International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-9, Issue-1, January 2021
https://doi.org/10.21276/ijircst.2021.9.1.6
Article ID IRP1126, Pages 26-32
www.ijircst.org

Meta-Modeling of AI for Software Modularization

Dr. Ahmet Egesoy

Assistant Professor, Department of Computer Engineering, Ege University, İzmir, Turkey

Correspondence should be addressed to Ahmet Egesoy; ahmet.egesoy@ege.edu.tr

ABSTRACT- Recent developments in artificial intelligence have surprisingly been only on the machine-learning related technologies. This growing trend brings new hardships to the already problematic AI programming sector that looks like a zoo of paradigms. AI is unfortunately full of incompatible technologies that can hardly cooperate in a common multidisciplinary project. These technologies are also under the threat of being abandoned in favor of the emerging machine learning techniques. However, there are many valuable ideas and concepts in the classical AI approaches that can be quite useful in the awaiting challenges of general AI. Such a great endeavor will necessitate everything we know about representing and processing knowledge. Meta-modeling of the AI domain as a whole can bring about model driven development as a glue for the fragmented development efforts. In the long run it also has the capacity to trigger a unification and revival of the art of AI programming around a more structured central paradigm.

KEYWORDS- AI, Model-driven, Semiotic.

I. INTRODUCTION OF THE PROBLEM

Artificial Intelligence (AI) is a very popular branch of computer science [1] that has made great progress in a surprisingly short time. Towards the end of the 20th century there was not much optimism about the future of AI since the contemporary developments have been in a rather eclectic fashion. Nevertheless, in just a few decades it has made great leaps both scientifically and commercially. AI-enabled tools now help to direct the efforts of sales teams by gathering useful information from raw data. They also help finding solutions for reoccurring problems, and efficiently addressing customer demands. Security, medicine and self-driving cars are probably the three pioneer sectors where innovations are the most frequent. There are a lot of success stories in all of these fields and the socio-economic impact of AI is getting more and more significant. The problem with AI is that it has always been just a blanket term that covered multiple small paradigms each of which were good at solving an entirely different type of problem. In fact, there is still no central theory in AI. There are expert systems, rule-based approaches, logic representations, evolutionary algorithms, constraint satisfaction systems and connectionist approaches (and more) each of which imposes its own point of view in problem solving. AI has grown more like a bush than a tree.

What happened recently was that one lonely stem from that bush namely *machine learning*, started to develop disproportionally and at a great pace. The *machine learning* approach is computationally one of the most expensive, but it is especially good at *pattern recognition* which is ironically the weakest side of symbolic computation on which computer technology relies on. The weakest feature of digital computers has become their strongest feature now. Most of what has been achieved are owed to a learning model called *neural network* which just gives up all that is *symbolic* and instead simulates *connectionist* structure of the human brain tissues.

Neural networks have been there since a long time ago. Only recently, three things came together to create this explosion of success. These are hardware capacity, tons of data available (especially on the Internet) and initial demand that created an economy that kept the progress sustainable (which was mostly e-commerce related).

Today neural-network based machine learning approaches are stretching towards fields other than basic pattern recognition tasks and achieving success too. Although this view looks like symbolic AI has come to an end, some people believe that it is not the case as was discussed in [2]. In symbolic AI there are many ideas and solutions that are too good to be given up. When we consider that general AI (strong AI) is the ultimate goal of the discipline, it is evident that good ideas and solid concepts are as important as success stories.

The time we will need to appreciate an integration of AI-related paradigms is approaching. This work does not claim to formulate the *grand unified theory* of AI. However, in my view, by solidifying the relations between some fundamental concepts, it is possible to lay a foundation for a better cooperation between the paradigms of AI. In time a common understanding of concepts may lead to more than one implementation of such unification.

In the short term, for promoting modular development and reuse, model-driven software development [3] can be a suitable foundation. *Model-driven software development* paradigm aims at supporting meaningful incremental development of the required models and code through the collaboration and automatic transformation of existing models. In this approach models are regarded as reusable first class artefacts and software development process is

viewed as a semi-automatic course that can produce code for multiple platforms.

