Towards Generic Semi-Automatic Transformation Process in MDA

Lamine Lafi 1, Zaki Brahmi 2, Jamel Feki 3, Slimane Hammoudi 4

1 University of Gabes, Laboratory Miracl, Tunisia, 2 University of Sousse, Laboratory RIADI, Tunisia, 3 University of Sfax, Laboratory Miracl, Tunisia, 4 ESEO, Ecole Supérieure de l’Ouest Angers, France.

Abstract—The Model Driven Engineering (MDE) has been proposed to support the development, maintenance and evolution of software systems. In this context several approaches for transformation models have been proposed in the literature. They suffer from two major limitations: i) they have been tested on homogeneous metamodels (e.g., Ecore, UML, Minjava, Kermeta...), and ii) they use common notions and concepts, such as modeling, matching and processing models. In some approaches they do not distinguish between matching, mapping and transformation. This leads to confusion between these operations. Indeed, it is well recognized that model transformation is at the heart of approaches MDE and it is derived from the mapping model, itself obtained after performing a matching process.

In this paper, we propose, in a first part, an extended architecture to semi-automate the process of transformation models. It is the separation of concerns by distinguishing the operations matching, mapping and transformation taking into account several factors (e.g., semantic resources, pre-matching efforts and post-matching efforts...). In the second part, and to test the heterogeneity of metamodels that will be taken as input by matching techniques, we introduce two new types of metamodels representing respectively the multi-agents system platform JADE, and the BPEL language for the web services orchestration. To validate our proposal, our architecture is implemented on a matching technique which details the various steps leading to a semi-automatic transformation process.

Keywords—Matching; Mapping; Transformations; Architecture; Metamodel JADE; Metamodel BPEL.

I. INTRODUCTION

Recently, the OMG has proposed the Model Driven Architecture (MDA)1 [1] to support the development of complex and large software systems providing an architecture with which systems: can evolve for meeting new requirements, old, current and new technologies, can be harmonized, business logic is protected against the changes in technologies, and legacy systems are integrated and harmonized with new systems.

In this approach, models are applied in all steps of development up to a target platform, providing source code, files of deployment and config, and so on. MDA proposes architecture to address the complexity of software development and maintenance which has no precedents. It claims that software developers can create and maintain software artifacts with little effort. However, before this becomes a mainstream reality some issues in MDA approach need solutions such as mapping specification and transformation definition [2].

In the literature, several issues have been studied and subject to intensive research, e.g. modeling languages [3], [4], model transformation languages [5] and mapping between metamodels [6]. Among these issues, model transformation languages occupy a central place and allow the definition of how a set of elements from a source model are analyzed and transformed into a set of elements in a target model. The discovery and generation of transformation patterns models is closely related to metamodels matching techniques. However, these approaches have some drawbacks: most solutions cannot be applied to models conforming to different metamodels; metamodels are models that describe the structure of models, in some case there is no distinction between the operations matching, mapping and transformations. This makes it difficult to decompose, distinguish and to customize these operations. In addition, there is no support for metamodels with different type. Hence, the heterogeneity of metamodels that will be taken as input is not supported such as metamodels representing respectively developed on the platform for multi-agents system Java Agent DEvelopment Framework (JADE), and the Business Process Execution Language (BPEL) for web services orchestration.

In this paper, we propose to push the semi-automation process one step further by using matching techniques [7], [8] to semi-automatically generate mappings between two metamodels. The contributions of this paper are two-fold: First, we propose a new architecture for the transformation process in the context of MDA. This new architecture extends the architecture presented in [9]. In order to introduces matching and mapping as first class entities in the transformation process, represented by models and metamodels. We use the term mapping as a synonym for correspondence between the elements of two metamodels, while transformation is the activity of transforming a source model into a target model in conformity with the transformation definition. The objective is to provide a precise definition of the operations of matching, mapping and transformations taking into account many factors (e.g.,
semantics resources, pre-matching efforts and post-matching efforts...). Second, we develop a methodology which illustrates the different steps of the semi-automatic transformation process. To validate our architecture, we propose two new metamodels. They will be used in studying supports all the main steps of our architecture, from matching between metamodels to the generation of transformation rules.

This paper is organized as follows: Section 2 positions our work and motivates our contribution for metamodel transformations, discusses key related works and situates our architecture relative to them. Section 3 presents an extended architecture for a semi-automatic transformation process and discusses the matching, mapping and transformation as three important components in this process. Section 4 presents our illustration for generating transformation definition in MDA between two metamodels using BPEL and JADE metamodels. Finally, section 5 concludes our work and presents future perspectives.

