Context-Aware Formalization of Inter-Model Relations

Ahmet Egesoy
Department of Computer Engineering,
Ege University
Izmir, Turkey
Email: ahmet.egesoy {at} ege.edu.tr

Abstract— **Model-Driven Engineering involves incremental development of code through the collaboration and transformation of models by automated tools. The aimed automation makes the semantic aspect of modeling, a significant implementation concern since in a way it requires the machines to understand what they are doing. In this paper we introduce an approach in which, models are accompanied by scripts that provide them with descriptions of their semantic context that participate in their usage as a model. Our approach focuses on inter-model relations and also uses some concepts from *Peircian semiotics.***

Keywords-model-driven; megamodel; paradigms; semiotics

I. Introduction

Model Driven Software development (MDSD) is an innovative paradigm in which models, instead of code takes the leading role in the software development process. In a typical MDSD project, there will inevitably be lots of models in complicated relations with each other. Most probably there will be many models that refer to each other or each other's elements. In the software development process it is common that one element of a system is defined in one diagram and referred in another. Classes for example are defined in CRC cards and UML class diagrams and are referred in many parts of the project such as the collaboration diagrams.

In a complicated development environment it is not easy for the machine-mind to determine what each model means and how it can contribute to answering the informational requirements of the project. As the projects grow and the requirements increase, models also get more and more complicated. Dividing the models into interrelated modules can be a good solution for dealing with complexity. In that case the structural and semantic *inter-model relations* get even more entangled. The main aim of this work is to propose a method for making the meaning and functionality of each model clear by constituting strong ties between the models and the other models, and also the systems that they model.

It should be noted that in this work the unification principle "*Everything is a model.*" as stated by Bezivin [1] is promoted as a starting point for analyzing different kinds of model interaction, instead of keeping the study restricted to UML class diagrams. We also use the term *context* in a broad sense such that it not only refers to the interpretation environment of the *model developer*, but also to that of the *model user*. Therefore in this work context awareness also implies semantically integrated, dynamic behavior of models as they respond to queries.

Most of this work is composed of a relational language's initial specification that makes use of Favre's Megamodel [2]. Formal syntactic definition of the language is not given but the changes made on the original approach are explained in detail, leaving little space if any for ambiguity. We should also note that the defined language is not an alternative for the existing Megamodel, since the purposes of the two approaches are completely different. Nevertheless by using existing terminology, a set of complicated definitions could be made tidier and easier to be expressed.

In the second section there is a discussion regarding how models receive their meaning. In the third section a new inter-model relational language will be described, based on the existing Megamodel concept. In the forth section there is a projection on how inter-model relations can be used by automated tools for answering queries. Fifth section contains the conclusions.

II. The Meaning of a Model

In MDSD where models should deliver working systems, determining the *meaning* of a model is not just a philosophical challenge but much more than that, it is a crucial design issue. In model driven software engineering, models should eventually end up in code or some part or aspect of code. The meaning of a model lies in what it does for you in the development process to reach the final product. In this respect models get close to traditional code in the sense that they both have perceivable input and output, and also perceivable functionality.

People are culturally inclined to think in terms of dualities and naturally assume a syntax-semantics duality which implies that language utterances have one unique meaning, just as they have one unique form. The everyday conversation often neglects to underline the accepted status of the model concept as a relation (or role) giving the impression that meaning was an intrinsic quality of models that had to be discovered. This leads to a confusing discussion about what semantics is and what it is not [3]. On the other hand it is misleading to think of meaning as an absolute monolithic entity that is related to language semantics. If that were the case, machine translation of Shakespeare into Japanese would be very easy. Instead, the meaning of a complex linguistic structure is usually acquired through a multi-phase symbol grounding process that generates a series of interpretations of decreasing abstraction degrees. In the interpretation process it is not only the language definition that takes control but the *context* in which the interpretation occurs and the intentions of the interpreter (person or machine) are also important. In literature this scheme creates concepts like *subtext* and *subjectivity*. In programming languages there is the concept of binding and binding times. Evidently a model potentially has more than one meaning.