The mainstream approach in model driven software development is based on a standard called *Model Driven Architecture (MDA)*. *MDA* is a layered architecture that enables building *domain specific languages (DSL)* for any domain, for more designer-friendly development. Domain specific languages are defined by proving a meta-model, and by conforming to a global singular meta-meta-model, they all integrate into the texture of MDA. In this approach, both language and transformation definitions are meta-model related issues.

Of course language semantics cannot be embedded into a meta-model which is aimed to deal with syntactical validity. However, it is my belief that a meta-model is still a good way for clarifying things and creating a sense of compatibility through a common reference for the use of terminology.

II. ANALYSIS OF THE DOMAIN

A. Programming Requirement

It is a solid requirement that AI has to be a programmable feature at least now and in the near future. It is a fact that from a humanistic point of view the success of AI has come too early and from an unexpected branch (machine learning). Our cultural readiness for accepting AI as a part of our daily lives is already questionable. The current trend in technological developments unfortunately seems to be demanding even more than that. Learning started to replace programming and machines are taking control of their development at a much earlier stage than expected.

The explainable AI movement [4] appeared as a resistance to invasive application of black box machine learning methods. Unfortunately, black-box Machine Learning (ML) models are increasingly used for making sensitive judgements and taking critical decisions. There is an interesting demand for transparency in the software development sector and all the areas where these products are deployed. There is a legal and ethical risk about these machine-originated decisions that are not straightforwardly justifiable or legitimate.

Humans do make mistakes but humans also take responsibility for and if possible try to correct their mistakes. Classical software engineering involves debugging as it relies on human programmers. However, when a system acts on its own and does not allow obtaining detailed explanations of its behavior, it is a great obstacle for process refinement. Therefore, the output of a model should be supported with adequate explanations. This is especially true for mission-critical applications such as medicine, security or autonomous vehicles in transportation. These areas where errors can be lethal, severely require the ability to backtrack responsibility, for ethical and legal reasons.

To summarize, some human control needs to be maintained in order to keep the AI projects ethical, scientific and manageable. We need to find new ways to keep AI transparent enough so that it stays as an object of engineering, design and programming.

Programming in AI has long been dominated by two programming languages LISP and Prolog (and their close relatives). Both are simple and elegant languages, that are very capable in spite of their simplicity. These languages however appeared in an era when AI was seen mostly as an experimental field. They rely on flexible but expensive constructs such as recursion (for both) and backtracking (for Prolog) instead of iteration statements; they are usually implemented as interpreters, and lastly they provide poor support for modularization. This last weakness is especially important because it prevents the application of proper software engineering for larger projects.

There have been some serious attempts for adding module feature to these languages (at the expense of their famous purity). Prolog for example has many implementations of both modular (basic implementations such as *packages*) and object oriented programming as seen in Table 1.

Table 1: Modularity in some Prolog versions [5]

Platform	**Ver.**	**Modules**	**Object O.**
SICStus Prolog	4.3.5	√	√
ECLiPSe	6.1_226	√	
SWI-Prolog	7.4.2	√	
Ciao	1.14.2	√	√
BProlog	8.1	√	√
JIProlog	4.1.5.1	√	
LPA-PROLOG	6.0		√
Visual Prolog	7.5, Build 7502		√
XSB Prolog	3.7	√	
YAP-Prolog	6.2.2	√	

Along with these implementations there has also been attempts to generalize Prolog onto a wider mathematical domain. Λ-Prolog was introduced as an extended Prolog and it included provisions for higher order functions, λ terms, higher order unification, polymorphic and abstract data types [6]. The language supported modules through its own dynamic logical foundations [7], however practical software engineering was still a challenge since it was too abstract when compared with standard Prolog. Although standard Prolog is based on Horn clauses, λ Prolog is based on higher order hereditary Harrop formulas, providing higher degree of abstraction nevertheless having to be much more expensive to be executed.