II. RELATED WORKS

In order to facilitate semi-automatic mapping generation and improving metamodel matching for model transformation development, [11] proposes the integration of existing matching approaches in a common framework and the reuse of matching results by applying different metamodel specific matchers (e.g., an instance matcher, a graph matcher, an annotation matcher, a data type matcher...) based mainly on model transformation type classification.

Trends do not cease to evolve, but we will now speak about a trend which is more recent (model transformation by examples-MTBE) for assisting the development of transformations is inspired by the approaches of programming by-example or by-demonstration [12], [13].

In these proposals, the main idea is to generate transformation rules or models converted from transformation examples. Atlas Transformation Language (ATL) rules are derived from transformation examples written in concrete syntax in [14]. In [15] they use an algorithm based on particle swarm optimization to generate a consistent transformation of a model. The transformation of a source model element is encoded as a particle which has to be placed in the space of possible transformations. Another MTBE approach is given in [16]: graph transformation rules are semi-automatically derived from mappings given by a user between two models. Analysis is then based on model element neighboring, inductive logics and interaction with an expert. In [17], information about the source and target metamodels, as well as about the transformation mappings in the examples are encoded in tables processed by the approach of Analysis Formal Concept (AFC). Transformation rules are then inferred based on the commonalities in the description of mapping examples. MTBE is a complementary approach which requires an extension for the architecture because the part of traces and examples has to be added into the transformation process. When MTBE becomes mature, a perspective would be to combine MTBE and metamodel matching based approaches to get hybrid architecture.

Numerous studies are interested by the process of model transformation and let’s talk now about those which are involved in the process of semi-automatic transformation in the context MDE rather than the context MDA which restricts the concept of models for technologies OMG’s perimeters, and focused on separation of concerns: mapping and transformation processes.

In [18], [19] the authors have proposed an approach separating mapping specification and transformation definition, and an initial foundation for performing metamodel matching is discussed.

In [9], they propose an extended architecture for the transformation process in MDA, allowing a semi-automatic generation of transformation rules and the semi-automatic generation of a target model from a source model. The first two main operations of their approach are: matching and transformation.

In the two later works, the authors have focused their efforts on the separation and distinction between mapping and transformation, or between matching and transformation, without showing that really they must distinguish between matching, mapping and transformation in order to avoid ambiguity and conflict in the understanding these three operations and improve that they are highly dependent and closely related. We noticed that most of these proposals have been tested on homogeneous metamodels (Ecore, UML, Minjava, Kermeta ...).

In this work we present a separation between the three operations above indicated (matching, mapping and transformation), we taking particular account of the various factors related to the pre-matching efforts, matching process and post-matching efforts that we identified in our previous work [20], when we have proposed a new approach for the metamodels matching evaluation, benchmarking and generation of transformation rules. In [21] we have only limited on metamodel evaluation and benchmarking. All that is related to the transformation of metamodels will be addressed in this paper in order to overcome the shortcomings and limitations mentioned above with our new architecture.

III. EXTENDED ARCHITECTURE FOR SEMI-AUTOMATIC TRANSFORMATION APPROACH

Fig.1 illustrates our proposal of architecture to the process of metamodel transformation called Generic Semi-Automatic Transformation Process in MDA (GSATP-MDA), which extended the one presented in [9] and allows semi-automatic generation of transformation rules between a source metamodel and a target metamodel.
Our way of looking at the architecture is through its design time and run time layers. The following process takes place when using the architecture for model transformation.

A. Design Time

It is within this layer that is our main contribution is to go deeper into the separation of concerns. The platform-specific source and target models are abstracted into platform-independent source and target models through a given transformation (PSM2PIM) specific to the concrete technologies used at the platform-specific level. By using our architecture, the mappings between the source and target metamodels are found and specified after executing the matching tool. Executable mappings are generated from the mappings specified in the previous step, and will be used during the run time layers. The first three main operations of our architecture are: matching, mapping and transformation. All components related to these operations, and their relationships are presented by the new architecture based on standard modeling OMG where the level M3 of the MOF approach is self-defined and allows defining metamodels at level M2.

At level M2, we find different metamodels required for the semi-automatic transformation process: The source and target metamodels involved in the transformation process (source MM and target MM), the matching metamodel (Matching MM), the metamodel mapping (mapping MM) and the metamodel transformation (Transformation MM).

