    (4)
e2: Emotion = Surprise(0.2)
```

Besides the union operator, the R-Algebra formalism introduces the *join* (.) operator, by which two sub-expressions are declared as co-occurring. To illustrate the join operator let us suppose that the modeler comes to conclusion that the original model is too coarse-grained since it does not include mixed emotions. A finer-grained model may be constructed by joining the original model with itself, as shown in (**6**).

$$\begin{aligned}\text{Emotion} = (&\text{Joy} \mid \text{Surprise} \mid \text{Fear} \mid \text{Sadness} \mid \text{Disgust} \mid \text{Anger}).\ (\text{Joy} \\ &\mid \text{Surprise} \mid \text{Fear} \mid \text{Sadness} \mid \text{Disgust} \mid \text{Anger}) = \\ &(\text{Joy} \mid \text{Surprise} \mid \text{Fear} \mid \text{Sadness} \mid \text{Disgust} \mid \text{Anger})^2\end{aligned} \quad (5)$$

In case of the emotion fragments, both operators fulfill the associativity, commutativity and distributivity law with the join operator as the one being distributed. Besides those, for any two emotion fragments holds that $F.F = F$ and $F \mid F = F$. Applying these rules results in the following *canonical* form of the model.

$$\begin{aligned}\text{Emotion} = &\text{Joy} \mid \text{Surprise} \mid \text{Fear} \mid \text{Sadness} \mid \text{Disgust} \mid \text{Anger} \mid \\ &\text{Joy.Surprise} \mid \text{Joy.Fear} \mid \text{Joy.Sadness} \mid \text{Joy.Disgust} \mid \text{Joy.Anger} \mid \\ &\text{Surprise.Fear} \mid \text{Surprise.Sadness} \mid \text{Surprise.Disgust} \mid \\ &\text{Surprise.Anger} \mid \text{Fear.Sadness} \mid \text{Fear.Disgust} \mid \text{Fear.Anger} \mid \\ &\text{Sadness.Disgust} \mid \text{Sadness.Anger} \mid \text{Disgust.Anger}\end{aligned} \quad (6)$$

A morph consisting of *Joy* and *Surprise* emotions may be instantiated as shown in the following pseudo-code:

```
e1: Emotion = Joy(0.6).Surprise(0.2)                      (7)
```

So far, the example models have dealt with the fragments for which the above-mention R-Algebra rules hold. Such fragments are called *entities*. An important consequence of those rules is that no alternative in a model may consist of two occurrences of the same entity fragment. Besides entities there is another kind of fragments, which are governed by different rules. Such fragment are called *wrappers*, which owe their name to the fact that they can be stacked on top of entity fragments and override their behavior. As a wrapper example, let us consider the morph model in (**8**), which models a path.

$$\text{Path} = \text{Origin}.(1 \mid \text{Move} \mid \text{Left} \mid \text{Right})^{N} \quad (8)$$

The model includes the *Origin* entity fragment, which encapsulates the origin of the path. The other three fragments *Move*, *Left* and *Right* represent the three basic movement steps. A composition of those three steps results in the transformation of the origin into the end point of the path. For wrappers, the distributive and associative laws still hold, but the commutative law and the rule $F.F = F$ are not guaranteed. A sample path morph evolution is in the listing (**9**).

```
Origin                                                    (9)
Origin.Move
Origin.Move.Left
Origin.Move.Left.Move
```

The *1* element in the model represents the so-called *unit fragment*, which plays the role of the unit element in the R-Algebra, for which the rules $1 \mid F = F$ a $1.F = F$ hold.

The *N* parameter in the exponent determines the maximum number of steps of the paths modeled by the Path model. As long as the model should be able to describe paths consisting of unlimited number of steps, the model would have to be designed recursively, as shown in (**10**).

$$\text{Path} = \text{Origin}.(1 \mid \text{Move} \mid \text{Left} \mid \text{Right}).\text{Path} \quad (10)$$

The morph model of a compass in (**11**) illustrates a use of the so-called *inverse fragments*. The inverse fragment of fragment *F* represents the antipodal feature to the feature represented by *F*. For a fragment and its inverse fragment it holds that $F.{\sim}F=0$, where 0 is the *null* fragment playing the role of the zero element in R-Algebra.

$$\text{Compass} = (\text{N}.{\sim}\text{S} \mid \text{W}.{\sim}\text{E} \mid \text{S}.{\sim}\text{N} \mid \text{E}.{\sim}\text{W})^{N} \quad (11)$$

The fragments *N*, *W*, *S* and *E* represent the four cardinal directions. Unsurprisingly, these fragments are modeled as wrappers, as they can be stacked to express directions more precisely. The fragments prefixed by the tilde correspond to the inverse fragments of the respective directions. Hence, the sub-expression $N.{\sim}S$ expresses the fact that the north fragment may not occur in the same alternative as S. This formula prohibits invalid compositions, such as N.S or W.E.S. These combinations simply cancel out from the model due to the rule $F.{\sim}F = 0$. Just as in (**8**), the *N* coefficient determines the resolution of the compass.

Having constructed the four morph models, it is possible to compose those models and model so a traveller's mind, as shown in (**12**).