B. Assessing Alternatives

One of the most traditional AI approaches involves creating (usually a hierarchy of) alternative solutions to a problem and then performing a heuristic comparison and search for the best option. This create/search routine may be repeated in a (probably recursive) loop until the desired solution is reached. This technique is very suitable for tasks like game playing or theorem proving. It also works well for solving a puzzle, finding way in a labyrinth, or finding a certain path that obeys certain constraints on a graph.

These problems require two things. Firstly, creating, traversing and managing trees should be made easy; secondly in each node of a tree the programmer should be

able to create a new environment of truths and variables. Which means that each node ought to have its own local namespace, logical statements and maybe rules.

C. Working with Patterns

At the two extremes of symbolism spectrum there are computer vision and formal language processing both of which are very popular fields of study. Formal languages are fully symbolic whereas images are not symbolic at all. On this spectrum, in between the two extremities there are natural languages, signals, voice and sensor data.

Symbolic and non-symbolic data create different difficulties when they are subject to pattern recognition. At the symbolic end, the formal languages are very easy to cope with. In compiler design for example patterns are detected very easily through a process called parsing. Even a language like C can be very effective for coding a parser, since nothing more than recursion is required for top-down parsing. Parsing natural language is not that straightforward and a bottom up parsing approach together with backtracking may be necessary. In this case Prolog would be a better choice.

Non symbolic data is much more difficult to interpret. In relatively basic applications features are extracted and evaluated. As problems get bigger and more complicated, machine learning techniques become dominant.

D. Fuzzy or Soft Programming

There are many post-modern challenges to classical programming such as soft-computing, multiple criteria decision taking and machine learning. There are also a lot of hybrid applications of these techniques. One important source of domain information is interaction with a human expert. *Fuzzy logic* makes this possible by creating a bridge between human mind and mathematics. *Fuzzy logic* is a kind of many-valued *logic* in which the truth values are considered to be real numbers between 0 and 1 (inclusive). It is used for representing the concept of partial truth, where the truth value is somewhere between true (1) and false (0).

Programming by using fuzzy data was an immediate requirement and attempts were made for implementing fuzzy logic in Prolog. Prolog-Elf [8], Fril Prolog [9] and F-Prolog [10] are examples of fuzzy Prolog implementations. Unfortunately, they are not easy to use and have never got very popular. A fuzzy Prolog is usually based on a concept of fuzzy resolution [11] which is said to be a *generalization* of the classical predicate resolution which standard Prolog relies on (the term "*generalization*" is a little tricky for complex phenomena like resolution which in the Prolog context sometimes works by failing).

E. Machine Learning

Image processing is a very typical success story of the machine learning paradigm. Medical imaging is a group of cooperating technologies that are indispensable for effective detection of diseases and anomalies. The digital output provided from these imaging sources are suitable for computer processing and employment of AI. The AI assistance cumulates in two fields: Computer Aided Detection (CADe) and Computer Aided Diagnosis (CADx) [12].

Theoretically there is no obstacle for a sufficiently complicated neural network system to exceed the success of a human expert in a well-defined pattern recognition task. With the emerging machine learning technologies more success stories may be expected from any problem that can be classified as pattern recognition. In fact, if the success is not achieved yet, it can be expected after the development of correct models or reaching a certain hardware capacity requirement.

In neural network technology a layered and densely connected network is trained with a large input and (corresponding) output set. Afterwards the network acts as a function that creates similar output for similar input. Of course this is a very rough description of the technique which has some more parameters and details.

One important variation is the employment of convolution transformations that are applied in image recognition systems, as a preprocessing of the raw image in order to recognize features that are related to the shapes in the image. The architecture of each project takes place as an amalgam of little transformations that form a directed network starting from raw data and ending up in a result.

Another variation is called the residual network architecture which is preferred for solving a technical difficulty called the *vanishing gradient problem*. It occurs in networks that are too deep and can be imagined as a confusion that originates from the inherent chaotic behavior of big interacting systems. Residual networks as applied in [13] solves the problem by applying fewer layers but a more densely connected network (so dense that it violates layering) with connections that overreaches layers.

There are other network topologies too like recursive neural networks, recurrent neural networks (RNN) and long short term memory type networks that are used for natural language processing. These networks can store information between cycles, so that they can respond to certain sequences of input.

