Abstract—The increase of the development process of complex software system leads to make the research and the practice for Model Driven Engineering (MDA) progressed over the last decade, the model driven architecture (MDA) defined at the OMG (Object Management Group),"Software Factories" proposed by Microsoft and the Eclipse Modeling Framework, are appears to raise the level of abstraction using models as information storage. Nowadays, the great number of meta-models and models leads to use MDE in various domains and make deferent models similar for the same application, the problem here is how to get similarities between these formalism or formalism instances?

In this paper, we propose an approach that can detect the mapping between two architectures by making model transformation using transformation language from architectures as input to a directed labeled graph as output, our approach is based on similarity flooding algorithm used in the fields of schema matching and ontology alignment.

Keywords-component; Model Driven Engineering, Model Transformation, Model matching, ATL, Oriented labeled Graph, Similarity flooding algorithm.

I. INTRODUCTION

Modeling and representation of software architectures are the main phases of the development process of complex software systems [1], the representation of architectures is based on the concepts of component (loci of computation), connector (loci of communication), and configuration (arrangement of components and connectors, and properties of that arrangement) in order to describe the structure of the system at a higher level of abstraction than objects or lines of code. This representation provides several advantages over the life cycle of software [2].

Component has always been considered to be the fundamental building blocks of software systems, the ways the components of a system interact are determinant for establishing the global system properties that emerge from the way the individual components are interconnected. Hence, class design entities as well, and architectural connectors have emerged as a powerful tool for supporting the design of these interactions [3].

With the great expansion of MDE (Model driven Engineering) in this days, it leads to create a large number of meta-models, for various domain, some for the web, Oriented Object Programing (OOP) and databases,... and with this diversity, tools are developed to make MDE more useful, but every one of these tools conforming to special meta-model, for example, code generator, model transformation tools, and graphical editor. Of course, several meta-models can describe the same kind of applications, so we can look forward to create a tool work on kinds of applications, the idea here is to adapt a given model conforming to other meta-models in the same kind of applications, to the meta-model that we worked with, for example there are several meta-model to describe the class model, and a lot of tools are dedicated to manipulate it, but everyone has its meta-model, So, the problem here is how to make a tool can automatically be adapted a Model conforming to other meta-models, and be conforming to the meta-model supported by this tool?

Already this problem is solved using model transformation, with manual written of the transformation rules, these codes are not difficult to written but they are large and numerous, this make a heavy work to developer to make it.

For making the task issuer hard we propose to generate automatically matching between the deferent meta-models and models.

The matching operation establishes mappings between two formalisms or formalism instance, this can be used to generate the transformation rules, this operation take in input two meta-models or models and made as result the correspondence between the model elements as XML schema, Database Schema or class diagram model...

After this introduction, the remainder of this Article is organized as follow: firstly we discuss related works, we chose the related work in schema matching and we based on the works which are closest to our approach, and in Section 3 it is pearly theatrical, we spot the light on these concepts, MDA, MDE, model transformation and matching. In Section 4 we present our proposed approach and our objective, section 5 the part of our contribution, is the way proposed of encoding a given architecture into directed labeled graph and finally, in Section 6 an illustrative study case is presented.

II. RELATED WORK

Matching techniques have been studied intensively in many various field, one of those techniques is schema matching in
data [4] it was precursor whereas, but now the research spot the light on the techniques ontology matching[5][6], there is also ontology matching is very active [5][7]. The matching operation focuses on classes, data type, and enumeration in paper [8]. The proposed class matching algorithm iterate until a fix-point is reached on evaluation if two classes match. Characteristics including attributes, references and relationships are then used: if the number of matching characteristics is grater the threshold average, the two classes are considered as similar. The used name, type and graph matching similarity, for data types, the possibility of representing a value of a type (e.g: int) by another type is used. For enumeration, the numbers of identical literals are compared.

In [9], architecture is proposed for semi-automating the generation of model transformations. A weaving meta-model describes the kinds of links generated by the matching engine and current transformation patterns. Matching is based on element to element and on structural similarities. Structural similarities computation is inspired by similarity flooding approach.

The authors of [10] propose an approach to generate automatically an alignment between two similar meta-models, and then generate the code of a model transformation by this alignment, this approach consisting in three phases:

First step: transformation of meta-models to directed labeled graph,


Third step: generate the alignment between meta-models.

And several research projects close to our approach have studied the idea of mapping between different meta-models [12] [13] [14]

### III. THEORY OF MODEL MATCHING

Modeling is at the heart of MDE [15]. it raises the level of programming from code to model in all phases of software development (conception...). Nowadays, MDE needs new approaches and solutions to their imperfections, one of the most interested topic for the community of MDE is model matching. Here we put it under light and understood their concept.

MDA (Model Driven architecture) is based on four layers [1]: meta-meta-model, meta-model, model, and the information (i.e. an implementation of its model). In this approach, everything is a model or a model element. In level M0, a real system is represented by model in level M1, and a model in level M1 conforms to a meta-model in level M2. These two important relationships of MDA are discussed in [16][17].

