Applying Model Driven Architecture Approach to Develop Software Systems – Case Study

Mohammed Abdalla Osman Mukhtar
College of Computer Science and Information Technology
Alzaeim Alazhari University, Khartoum, Sudan
Abofatima92@gmail.com

Abstract: Model-Driven Architecture (MDA) has been initiated by Object Management Group (OMG) as a new approach to design and develop a complex system. Some developers argue that this approach is still relatively unripe and still need more supporting tools to become mature enough. In this paper, we try to convince those opponents by applying MDA approach in specific case study to proof that MDA has become most preferable technique to develop or reengineer a wide range of software products. The idea behind MDA is both simple and grand, because it is trying to decouple the definition of application systems functionality from technology platform that run on. The objective from this decoupling is to ensure that the functionality of the system can be preserved even there is changing affect the underlying technology platform. We also highlight the most important steps taken to generate the code from high level and low level system abstraction using formal models, excluding the last step which is delegated to generate the code from the last generated model. Case study provided to support the possibility of developing a complete software system using only OMG asset tools. Experiences learned from this case study can be applied to all similar cases.

Keywords: MDA, PIM, PSM, QVT, Mapping Specification, Model Transformation.

1. Introduction

Model-Driven Architecture is a major initiative by the OMG, and therefore most of the computing industry, to re-use program code. MDA has been the driving rationale behind a large number of the OMG standards efforts since 2001 [1][2][3]. MDA is trying to support ever-changing business environments by using effective tools to design complex systems, and keeping the software development cost (coding effort, delivering-to-customer, etc.) at an reasonable margin [4].

MDA uses a number of powerful standard tools, like UML [5], MOF [6], OCL [7], XMI [8], BPMN [9] and QVT [10]. Mainly MDA is initiated to shift the focus of software development process from coding to modeling to achieve the great goal, which is separating system functionality from specific implementation platform.

Using MDA-based development approach means the developers will use models as the main product of the process of software development, either for software development and maintenance or software requirements documentation. Referring to Figure 1, models are used to represent different levels of abstraction to describe system architecture. These levels can be arranged into four major steps: step one is Computational Independent Model (CIM), followed by second step Platform Independent Model (PIM), we can have more than one Platform Specific Model (PSM) due to platform needed to use in this third step, and the last step is Implementation Code Model.

According to these stages, there is two kind of system abstraction, first one contains the high level abstraction models (CIM, PIM) which represent the main system characteristics and the system functionality as proposed and expected by system designers. The second kind of system abstraction contains the low level abstraction models (PSM and Implementation Code Model) which construct the implementation details according to specific platform. Mapping specification from High level abstraction models to low level abstraction models is important. There are transformation tools automate this mapping specification by using special transformation techniques [11][12][13][14].

This paper is organized as follows. Section 2 is representing the ultimate goal of MDA as a reliable method to support creation and development of large software systems providing a new methodology where
systems can be evolved, and technologies can integrate and synchronize. Section 3 discusses the concept of model transformation because it is representing the main contribution of MDA in software development field. In section 4 there is a credible case study to support what is the scenario when we try to develop a complete software system. In section 5 there is general description for the methodology that can be used to realize the provided case study. Section 6 discusses some results should we execute the provided case study. The last section is the conclusion of our research.

2. The Ultimate Goal of MDA

The ultimate goal of MDA is to be able to develop applications in established domains without developing any new program code. MDA approach is introduced as an important and reliable method to support creation and development of large software systems providing a new methodology where systems can be evolved, and technologies can be integrated and synchronized [15]. As we can see in Figure 2, Model driven architecture approach is depending on architecture constructed four meta-layers each of which is an instance of the layer above. This means the objects for example of layer zero (called M0) is an instance specified by model which is in layer one (called M1). On other hand, we can say M1 holds the model of M0 objects. Similarly, M2 is a model describes the instance which is in M1 that often called a metamodel because it is a model of model. In fact, M2 corresponds to UML [16] or any other modeling language can be derived from M3 metamodel. Finally, layer M3 holds a model of the information about M2. M3 is corresponds to Meta Object Facility (MOF). It is a metametamodel language because it represents the conceptual model of the modeling language [17].