Whatever the inner details may be, the components of a machine learning project usually are structured like the example in Fig. 1 which is a preliminary design for an ongoing project about automated triage by using three dimensional CT (computerized tomography) images of head scans.

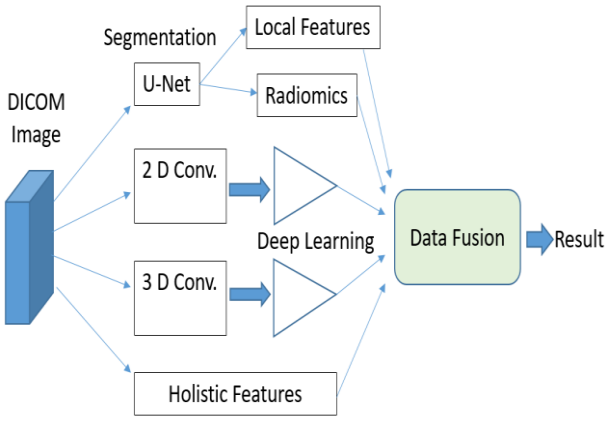

Fig. 1: Automated triage project architecture

As can be seen in this architecture the whole process can be modeled as an application of the *data pipeline computing* paradigm. Since there is just one entry and one exit, it is also possible to approach the domain from a functional

(LISP-like) point of view. From modelling point of view, the *transformation* concept in model driven software development [3] has a direct potential application here. Going a little deeper into this case one would notice that supporting rules of logic (or more generally fuzzy logic) in the design language may provide some benefits in detailed design. For example, the DICOM image provided as input has approximately 250 layers most of which are normal (healthy) images. However, the number of normal images is unimportant if there is a single very serious indication of an injury or a malignant disease in any one of the 250 pictures. Therefore, for meaningful assessment there should be a *disjunction* operation in between each of the 2d prediction results that are obtained by processing the layers one by one. The result of the whole scan is malignant, *if image1 OR image2 OR …image2*50 is malignant (or risky). Fuzzy computation of risk requires one of the *fuzzy triangular conorms* to be used. The design of image recognition (or similar) projects therefore requires logical connectives that bind the various sections of data together. Fuzzy logic is a better alternative than plain logic since the system uses non-discrete values everywhere and a loss of information is not tolerable.

III. THE AI META-MODEL

Firstly, we should note that a *meta-model* does not aim solving any of the problems of the domain that it refers to. For example, it does not claim to show a better way to perform clustering, classification or search. That is the job of the semantic dimension of developing a new paradigm. The *meta-model* deals with the syntactical issues, which is in a way solving the *accidental* problems of program development (as the term was used by Brooks [14]) rather than *essential* problems. It is a bit like focusing on the shelves of a library before considering the book collection. It may seem unnecessary as a first move but in fact it can make the subsequent moves to be more structured and to the point.

The meta-model that will be partially presented here employs UML class diagrams which should be perceived on the conceptual level.

There is also one diagram by which *compatibility* has been described by using an extension of UML based on a specific meta-model, called the *Mega-model* which was developed by Favre [15] so as to facilitate the definition of inter-model relations. The *mega-model* introduced by Favre can be described as: Every model is a system with a special mega-relation (called RepresentationOf and denoted with the Greek letter μ) with another system and any system can have any mega-relation freely with any other system.

The Mega-model contains a small set of relations marked with Greek letters χ, μ, δ, ε and τ. In this language χ means *conforms to*, μ means *representation of*, δ means *part of*, ε means *element of* (for sets) and τ means *transformation*. We have extended this notation by redefining μ as symmetry (or similarity) and adding Σ to denote *signification* (*signifies*). This notation [16] is necessary for moving slightly towards the semantic side and defining modeling relations as well as semiotic relations. These Greek letters are also very useful inside the UML diagrams.