A model without any contextual information is like a map without a legend, a scale, a north arrow, a title or a "*you are here"* mark. Note that these things are semantic anchors that link the map to the real world in various ways and in their absence; it would be very difficult to make an effective use of any map. Unfortunately this is the case for many object-oriented models that are created during various phases of software development processes. In [4] Daniels suggested that an "*indication of purpose"* was necessary for models. We have mentioned in [5] the problem as the *orientation problem*. Our theory is that if a sketch is drawn for some system with the aim of using it as a model, (especially for formal usage that involves a *machine's interpretation*) we have to state what exactly has been drawn on paper and how exactly it has been represented (with which perspective, simplifications and stylization etc.).

There are two areas that need to be clarified for each model. One area is the *language and context* in which the syntactical model elements are to be interpreted and the other is the *modeling services* provided to the users of the model. Through this language the model may provide modeling services such as referencing, creation or modification of the original system or parts of it.

Fig. 1 shows a hypothetical model representation as a layer between the language that it uses and the language that it creates. The *linguistic references* label in the figure represents all the meta-information accessible by the model in order to build the correct expression. This includes its meta-model (dealing with syntactical issues) and also the contextual layers that the model refers to. The domain references label marks the entities that the model *talks about*. The *modeling interfaces* are implicitly or explicitly defined transformation rules that instruct the modeling environment about how the queries (and which ones) should be answered by using the information stored in the model.

Some of the modeling services provided by the model can be defined explicitly by a rule written by the modeler. A rule may define for example how to use a class diagram as a definition for the classes that it contains. The other services can be deduced from the relations of the model with other models.

For example if two models are known to have *symmetry* between them, this means that certain attributes of one model can also be read from the other model.

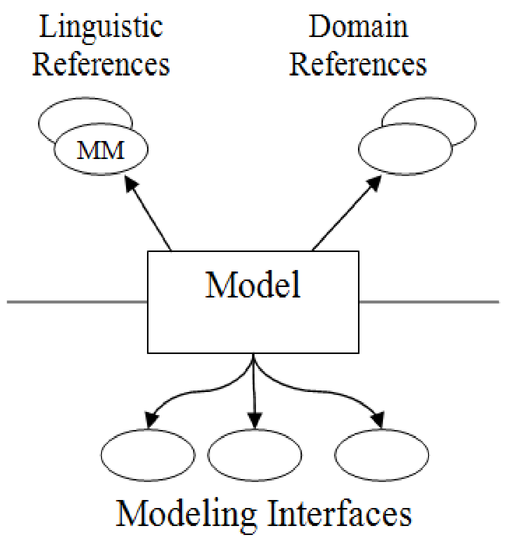

Figure 1. Linguistic relations of a model

Fig. 2 is a representation of a *modeling interface*. It is defined as a transformation between a query and the result that it produces. The script (or rule) in the interface guides the transformation by referring to the model. Queries may come from the user or from the environment itself as a part of a bigger transformation task. Each new interface of a model is a straightforward definition of a new type of use for that model.

III. Relations between Models

A key aspect of formalization for models is to codify inter-model relations in a precise and effective way. Favre's *Megamodel* [2][6] is a significant concept of MDE that deals with inter-model relations.. The relations of this language are given below with their short definitions [2].

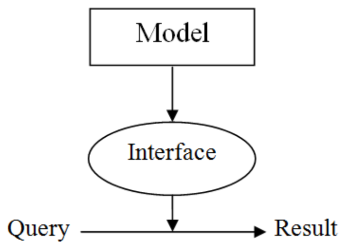

Figure 2. Model interface performs a transformation

The interesting aspect of the Megamodel is that it addresses a strategic field of MDE that had been neglected for a long time; that is expressing and comprehending the evolutionary situations in a model driven development process. The Megamodel defines a small but adequate relational language for this task. It is composed of relations whose labels are usually abbreviated as Greek letters

δ (DecomposedIn) **:** Whole-part relation.