At level M1, we find different models involved in the semi-automatic transformation process: the source model and the target model, the matching model (Matching M), the mapping model (mapping M) and transformation model (Transformation M).

Below we present the different operations matching, mapping and transformation and the links between them:

1. Factors and parameters that must be taken into account during the pre-matching phase such as semantics resources, Initial Mapping, Initial Parameters, Auxiliary resources and additional information of metamodel and they are denoted by (1) in Fig.1. The most important factor in this step is the semantics resources, since they remove the ambiguity by exploiting semantic resources such as ontologies, dictionaries.

2. The matching operation (c.f., Fig.1 (2)) is the process that aims to produce possible mappings between two metamodels; generally, this task involves finding equivalent or similar elements between the elements of two metamodels. Both metamodels designed by source and target (representing respectively a PIM and PSM) will be taken as input by the matching algorithm. The result produced a mapping model. Matching model conforms to a metamodel matching that implements techniques that consist in finding modeling concepts semantically and syntactically similar between elements of metamodels. Similarities and correspondence between the elements of metamodels are stored in a mapping model which is also conforms to a metamodel mapping. This metamodel defines the different types of links (relationships) that could be generated by the matching model. Each link type corresponds to pattern transformation specified in the transformation model.

3. Since no generic solution for model transformation exists, it is necessary to provide the human expert can verify the mappings obtained, and, if necessary, to update, modify or improve the first version of the mapping (c.f., Fig.1 step (3)) obtained. This is one of the key steps of the transformation process proposed by our new architecture. The intervention of expert users to complete and validate the results is required. Corrections and validation are performed either by using the language OCL (Object Constraint Language) which allows to add constraints, add matches that have not been determined by the matching technique or eliminating matches falsely identified. He can also interact by using models of mappings validated manually by experts. Finally, a transformation model, conforms to its metamodel transformation, is derived automatically from the mapping
model. A transformation model is essentially represented by a set of rules that specify how the elements of the source metamodel are transformed into target metamodel. These rules are expressed in a transformation language such as Atlas Transformation Language (ATL standards-based on MOF). Usually the transformation model is enriched by information such as that relating to the execution environment, and produces a transformation program ready to run. This last part is often performed by a software engineer who implements a business model in a specific platform.

#### B. Run Time

In this layer the level M0 present the implementation phase; in fact, the transformation engine takes as input a source model and a transformation program and produces a target model in output. The platform-specific source instance is abstracted into a source model through a given transformation (PSM2PIM) specific to the concrete technologies used at the platform-specific level. The executable mapping rules are executed for the source model and a target model corresponding to the source model is generated. The target model is serialized into a platform-specific instance target through a given transformation (PIM2PSM) specific to the concrete technologies used at the platform-specific level. Finally, a transformation engine takes the source model as input, performs the transformation rules, and generates the target model as output.

### IV. ILLUSTRATION OF GSATP-MDA: BEPL2JADE

To illustrate our transformation approach on heterogeneous metamodels we use two distinct areas: Web services and Multi-Agents. Our main motivation is the complementarities between these areas; in the sense that the Multi-Agents Systems (SMA) provide the coordination framework and web services provide the infrastructure. Hence the birth of a new paradigm called SOMAS for Service Oriented Multi Agent System. To explore these complementarities, the main idea in this work is to make the transformation of the BPEL orchestration language to a multi-agent system deployed on JADE platform using the model driven architecture (MDA). The objective of this transformation is to overcome the orchestration problems such as centralization and dynamicity that can be resolved by a SMA system which is naturally decentralized and dynamic.

#### A. Metamodel BPEL

BPEL is a Web service orchestration language that is layered on top of WSDL. It is worth noting that BPEL is a considerably large language, whose specification document is several hundreds of pages long. Hence, providing a complete depiction of all the BPEL features is out of the scope here. In BPEL, the composition result is termed a process, involved services are labeled partners, and message exchange or intermediate result transformation is called an activity. A process is a set of activities. The basic concepts of BPEL are presented in [22].