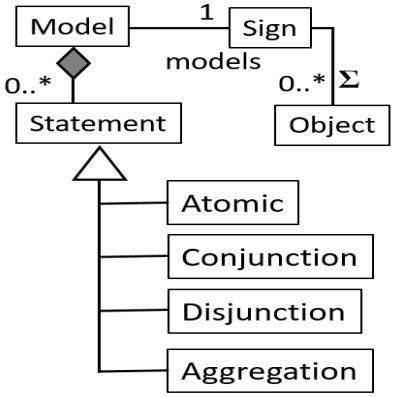

Fig. 2: Model interior structure

Fig 2. shows an internal definition of a working model of an existing object in UML class diagram syntax. These are special models like the *map of France* (special to the country of France) or a record of Student *Smith Jones*. The model is composed of two parts. One is a set of logical statements about the subject (which is an individual) and the second part is a sign that points at that individual.

Map of France most probably has a title on it that informs the user that it is a map of the country called France. Without this title (unless the user recognizes the map somehow) the map would be useless because it would fail to make any logical statements without an anchor that links the statements to any known references. So the title serves as the sign that points at the object (individual) of the map, and the rest of the map provides the statements.

Same thing is true for the database record for *Smith Jones*. The first two fields of the record are Name and Surname which together constitute a sign that points at the individual. (Sometimes this role is duplicated as *StudentNo* for example.) The rest of the record provide the statements. Such as "*his age is 20.", "his GPA is 3.1* and so on.

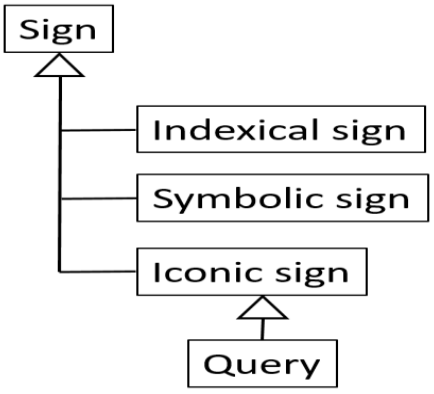

Fig. 3: Sign types

In Fig 3, an inheritance hierarchy is given for types of signs. *Indexical*, *Symbolic* and *Iconic* signs are parts of the semiotic literature [17]. An *indexical sign* is a sign that signifies its object by physically pointing at it; that is through an unmistakable physical connection that does not require any subjective interpretation. A good example from the world of software development could be a C pointer (which is hardware interpreted thus solid enough).

A symbolic sign is one that needs an interpretation to be

able to signify its object. This interpretation is purely symbolic, in the sense that there is no need for a physical connection or resemblance between the sign and the object. In fact, the connection is purely in the mind of somebody who interprets the sign (could as well be a machine). The connection is arbitrary and hypothetical. A good example is the way a letter in a phonetical alphabet signifies its sound. The connection between letter "*A*" and its sound can only be constructed by someone who has this *assumption* in his mind. The identifiers and keywords in a program are good examples of symbolic signs.

The third kind of sign is an iconic sign, which signifies its object by resemblance (by being like it). A picture of something for example can serve as an iconic sign for it.

There are also hybrid cases of signification. A traffic sign with the picture of an animal on it can be regarded as *iconic* and *symbolic* at the same time while another traffic sign with an arrow on it could be seen as *indexical* and *symbolic*.

A query can be regarded as an iconic sign since it binds its object through a physical comparison. The query contains (not given in the figure) a domain (like a filename for example) and a pattern to search for, within that domain. This way it signifies multiple entities. Support for fuzzy logic would require partial signification too. A query such as "*the tall people in the class*" should return *pointers of different strengths* for a number of students. This issue is referred in *fuzzy sets* but not studied within the context of semiotics.

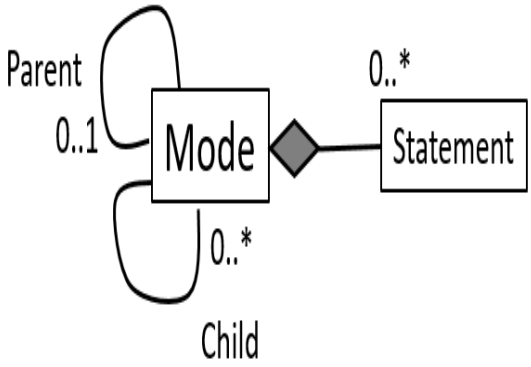

Fig. 4: Modality tree

The construct called *Mode* constitutes a tree (as can be observed from its cyclic links in Fig 4.). This can be a tree for alternatives for game playing. It can also support an inheritance tree. Each mode contains a set of logical statements that are valid within its own environment. *Logical statement* of course is a very flexible construct that can be used in order to represent almost anything.