#### A. Model transformation

It is the manual or automatic generation of a target model from a source model, according to a transformation definition. From a conceptual point of view the explicit distinction between mapping specification and transformation with the MDA philosophy, i.e., the separation of concerns. Moreover mapping specification could be associated with different transformation definitions, where each one is based on a given transformation definition meta-model.

### B. Mode matching

Matching A between two formalisms S1, S2 composed respectively by two groups of elements E1, E2 is a binary relation, \( A \subseteq E1 \times E2 \). Couples (a, b) \( \in A \) are called mappings, for each element \( a \in E1 \) there is max couple (a, x) \( \in A \) and the same propriety is verified for each element b\( \in E2 \) here the alignment called 1-1, else we called it n-m [18]. According to [19], the process alignment, as illustrated in figure 1, can be separated to three phases:

- Conversion of the formalisms to internal format of the process alignment,
- Calculation of similarities for each mapping (Process alignment),
- Selecting a subset of mappings.

#### Figure 1. Process of the alignment.

### IV. OVERVIEW OF THE APPROACH

In this paper, we propose to make model transformation from an input architectures conforming to the meta-model C3 [20] to a output directed labeled graph model and then applying similarity flooding algorithm [11] to generate the alignment between the two architectures by using ATL [21] (ATLAS Transformation Language), We chose ATL cause is a hybrid model transformation language that allows both declarative and imperative rules. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models. Developed on top of the Eclipse platform, the ATL Integrated Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aims to ease development of ATL transformations. Our objective is to make the step of transformation from architectures to directed labeled graph easy and automatically and get the alignment and use it to generate automatic transformation rules.

We chose the architectures as input because there is similar work done by Falleri et al. [10] but the input is class model conforming to meta-model.

There are many other approaches working on the model matching we can classify them in 2 main categories: schema level approaches that produce matching using schema and instance level approaches that use instance of schema to produce the alignment. Here we chose the Similarity Flooding Algorithm [11]; it is an existing approach concerning the matching problem, for two main reasons. First these applications takes as input directed labeled graphs, this operation will be easy by make it automatically with tool model transformation. Second similarity flooding algorithm is based on many phases that we can implement it using model
transformation languages in the next section we present this algorithm and how we can implement it using ATL as Model Transformation Language.

Our work, as showing in Figure 2, is composed on two main phases to get the alignment; first we transform the Architecture to labeled oriented graph using the configuration that we propose in the next section, these transformation are implemented using ATL, after that we get the alignment using similarity flooding algorithm.

A. Similarity Flooding Algorithm

Matching or finding similar element in to schema or model play a key role in the automatic generation of transformation rules from model to other, Melnik et al. [11] present a matching algorithm named ”Similarity Flooding Algorithm (SFA)” based on fix-point computing, this algorithm takes as input two labeled oriented graphs and produce the mapping between their elements. This SFA has five main phases and we note that their phases are simple transformation from models conforming to the labeled oriented graph meta-model to models conforming to the same meta-model:

a) Create the connectivity graph: the goal of this phase is to create possible associations between nodes,

b) Create the propagation graph: a new graph was created in this phase based on the connectivity graph by adding opposite arc to each arc if doesn’t exists; after we change the arc labels by weight. Melniketal. [11] propose many ways to compute it,

c) Compute the initial similarity values: in this phase a similarity value between 0 – 1 assigned to each node and here Melnik et al. [11] propose many ways to calculate this value.

d) Propagation the similarity value: here SFA calculates the similarity values, for this it uses an iterative algorithm [11],

e) Mapping filter: this phase deletes many mapping with many criteria, the matching is the rest mapping.

In our work all these phases are implemented using ATL as the illustration in figure 3.

V. FROM ARCHITECTURE TO LABELED ORIENTED GRAPH

As we said in the previews section we propose a mapping (configuration) from C3 Architecture to labeled oriented graph, this transformation is the most important phase in this work because it leads to create the inputs to SF algorithm, here we try to get the best configuration to get the best results from the SF algorithm because the structure of graph is important when this Algorithm compute the values of similarity, in the following, we explain how we build the graphs from the architecture.

We propose a correspondence between C3 elements and labeled graph elements, this idea is to detail as we can the model. For this we put the following rules:

- Each port is a node and the node’s names are the same as port’s names,
- Each component is node and the component’s names are the same as node’s names,

![Figure 2. Overview of the proposed approach](image)

For labels of arcs we defined a list which contains the deferent relations between elements and here we present them:

A. Type: label for arcs between the elements and their types, C3 language have 4 principal first class entities showing in his meta-model. That elements we called them in our approach types of element, In C3[22] the first class entities are:

1. Component: represent the computation elements and data storage for the software and each component can have one or more ports,
2. **Connector**: represents interactions among components for supporting their interactions, and as well as component each connector can have one or more ports,

3. **Ports**: every element may have their ports with which it connect with its outside environment and publish it requirements services as well as its provided services, from here C3 defined two types of ports required ports, in Figure 4 we present them as ReqPort, and provided ports, in Figure 4 we present them as ProPort,

4. **Configuration**: represent a graph of elements components and connectors that describe how they are fastened to each other element, configurations may be having port as well as components and connectors.