μ (RepresentationOf) **:** The relation between a model and the thing it models.

ε (ElementOf) **:** The relation between a set and its element (Set concept includes languages too.)

χ (ConformsTo) **:** The relation between a model and its meta-model. (Also that between a grammar and its instance)

τ (IsTransformedIn) **:** The relation between the input and output of a transformation. (No constraint)

This language should be extended in order to form a more elaborated language that is able to describe inter-model relations with higher precision and detail. For this purpose we propose to use:

- Some new relations
- Parameters for the existing relations
- More concrete definitions

The whole-part relation δ can be generalized into a family of relations called as the *structural relations*. These relations are about the structure of a system or the syntax of a model. The relation is present between a record and its fields, as well as an array and its elements. It forms a hierarchy tree that starts with the biggest piece at the root and ends up with the smallest ones at the leaves. Any given hierarchy tree is probably one of the many possible structural views of the system. Different schemes of parsing the system may result in different structural views. So our δ relation should be able to express the following through its attributes:

- The name of the part or subsystem
- If the system is a list or array, the element number
- The view in which the relation holds (a name for the method of parsing)
- Moving upwards in the tree towards the root.

The first two are related with the identification of the part within the whole, and the third piece of information indicates the specific perspective of the system (or type name may be indicated for a polymorphic object) under which the part exists. By meeting the forth requirement it is possible to promote the relation into being a complete language for tree browsing. Simple directory path expression-like syntax is adopted for addressing the nodes of the tree.

The Fig. 3 demonstrates the extended δ relation by instantiating it between a basic student record (the node on the left) and its field stdNo (denoted by the node on the right, meaning student number). Although a basic record is very unlikely to have other views, the example also indicates the corresponding type of the data structure (student) according to which the part (stdNo) is refered. The part's name (for this case the field's name) is written as superscript and the view's name is written as subscript. The relation underneath is in the

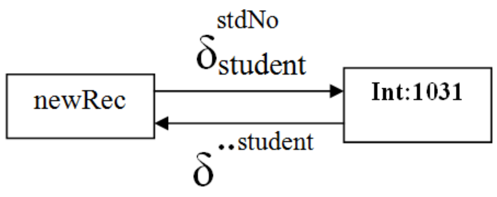

Figure 3. Structural relations

opposite direction and it indicates a structural relation from the pieces towards the whole. The superscript starts with double periods, which indicates moving upwards. The label *student* is written next to it, meaning that the target of the relation is the closest container system, which is of type *student*. Without any type name, double period just means *immediate container*.

The signification relation Σ (capital sigma) which originates from the semiotics science [7][8] is one of the relations that should to be added to the Megamodel's set of relations. The sigma family of relations is composed of the most basic signification relation called the *indexical signification* [8] and the most general understanding of signification which is called the *symbolic signification* [8].

Semiotics is simply about meaning of things. In the science of semiotics any meaningful linguistic entity is called a sign and it is simply visualized as a pointer that points at a real-world or a conceptual object. The science of semiotics deals with the mechanisms of various instances of pointing at (called semiosis) that can take place directly or indirectly through some medium and being interpreted by an intelligent mind.

Indexical signification is a bipartite relation that simply indicates that the source points at the target. The signification takes place immediately by using a physical tie, without any need for symbolic interpretation. It is simply a pointer. Indexical signification is shown with a sigma, subscripted with an "i" (Σ_i) . Symbolic signification on the other hand is a triadic relation. It is based on the *semiotic triangle* of *Perician semiotics* [9] but with a little modification [10]. The element at the top corner of the triangle called interpretant (or concept) has been replaced by an interpreter for some theoretical and practical reasons [10] related to implementation on computers.