Child modes can also reuse or override the information inherent from their parent. With proper semantics this element can be used for controlling *scope* and *environment* of information.

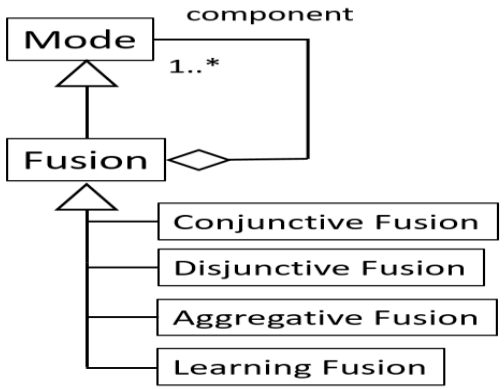

Fig. 5: Data fusion and its types

In Fig 5. the meta-model construct *Fusion* is defined as an extension of the construct *Mode*. As well as information that it inherits from its parent *Mode* (which are thought to be meta-information), *Fusion* also has a set of components that are other Modes. This works just like multiple inheritance. Fusion takes potentially conflicting information from its components and merges them in a way that is proper for the application. Logical connectives are added to the diagram as types of *Fusion* and *Learning Fusion* has been added as a forth type. This classification is of course very shallow and there are a number of statistical formulas used for data fusion. These techniques are omitted since they are implementation details.

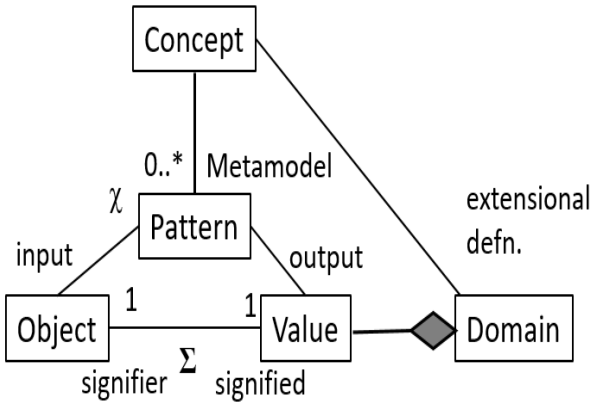

Fig. 6: Concept as an interpreter

In Fig 6. the relation between *Concepts* and *Objects* is depicted. *Concept* is similar to a class in *object oriented paradigm*, but it is more abstract. A *Concept* may have various *Patterns* as alternative *Metamodels* (different ways to represent the same concept). *Patterns* can be thought of as a set of relations that hold between a set of variables. The χ label represents a *conforms-to* relation between the Object and a *Pattern*. The *Pattern* of the *Concept* can read the *Object* and create its meaning as a *Value*. What is not shown in the figure is that the *Object* is an instance of the *Concept* (not to make the diagram too busy). *Value* belongs to the *Domain* which is the extensional definition of the *Concept*. The Σ relation between the Object and the Value is the basic signification relation. The whole diagram is about the case that, an instance of a *Concept* can point at a *Value* within the *Domain* of that *Concept*, through the interpretation of (a

meta-model related to) that *Concept*. The diagram simply describes the relation between *type* and its *instance*.