![Fig. 2 BPEL language metamodel](image-url)
As presented in [23], existing standard BPEL orchestration engines has revealed the subsequent: they assume that the execution of composite Web service is controlled by a single central scheduler. This approach has several shortcomings since it engenders a large number of messages between involved component service providers and central scheduler. As well, the central scheduler represents a bottleneck and a central point of failure. On the other hand, BPEL supports predefined and static composition. That is, component services are selected in advance which is not flexible since any changes to the involved component services require changes to the overall composition. As well, new web services are offered and others disappear quite often. In addition, the organizations involved in a web service composition may change their business rules, partners, and collaboration conditions. Besides, software, machine or communication link failures may render certain sub-process of composite services unavailable, precluding thus the successful execution of the business process. Therefore, services-based systems are inherently vulnerable to exceptions. This motivates our researchers to make a BPEL process more intelligent by transforming it to a Multi-Agents System. Thus, a BPEL process can self-adapt to the web service characteristics.

B. Metamodel JADE

Each agent in JADE has a set of active behaviours. A Behavior is a java object represents a task that an agent can carry out. A Behavior is composed of three Behaviors; each one of them is itself composed of a set of sub-Behaviors:
- **Simple Behaviours**: is composed of three sub-behaviors (One-shot Behavior, Cyclic Behavior, GenericBehavior).
- **Composite Behaviours**: allows an agent to perform a composed task. A composed behavior can be: SequentialBehavior, FSMBehavior or ParallelBehaviors.
- **Planified Behaviours**: allows to an agent to perform a planified task. There tow categories of PlanifiedBehavior: WakerBehavior and TickerBehavior.

Fig. 3 presents JADE metamodel.
These two metamodels are highly heterogeneous. As mentioned in the section 3, you need an external semantic resource. In this work, this resource can be a dictionary composed of a set of pair \((X, Y)\); where \(X\) a concept in BPEL and \(Y\) its equivalent concept in JADE. Table 1 we present the identified pair \((X, Y)\).

**Table 1.** Semantic resource as a dictionary

<table>
<thead>
<tr>
<th>BPEL</th>
<th>JADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Agent</td>
</tr>
<tr>
<td>Sequence</td>
<td>SequentialBehavior</td>
</tr>
<tr>
<td>Flow</td>
<td>ParallelBehavior</td>
</tr>
<tr>
<td>Invoke</td>
<td>Send</td>
</tr>
<tr>
<td>Pick</td>
<td>BlockingReceive</td>
</tr>
<tr>
<td>while</td>
<td>CyclicBehavior</td>
</tr>
<tr>
<td>partnerLink</td>
<td>AgentStat</td>
</tr>
</tbody>
</table>

In Table 1 we have indentified some correspondences between concepts in metamodel BPEL and their equivalent in metamodel JADE. These manual similarity will be used for writing the transformations rules between the two metamodels and they will be implemented by our tool BPEL2JADE (c.f., Fig. 4), since we suggest that these correspondences are difficult to indentify by an automatic process.

### C. BEPL2JADE

In order to implement our transformation approach we are currently concentrated to develop a tool called BEPL2JADE. It is presented in the Fig. 4.

![Fig.4. Architecture of the tool BEPL2JADE](image)

This architecture consists of three basic modules:

- **Transformation:** This module use our semi-automatic transformation approach presented in section 3, in order to generate transformation rules between the two metamodels BPEL and JADE. This is done during the design-time.

- **Instantiating:** In order to generate a JADE program corresponding to a BPEL business process during run time phase, we instantiate transformation rules generated by the transformation module.

- **Deployment:** The result of the instantiating module is an XML file is generated. To deploy this program JADE, a step of parsing is required.

### V. CONCLUSIONS

In this paper, we have presented our approach for a semi-automatic transformation process in MDA with the new architecture. We argue that a semi-automatic transformation process will be a great challenge for the approach MDA as there is not yet a complete solution that automates the development of model transformation. This process is semi-automatic because the metamodel matching algorithm finds corresponding elements that are then validated, rejected or corrected by an expert user. The key principle for our approach is to consider matching, mapping and transformation of metamodels as first class entities in MDA.

As an immediate extension for this work, we concentrate our efforts firstly on testing with two heterogeneous metamodels (BPEL and JADE) to validate and implement our tool BPEL2JADE for semi-automatic transformation. Secondly we validate our architecture GSATP-MDA by testing it with several scenarios and several types of metamodels. The semi-automatically generation of transformation rules is related to the mappings obtained through the metamodel matching process.

As a future work, we intend to reach BPEL2JADE implementation and test it on real cases. In order to validate our proposal and get a better efficiency of BPEL2JADE, we would to use ontology as a semantic resource.

### REFERENCES