Symbolic signs as defined in semiotics [8] are those that point at their object through an interpretation operation that is based on conventions or norms. This physically means that the relation between the sign and the object is random. Randomness is implemented by introducing a third parameter called the *interpreter* which constitutes the *pragmatic* relation between the sign and the object. Interpreter provides a *transformation* on the sign, yielding an indexical reference to the object (which does not require further interpretation). In some cases the interpreter can also represent the *context* in which the sign functions. For instance in the programming language *C*, an integer value can stand for a character or a Boolean value as well as a numerical value. Similarly a class name in UML may refer to an entity in the design model, as well as another one in the conceptual model. In some languages such ambiguities are prevented by implicitly stating the required interpretation.

Signification is a triadic relation. The interpreter name is shown as a third participant of a triadic relation as in the Fig. 4. Alternatively it could be written as a subscript to the letter Σ.

Peircian semiotics [9][8] mentions a third kind of signification called *iconic signification*. In [10] it is argued that *iconic signification* which relies on similarity can be handled as a special case of *symbolic signification*, therefore it does not require a special representation.

Examples of the sigma relation are all around us, since it represents the denotation of concepts, real world objects and computational objects in a similar fashion. The relation between the value in the name field of a student record, and the real student in the real world is signification. Likewise an identifier signifies a value for a compiler (or interpreter) and a query signifies a result set for a relational database. Signification (semiosis) takes place everywhere that there is language.

The next relation that should be considered is μ. In the Megamodel this relation is named as *RepresentationOf* and it is defined as the relation between a model and its *system under study (SUS)*. On the other hand as the unification principle (Everything is a model.) [1] suggests, the original definition of the modeling relation that is generally accepted does not impose any restrictions on the specific shape of a model. In fact a model is defined as being any system that can provide information about another system [11]. This definition is so broad that any two systems that are somehow related can potentially considered to be models of each other. For instance the presence of a χ relation indicates that there is some structural information on the other side. A Σ relation shows that the address of the system is known and δ means that the other system is already a part of the original system so in all these cases there is some information shared.

Modeling concept is too general to be the definition of a relation in a scripting language as μ relation is expected to function. A well-defined relation should be making statements about its source and target. On the other hand it is true that the common understanding of modeling involves that the model *resembles* the *SUS* somehow. Therefore one relation that would worth representing is the *symmetry* between two systems.

Being a mathematical concept, *symmetry* has a concrete definition. Two mathematical objects are said to be symmetric to each other if one can be obtained from the other through a sequence of operations that do not change some invariants. In this case the symmetry is said to be taking place with respect to those operations. It is hard to define a general methodology to detect symmetry between two complex systems. However with the help of the signification relation Σ, it is possible to formulate a necessarily general definition.

Symmetry may take different forms based on the nature of invariant that is preserved between the systems. It can be some *structural symmetry* (*similarity*) or a purely abstract symmetry. The preservation of an invariant between the systems can be visualized as two systems pointing at the same value on a certain domain. This way it is possible to formalize the similarity type in terms of signification relations and express them as semiotical patterns [10].

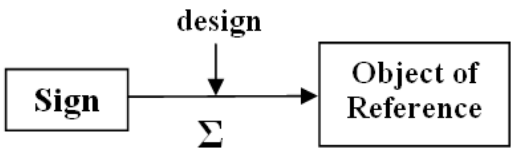

Figure 4. Symbolic signification triad

As depicted in Fig. 5, there is structural symmetry between an *object* and a *model* if their interpretation in the same language (more precisely by the same interpreter *I*) yields the same value. The common interpreter is written as a subscript for the μ relation and the relation is named μ_I. Structural symmetry takes place when for example a complex arithmetical expression is simplified (without changing the value) or a software system is re-factored without changing its functionality. There is no language change here.

There is also a loose kind of symmetry that we call *semantic symmetry*. It simply means that two systems have the same meaning although they are written in different languages. The acquisition of the model from the system in this case corresponds to a *translation*.

Fig. 6 shows the representation of semantic symmetry in terms of signification relations. Because the *model* and the *object* are represented in different languages, there is a different interpreter for each. When the relation is meant to define a transformation between the object and the model, it corresponds to a *translation*. *Translation* is probably the most well known transformation in the MDE literature since it provides moving from one technological space to the other. Transformation from the UML class diagram into a *database model* is a common example. Straightforward examples for this phenomenon are translation of basic sentences between natural languages and translation of code between programming languages.