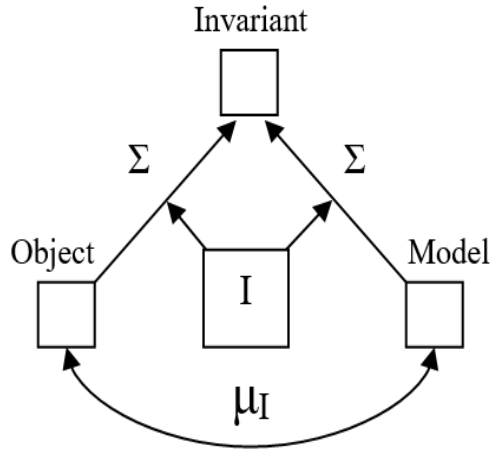

Fig. 7: Compatibility between entities [16]

In Fig 7, is from our former work [16] and it uses our extended *mega-relations*. This figure shows the idea behind the μ relation which indicates similarity. Two things are similar if they point at (as signs) the same entity (called *Invariant* here) through the same interpreter *I*. The two entities are arbitrarily named as *Object* and *Model* although the relation is symmetrical. It is true that when two things are similar, they can be used as models of each other. Naming one as the *Model* is just an intentional role that cannot be attributed to an objective inherent quality. Therefore, instead of the *modeling relation*, as Favre has suggested [15], *Similarity* is the preferred label here.

Another point of attention is that μ is context dependent and it depends on a certain interpreter *I*. Two things are equivalent only with respect to a certain point of view. We can imagine that *Object* is *three apples*, Model is *three oranges* and *I* is simply the counting operation. In this case the Invariant is simply the number 3. *Object* and *Model* are said to be similar with respect to the counting operation.

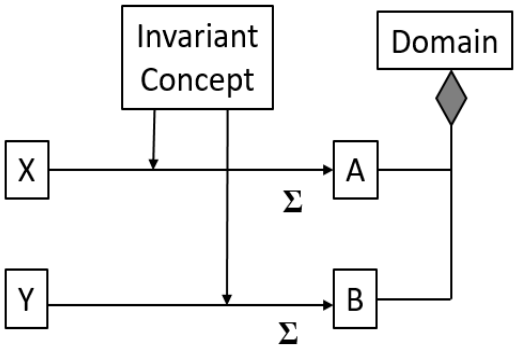

Fig. 8: Two values on a Domain.

In Fig 8, the *Invariant* acts as an interpreter now. It is not shown in this diagram that the *Invariant Concept* is signified by both *X* and *Y*, however it acts like their data type. This means that *X* and *Y* belong to the same data type (they are compatible). Being their data type, the *Invariant Concept* can interpret these two values leading to two positions on a common domain namely *A* and *B*.

After defining these variables, it is possible to place a foundation for fuzzy values. It can be stated that:

Similarity (X,Y) = 1 / Distance (A,B)

or rather its context dependent version:

Similarity (X, Y, IC) = 1 / Distance (A, B, Domain)

where *IC* is the *Invariant Concept*.

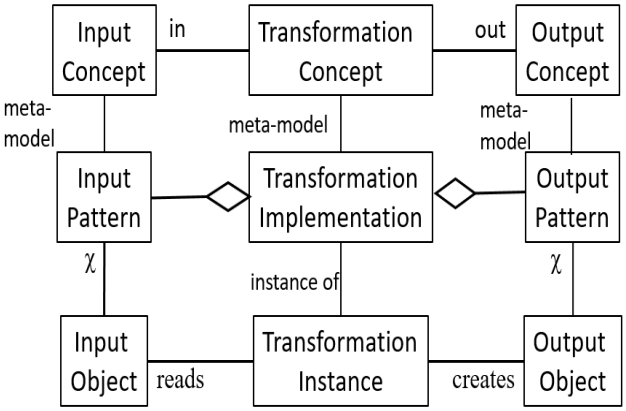

Fig. 9: Transformation

Fig. 9, depicts a grid-like diagram that demonstrates the general components of the transformation of data. The three nodes at the top constitute the *interface* part of this concept and that is the only part we would worry about, in the case of encapsulating machine learning applications for example. The two layers beneath that layer, represent the implementation of the transformation.

Transformation as a *Concept*, defines input and output as *Concepts* (classes), while the implementation of that transformation contains meta-models for *Input* and *Output*. Input meta-model shows how to read the *source* and *output* meta-model shows how to write the *target*. The Transformation Instance is the actual (dynamic) transformation that takes place between the objects.