B. **Service**: As presented in [22] the architectural elements connect with their environment via ports and share their services from Provided ports to required port, from here we put the label of the arc between Provided ports and required port and call it service,

C. **Owner**: As showing C3 description each architectural Element may have their ports, So, it is a relation between port and its element, we put it as an arc in the produced labeled oriented graph,

D. **Contain**: label for arcs between architecture elements and the configuration that contain them.

We illustrate our proposed correspondences from C3 to labeled oriented graph in Figure 4.

VI. CASE STUDY

To illustrate our approach we present this example shows in the figure 5, two architectures conforming to meta model C3, we are going to show clearly the main transformation rules through the three steps, firstly from architectures to directed labeled graph and second from this last to Connectivity graph, third from Connectivity graph to propagation graph and finally getting the Alignment Values.

Our application is developed based on model transformation from the beginning, so, it’s made according to C3 meta-models and labeled oriented graph meta-model, that are presented them in Figure 6.

As first step in formalization of the meta-models we need to use Eclipse Modeling Framework [23] for describing the metamodels as Ecore diagrams, this diagram specifies how to create a well-formed architecture (compliant to the C3 meta-model), and any well-formed architecture can be expressed as labeled oriented graph using the transformation rules that we developed using ATL according to our approach.

The most important in this paper is ATL rules and how we make the automatic transformation from architecture to graph, here we present an ATL module contain the rules that can make this phase, using ATL we just implement the rules of correspondence that we presented them in the theoretical section of our approach. The source code in Figure 7, describe how each component is transformed to a node in the target model. The idea is to initialize a new node the name of the related component. In the same rule we create an arrow from the actual node to the

Type node, with label ‘TYPE’, after that for each related port, we create an arrow form the node related to this port to the node related to the component with label ‘Owner’.  

![Figure 4. Transformation of architecture to Labeled Oriented graph.](image-url)

Figure 4. Transformation of architecture to Labeled Oriented graph.

![Figure 5. (a) First Eshop architecture, (b) Second Eshop architecture.](image-url)

Figure 5. (a) First Eshop architecture, (b) Second Eshop architecture.

The application of the transformations on the given example, that’s given the graphs showing in the Figure 8, our implementation is worked on XMI[24] models, so the graphs in Figure 8 are an XMI Code conforming to Labeled graph model, Now we are working on a graphic plugin for this work, using GMF[25](Graphical Modeling Framework).

The second phase of our work is the implementation of the
similarity flooding algorithm using model transformation language. As we said in the previous sections, if we look to the SF algorithm phases, we note that these phases are just transformations graphs from two input graphs to connectivity graphs and then to propagation graphs. We implement all these transformations using ATL.

With application of this algorithm on the given example and after many iterations depend on the value of epsilon $\varepsilon$, in our case $\varepsilon = 0.05$ and after filtering according to SF filtering methods proposed in [11]. The method we use to filter the produced multi-mapping is called SelectThreshold[11], the results are shown in Table 1.

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>R1</td>
<td>0.91</td>
</tr>
<tr>
<td>P1</td>
<td>P1</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>R2</td>
<td>0.87</td>
</tr>
<tr>
<td>P2</td>
<td>P2</td>
<td>0.87</td>
</tr>
<tr>
<td>Product lookup</td>
<td>Product lookup</td>
<td>0.96</td>
</tr>
<tr>
<td>Inventory System</td>
<td>Inventory System</td>
<td>0.61</td>
</tr>
<tr>
<td>Order System</td>
<td>Order System</td>
<td>0.64</td>
</tr>
</tbody>
</table>

**Table 1. Results of the algorithm application.**

To improve our approach, we make more experience on various architectures alignment. We evaluate our results with common metrics from the information retrieval field [26]. For each alignment we compute the precision with the relation presented in Eq.1. The results of our experiments are shown in Figure 9.

\[
\text{precision} = \frac{\#Correct\_found\_mapping}{\#Total\_found\_mapping} \tag{1}
\]
In this paper we have presented a way to make transformation from two architectures conforming to meta-model C3 as input into directed labeled graph and then make the alignment, by using the transformation language ATL, We introduced. Several transformation rules of ATL, the execution in order of these rules drive us from two architectures as input passing by the directed labeled graph and five steps of the similarity flooding algorithm to an output the final value of similarity.

Our approach increases the quality of alignment and accelerates the development of the transformation. The main goal of our idea is make the steps from architectures pass by directed labeled graph to alignment easier and automatically using model transformation.

REFERENCES