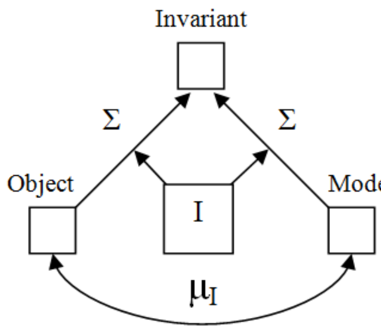

Figure 5. Structural symmetry relation as a semiotic pattern

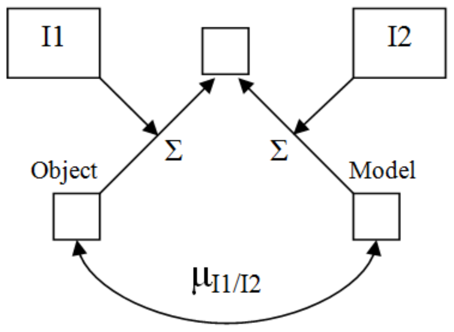

Figure 6. Semantic symmetry relation as a semiotic pattern

Because there are two languages involved, semantic symmetry is not physically symmetrical. In order to indicate which interpreter applies to which system, both are indicated in the subscript. So the relation is named as $\mu_{I1/I2}$ when written from the object to the model and as $\mu_{I2/I1}$ when written from the model to the object.

Another relation that we propose as a new language element for a better expression of inter-model relations is called as *Views* and represented with a σ (lower case sigma). This relation holds between two systems when the elements of one provide indexed access to the elements of the other. This creates a new view of the original system. In contrast with a μ type of model that provides an encoded copy, a σ model provides access to the original (thus up to date) information but through its own format and perspective. σ relation can be visualized as the relation between the Σ interpreter of a language and the domain of that language. The interpreter interprets the signs and returns indexical signs (true pointers) that point at the values in a domain. At this point if we interpret the domain as a referable object, the interpreter becomes its model of type σ.

The Fig. 7 shows this condition that takes place in a signification relation. The interpreter translates the signs into a pointer that points at the small rectangle labeled as *V* (Value). By doing this, the interpreter is also providing an index to the elements of *Domain*. Between *V* and *Domain* there is δ relation (implicitly shown) and as a result between the *Domain* and the In our approach the letter χ continues to denote the *ConformsTo* relation. We tend to see it as a ground element (not a derived relation) for the basic practical reason that it can be implemented directly. In fact it constitutes one of the two main methods for implementing sets (The other method is to explicitly list the elements.) and χ is also used more often than ε (*ElementOf*). Grammars are structural models with clear-cut interfaces. They are used for reading, writing (creating) and checking (similar to reading) systems of a certain structure. Some grammars can not be used for creation because they provide a *descriptive model* of the system (in contrast with a *generative model*) that does not give a complete picture of the structure. This is an important aspect for the *ConformsTo* relation so we propose that it should be indicated on a

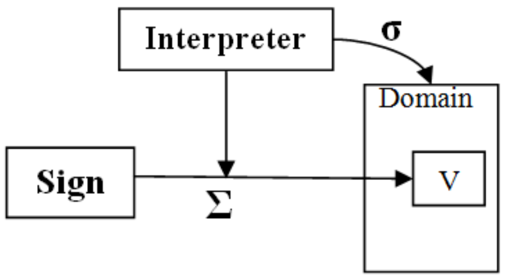

Figure 7. Semiotic pattern for σ

superscript of χ as a minus sign for *descriptive models* and as a plus sign for *generative (complete) models.*

The Fig. 8 describes a reading operation on which it is possible to see χ and σ relations in context. The system to be modeled is being read by using a descriptive meta-model which it *conforms to*. The reading operation is marked with the letter τ as it is a transformation. The operation creates a new model which is a view of the system. The relation between the view and the system is labeled as σ with a subscript *M* (the meta-model's identifier) and a superscript *minus* (indication of an incomplete view).