![Figure 2. 4 Layers architecture of MDA](image)

The term Platform Independent Model (PIM) is used to identify a model that structurally describes functionality of a system under process without any details about implementation aspect. While Platform Specific Model (PSM) is used to refer to models that have information about implementation details [18].

MDA should be taken as an approach to integrate and harmonize between two different technologies. MDA methodology has been built on specifying the mapping between the items that constructing the source model (PIM) to the corresponding items which are constructing the target model (PSM).

The mappings from source model to target model can be complex due to a given item from PIM may be mapped to many items in PSM, for instance a single element in Business Process Modeling Notation (BPMN) as PIM can be mapped to a DDL(SQL), an Entity Java Bean (EJB), and Remote Interface [10].

The main innovation in MDA is to attempt to remove implementation detail altogether from the PIM specification, and to package the PSM code base into coherent subsystems whose constructs are on the same sort of scale as the PIM constructs. This is why the relationship between the PIM and PSM is considered to be a mapping rather than a compilation or interpretation. A secondary but significant innovation is that all the models are MOF model instances, thereby leveraging the large and expanding base of MOF-based toolsets. The collection of MOF-based models, PSM- interfaced code, and toolsets is a significant increase in the scale of capital equipment available to software developers [19].

3. Model Transformation

With the widely used of model transformations there is a rising need for formal methods concentrating on a systematic model transformations development [20]. There are two types of model transformation: Vertical Transformation and Horizontal Transformation. Vertical model transformations are affecting system abstraction level from specification prospective, while the Horizontal transformations do not. Vertical model transformations are used during forward or reverse system engineering to abstract or refine a model. Horizontal model transformations are used to enhance or optimize or reconstruct the internal structure and/or quality of software model [21]. According to figure 2, the vertical transformation can be done from one layer to other direct lower layer (i.e from M3 to M2, or from M2 to M1), while the horizontal transformation can be done within the same modeling layer.

Some of MDA critics argue that creating and developing of more than one model (one for PIM and another one for PSM) affects MDA efficiency, and this serious weakness can be considered as a factor to overbalance the good advantages of MDA approach. MDA developers defend this accusation by saying after a PIM has been created, the process of PSM instantiation can completely be done automatically using model transformation process [22].

PIM and PSM models describe one system and these two models have much in common. Therefore, it is significant to propose a transformations definition that simplifies the process of converting a PIM model that used to describe system functionality into one or more PSM model that used to describe the
implementation of this functionality for the same system. It is noteworthy that each of these two models can be created using different modeling languages, off course with different set of notions that may be built on different platform technologies. Model transformation definition is an algorithm which is automatically and executed using some formal language[23]. There are many languages to automate and execute this transformation like KerMeta[24], MATA[25], or QVT[26].

A. Model Transformation Scenario

To complete the picture about using MDA approach to create software system, we need to know the steps that should be taken. QVT is the main model transformation language, which is used by most of OMG assets and many MOF-based modeling languages. In our case study, QVT should be taken to generate PSM instances that later should be passed to the tool of code generation to generate final code. Figure 3 represents models needed as input to execute the QVT code, QVT code receives three inputs: PIM model, PIM model instance, and PSM model. The output from executing of QVT code according to these three inputs is the model instance of PSM model.

![Figure 3](image.png)

Figure 3. Using QVT Engine to Generate PSM Instances

4. GraphEditor(GE) Software System – Case Study

We want to build an editor which will allow us to specify State Transition Machines (STM), using the graphical notation of Figure 4[27]. We know that this case study is a simple one, but it is suitable to cope all what we need to do during the system creation lifecycle using MDA approach, and we believe that most of complex systems structurally similar to this case study. This GE will be able to design an application in which patients buy treatment card once obtaining a location on reservation list. When they are going to the pharmacy, they will buy medication as it has been described in the prescription. Due to more examinations, patients may be rescheduled on reservation waiting list.