Supporting an image recognition learning system requires further details such as degree of *abstraction* and *generality* of data as properties of the input and output concepts. When designing neural network topologies these parameters take action. For example, a wider network is said to support more specific information (less general) while a deeper network supports more abstract information (less concrete).

IV. CONCLUSION

Our attempt in this work has been a bit like trying to develop object oriented paradigm starting from UML, so it is a confusing task that is hard to position within the big picture. However, the AI domain is inherently messy and with the rise of machine learning, software development is becoming an ad-hoc endeavor while research is taking the form of communicating dinner recipes.

AI should be engineered in a structured, integrated and multi-paradigm fashion. In this work a meta-modeling approach was advocated as a first step for the unification of the AI domain. In the short term this may support model driven development of AI applications and in the long term it may constitute the first step for laying the foundations for a grand AI paradigm.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] P. H. Winston, "Artificial Intelligence", Addison-Wesley, Reading, Ma, third edition, 1992.

[2] J. Wang, (2021, January) "Symbolism vs. Connectionism", A Closing Gap in Artificial Intelligence", Jieshu's Blog, 2017, Available: http://wangjieshu.com/2017/12/23/symbol-vs-connectionism-a-closing-gap-in-artificial-intelligence/

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

[4] A. B. Arrieta, *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI", Information Fusion, Vol 58, June, Elsevier, 2020, pp. 82-115.

[5] M. J. Kemtongue, "Modularization Challenges in Prolog: What to Divide and Conquer in AI", in Intelligent and Fuzzy Techniques in Big Data Analytics and Decision Making, Springer, January 2020, pp.330-337.

[6] G. Nadathur, D. Miller, "An Overview of Lambda Prolog", Technical Report, University of Pennsylvania, Scholarly Commons, Department of Computer and Information Science, June 1988, unpublished.

[7] D. Miller. "A proposal for modules in lambda Prolog", in Proceedings of the 1993 Workshop on Extensions to Logic Programming, volume 798 of Lecture Notes in Computer Science, 1994, pp. 206–221.

[8] M. Ishizuka, N. Kanai, "Prolog-Elf Incorporating Fuzzy Logic" in: IJCAI 9, vol. 2, 1985, pp. 701-703.

[9] J. F. Baldwin, T. P. Martin, B. W. Pilsworth, "Fril: Fuzzy and Evidential Reasoning in Artificial Intelligence", John Wiley and Sons, 1995.

[10] D. Li, D. Liu, "A Fuzzy Prolog Database System", John Wiley & Sons, New York, 1990.

[11] R. C. T. Lee, "Fuzzy Logic and the Resolution Principle", Journal of the Association for Computing Machinery, 19(1), 1972, pp. 119-129.

[12] J. Gao, Q. Jiang, B. Zhou, D. Chen, "Convolutional Neural Networks for Computer-aided Detection or Diagnosis in Medical Image Analysis: An Overview", Mathematical Biosciences and Engineering, 16(6) 2019, pp.6536-6561.

[13] J. J. Titiano, M. Badgeley, J. Schefflein et al., "Automatic Deep-neural-network Surveillance of Cranial Images for Accute Neurological Events", Nat Med 24, 2018, pp.1337-1341.

[14] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, IEEE Computer Society Press, 20(4), 10-19, April, 1987.

[15] J. M. Favre, T. Nguyen, "Towards a Megamodel to Model Software Evolution through Transformations", SETRA Workshop, Elsevier ENCTS, 2004.

[16] A. Egesoy, "Context-Aware Formalization of Inter-Model Relations", International Journal of Computer and Information Technology (IJCIT), Vol:3, No:6, November 2014, pp.1461-1467.

[17] D. Chandler, (2021, January) "Semiotics for Beginners", Aberystwyth University, Available: http://visual-memory.co.uk/daniel/Documents/S4B/

ABOUT THE AUTHOR

Ahmet Egesoy (PhD in Computer Engineering) is an instructor and Assistant Professor in Computer Engineering Department of Ege University Izmir, Turkey. Research interests include object-oriented programming, design patterns, model-driven software development, artificial intelligence, programming languages, programming paradigms, philosophy of the language, semiotics and knowledge representation.