As it can be noticed from the Fig. 8, the τ relation is triadic by nature as it requires a *meta-model* as well as the system (to be transformed) as input. A meta-model functions as a grammar for reading and creation operations, however for more complex transformations a full transformation script should be provided.

The last Megamodel relation given by Favre [2] is the *ElementOf*: ε relation does not need modification, thanks to its mathematical root, providing reliable semantics and adequate abstractness. On the other hand it also does not seem to be a popular linguistic element that is expected to be used frequently. The minimalism and completeness of the Megamodel relation set are beyond the scope of this work, however the challenge of building the perfect canonical set of inter-model relations (the perfect Megamodel) should be noted as an important concern for future work.

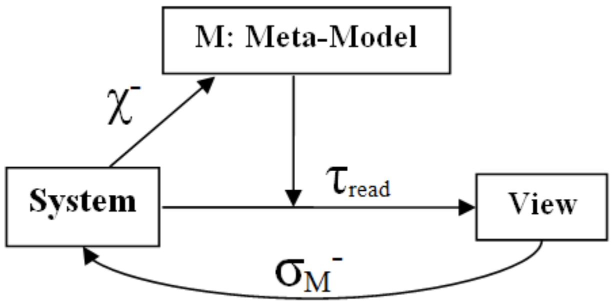

Figure 8. Reading the system with a meta-model

IV. Derived Modeling Relations

The central theme of formalization of inter-model relations is that if they constitute a rich and precise language, they can be used in order to create models dynamically, as the need arises. This can be achieved by making knowledge available to the machines by making sure that all the models in a project are semantically connected. This way every model has a relative semantic address with respect to any other model and functionally the *whole project can become one big model*. For semantic completeness the projects should also contain references to the real world or some form of ontological domain on a common network.

The example in Fig. 9 contains a student (in the real world) whose name is Bob, and a computer record object that models it. The modeling relation between a real world object and a computer object (which is in fact a record) is always an incomplete semantic symmetry. This is shown as μ- in the figure and the case that it is only semantic symmetry is not explicitly indicated but it is obvious. There is a translation from the real world to a data model. + Σ part of the expression indicates that the model not only is symmetrical to Bob but it also points at Bob. This is because the record contains a Name field with which real world people can be referred. This makes the record a logical statement (like Bob's age is 15 and so on). The Student object *ConformsTo* the Student class and this relation is complete (marked as +). Between the Student class and Bob there is a derived relation. This is also some kind of indirect modeling because by looking at the class it is possible to tell things about Bob (like that he is a student). This relation between Bob and the class can be written as a combination of the other two relations as: χ+ o (μ- + Σ).The letter "o" stands for *function composition operator*.

Suppose that a university information system is being developed in Java, in a model driven development environment, and at some point in the process, the developer asks the tool to create a code frame for a class called *Advisor*. The first thing the tool should do is to check if a *complete model* of the *Java class Advisor* exists or not. This may be in the form of a detailed class diagram, a piece of pseudocode or a CRC card. If the environment has access to a suitable transformation script from the modeling medium to Java, that script can be called with the instruction to create the Java class.

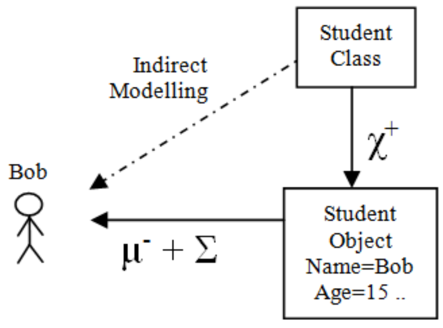

Figure 9. Modeling a real world object

It is also probable that a model does not exist for the *Java class Advisor*. In that case the tool checks all the models for any rule that promises to provide a symmetric model of *Advisor* (one with that conforms to *x* where *μ (Advisor,x)*). It is very likely that a general class diagram would answer this call. The main purpose of such a class diagram is not to define the individual classes that it contains. It usually serves as a structural model of the system as a whole and focuses on the relations between the classes. However by looking at a class diagram it is possible to perceive some of the class definition and a code frame can be written.