$$\text{Traveller} = \text{Person}.\text{Destination}.\text{Path}.\text{Compass}.\text{Emotion} \quad (12)$$

A traveller's state of mind includes the concept of the "self" represented by the *Person* model. The traveller is aware of his or her destination represented by the additional entity fragment *Destination*. In contrast to the two preceding models, which may be considered immutable per a traveller's instance, the remaining models are mutable. The ongoing journey is represented by an evolution of the *Path* model and the current direction to the destination is represented by an alternative of the *Compass*

model. The current compass state (alternative) influences the traveller's decisions and hence the path. The Emotion model reflects various factors, such as the distance from the destination, the weather condition (outer condition example) etc. Its presence may also influence the path decisions. The dynamics of the whole system and the influence relationships between the sub-models are defined and driven by the morphing strategy of the morph model, which determines the composition of morphs.

## 5 Applications

So far, OM has been used in a couple of applications using OM's proof-of-concept implementation called Morpheus [17]. Among the most interesting applications is the implementation of Onto UML models [11]. The Appendix 1 of the author's thesis [16] demonstrates how to transform a school enrollment OntoUML model to a morph model. That example suggests that Morpheus could fill the gap between OntoUML models and their implementations in standard object-oriented languages.

Additionally, in [18] the author demonstrates how to build an application following the principles of the Data, context, interactions (DCI) paradigm on top of Morpheus [12]. That demonstration suggests that DCI may be seen as an architectural style within the OM paradigm. Other applications are described in the author's thesis [16].

## 6 Conclusion

Research in the field of cognitive psychology shows that the prototypical view of concepts is more suited than the Aristotelian view to describe a majority of everyday concepts. However, although the Aristotelian approach has been abandoned in many fields such as biology, psychology etc., it remains the foundation of OO programming. This paper presented a new OO paradigm called Object Morphology, which is supposed to fill the above-mentioned gap. OM is a conceptual framework for modeling protean objects; i.e. objects that may assume various forms upon their creation or also during their existence. The basic tenet of OM is that the concepts of the phenomena consisting of protean objects may be built through the construction of morph models describing the possible forms of the objects. Morph models are constructed by means of the R-Algebra expressions. The Prototypical Analysis section suggests raw guidelines, formulated in terms of Prototype theory, for analyzing the problems featuring protean objects. A significant achievement of the work on OM is the reference implementation of OM, also known as Morpheus. The purpose of Morpheus is to provide a proof-of-concept prototype to allow for the evaluation of the OM paradigm in real applications. There are, nonetheless, some limitations and issues that have yet to be overcome. The main drawback concerning the OM paradigm lies in the fact that OM has not been used in any real-world application yet. Regarding Morpheus, the most important issues are its performance issues and tendency to produce boilerplate code. These issues might be attributed to the immaturity of the software as well as the limitations of the Scala platform, which had to be heavily customized.

## References


1. Zhen, L. & Tate, D. (2011) Managing Intra-Class Complexity With Axiomatic Design and Design Structure Matrix Approaches. Dajeon: Proceedings of ICAD2011, The Sixth International Conference on Axiomatic Design.
2. Booch, G. (1993) Object-Oriented Analysis and Design with Applications. 2nd ed. Redwood City, CA: Addisson-Wesley Professional.
3. Gamma, E. et al. (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.
4. Garzás, J. & Piattini, M. (2007) Object-Oriented Design Knowledge: Principles, Heuristics and Best Practices. Hershey PA, 2007. Idea Group Publishing.
5. Madsen, O.L. (n.d.) Open Issues in Object-Oriented Programming, A Scandinavian Perspective [Online]. Computer Science Dept., Aarhus University Available at: https://users-cs.au.dk/olm/index.html/PUB/OpenIssuesInOO.pdf .
6. Deitel, H.M. & Deitel, P.J. (2005) Java How To Program. 6th ed. Upper Saddle River, New Jersey: Prentice Hall.
7. Rosch, E. (1973) Natural categories. Cognitive Psychology, (4), pp.328-50.
8. Taivalsaari, A. (1996) Classes vs. Prototypes, Some Philosophical and Historical Observations. Journal of Object-Oriented Programming, pp.44-50.
9. Krogh, B., Levy, S., Subrahmanian, E. & Dutoit, A. (1996) Strictly Class-Based Modeling Considered Harmful. In System Sciences. Wailea, HI, 1996. IEEE.
10. Mihelič, J. (2011) Prototype-based Object-Oriented Programming [Online]. Univerza v Ljubljani Available at: http://lalg.fri.uni-lj.si/~uros/LALGinar/arhiv/ProtoOOP.pdf .
11. Guizzardi, G. (2005) Ontological Foundations For Structural Conceptual Models. Centre for Telematics and Information Technology, University of Twente.
12. Reenskaug, T. & Coplien, J. (2009) The DCI Architecture: A New Vision of Object-Oriented Programming [Online]. Available at: http://www.artima.com/articles/dci_vision.html .
13. Steimann, F. (2006) The paradoxical success of aspect-oriented programming. In Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '06). New York, NY, USA, 2006. ACM.
14. Harrison, W. & Ossher, H. (1993) Subject-oriented programming (a critique of pure objects). In Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '93). Washington, D.C., 1993. ACM.
15. Řepa, V. (2012) Modelling Life Cycles of Generic Objects. In Proceedings of the International Conference on Information Systems Development. ISBN 978-1-4614-7539-2.
16. Šlajchrt, Z. (2016) Object Morphology – A Protean Generalization of Object-Oriented Paradigm. Dissertation Thesis. [Online]. Available at: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=27230;lang=en.
17. Šlajchrt, Z. (2015) Morpheus, A Proof-of-Concept Implementation of Object Morphology in Scala. [Online]. Available at: http://github.com/zslajchrt/morpheus.
18. Šlajchrt, Z. (2016) Designing Applications Using Data-Context-Interactions Architecture In Morpheus. Journal of Systems Integration, 7(2).