If a class diagram is allowed to manufacture special models for the classes that it contains, then this should be formally stated as a modeling interface that defines how an incomplete definition of a class can be derived from the general diagram. This can be done by writing a transformation script. For this case where all the needed information is already there, the transformation performs a reading operation and forms a view of the required definition.

$$\sigma \text{ o } \mu^{-} \text{(Advisor):- } \sigma^{+} \text{ o } \delta^{\text{Advisor}} \text{ (CD)} \qquad (1)$$

The script (1) is an *interface* rule example for a model. It gives a representation of the rule that could instruct a tool that such modeling is possible. The first line is the left hand side of the rule and it specifies what kind of service is available. Here it states that an incomplete reference to a symmetric model of *Advisor* can be simulated. The right hand side of the rule (second line) describes how to achieve this. The script tells the tool to start with *CD* (class diagram's name) then take the part called *Advisor* (the delta relation) and then produce a full reference to the available information. This rule of course is just for the purpose of demonstration of the *interface* phenomenon and typically more general rules are needed without mentioning any class names directly.

V. Conclusions

In this work, an approach that aims achieving semantic and functional integrity in a model driven software development environment has been introduced. In this approach, a declarative language is formed by extending Favre's Megamodel [2] and introducing two new relations: *signification* and *viewing*. This language can be used in order to specify *semantic positions* for models relationally, with respect to each other and the real world concepts. The addition of semiotic relations and concepts like *symmetry* to the paradigm provides a better understanding of the inter-model relations. This extended language with its increased expressive power enables the composition of complex inter-model relations by using the elements of a finite Megamodel. For future work we aim the implementation of this scripting language and the required inference mechanism and make them parts of an integrated modeling and transformation environment project.

REFERENCES

[1] Bezivin, J., "On the unification power of models". Journal on Software and Systems Modeling vol. 4, 2005, pp. 171–188.

[2] Favre J.M., Nguyen, T., "Towards a megamodel to model software evolution through transformations", SETRA Workshop 2004, Elsevier ENCTS, vol. 127, no:3, 2005, pp. 59-74.

[3] Harel, D., Rumpe, B., "Meaningful Modeling: What's the Semantics of 'Semantics'", IEEE Computer Society, vol. 37, no. 10, 2004, pp. 64-72.

[4] Daniels, J., "Modeling with a sense of purpose", IEEE Software, vol. 19, no. 1, Jan./Feb. M. Fowler, Eds. 2002 , pp. 8-10.

[5] Sikici, A., "Modeling from a semiotic perspective" Proceedings of the 2005 Symposia on Metainformatics, ACM International Conference Proceeding Series, vol. 214, no:14, New York, NY, USA, ACM ,2005.

[6] Favre, J.M., "Towards a basic theory to model model driven Engineering", 3rd Workshop in Software Model Engineering (WISME2004), at the 7th International Conference on theUML (UML2004), Lisbon, Portugal, 2004.

[7] Ryder, M., "Semiotics: language and culture" in Encyclopedia of Science, Technology and Ethics. Macmillan Reference, USA, 2005.

[8] Chandler, D., Semiotics for Beginners, Aberystwyth University, http://visual-memory.co.uk/daniel/Documents/S4B/,(Accessed 10.October.2014), 2005.

[9] Peirce, C.S.: Collected Papers of C. S. Peirce, Volume 8. Arthur W. Burks, Eds., Harward University Press, Cambridge, Massachusetts, 1958.

[10] Egesoy, A., Topaloglu, N., Y., "A Bottom-up model of computational semiotics", Information Systems in the Changing Era: Theory and Practice, Proceedings of ICISO'09, Aussino Academic Publishing House, 2009, pp. 26-32

[11] Minsky, M.L., "Matter, mind and models", Semantic Information Processing, ed Marvin Minsky, MIT Press, 1968, pp. 425-432.