![Figure 4](image.png)

Figure 4. Simple State Machine to Describe Patient Treatment Steps

A. First Step

The first step is to create the metamodel of STM as PIM (Figure 5), to take as first input passed to QVT Engine using XML Metadata Interchange (XMI) file format. XMI is an important tool initiated by OMG to facilitate model interchanging property between different modeling tools[28].

![Figure 5](image.png)

Figure 5. STM Metamodel (PIM)

B. Second Step

Now as the second step we need to create the PIM model instance also to pass as second input for the QVT Engine. To do that most of modeling languages like MOF and UML has Instances Model which is used to provide a standard way to talk about instances[19]. Figure 6 represents an example of some instances of instances model for STM case study, while Table 1 contains a complete description to all instances in this case study.

![Figure 6](image.png)

Figure 6. Some Instances of Instances Model for STM Metamodel (PIM)
Table 1. All instances of Instances Model for STM Metamodel (PIM)

<table>
<thead>
<tr>
<th>Obj.</th>
<th>State</th>
<th>Transition</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WishTreat</td>
<td>WishTreat</td>
<td>Reservation</td>
</tr>
<tr>
<td>2</td>
<td>Healthy</td>
<td>Healthy</td>
<td>TakeMedic</td>
</tr>
<tr>
<td>3</td>
<td>ReserveList</td>
<td>ReserveList</td>
<td>TakePharmacy</td>
</tr>
<tr>
<td>4</td>
<td>PinMedic</td>
<td>PinMedic</td>
<td>None</td>
</tr>
<tr>
<td>5</td>
<td>BuyMedic</td>
<td>BuyMedic</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>Healthy</td>
<td>Healthy</td>
<td>FeelSick</td>
</tr>
<tr>
<td>7</td>
<td>ReserveList</td>
<td>ReserveList</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>MoreDiagnos</td>
<td>MoreDiagnos</td>
<td>None</td>
</tr>
<tr>
<td>9</td>
<td>ReqDiagnos</td>
<td>ReqDiagnos</td>
<td>None</td>
</tr>
<tr>
<td>10</td>
<td>PreMedic</td>
<td>PreMedic</td>
<td>None</td>
</tr>
<tr>
<td>11</td>
<td>ToPharmacy</td>
<td>ToPharmacy</td>
<td>None</td>
</tr>
<tr>
<td>12</td>
<td>TakeMedic</td>
<td>TakeMedic</td>
<td>None</td>
</tr>
<tr>
<td>13</td>
<td>FeelSick</td>
<td>FeelSick</td>
<td>None</td>
</tr>
</tbody>
</table>

C. Third Step

Third step is to develop GE metamodel as PSM (the platform in this case can be any graph editor or other MOF-based modeling language) to use as a third input passing to the QVT Engine through QVT code. As we mentioned earlier the output from QVT Code is the instances of PSM (GE) metamodel. Figure 7 shows the GE metamodel.

D. Fourth Step

After preparing all input requirements to execute the QVT code, the fourth step is to specify independently the mapping between PIM metamodel and PSM metamodel:

Table 2. PIM to PSM Mapping Specifications

<table>
<thead>
<tr>
<th>STM (PIM)</th>
<th>GE (PSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Node</td>
</tr>
<tr>
<td>Transition</td>
<td>ArcEnd, which will be specified by EndNotation attribute.</td>
</tr>
<tr>
<td>Event</td>
<td>Arc</td>
</tr>
</tbody>
</table>

E. Fifth Step

The last step in this case study is to write QVT code to generate instances of GE or PSM. Figure 8 contains the QVT-R code to implement and execute the mapping specification of Table. 2. This code used to generate model instance for the PSM of Graph Editor (GE) Software.

```
transformation STMGE(STM:PIM_STM, GE:GE_PSM){
  key State {name};
  key Transition {source, target, triggeredBy};
  key Event {name};
  key Node {nodeText};
  key ArcEnd{arcText};
  key Arc {source, target};
  top relation STMToGraph {
    n : String;
    check only domain STM s : PIM_STM::STM{
     stm = st : PIM_STM::STM {name = n},
      name = ns};
    enforce domain GE g : GE_PSM::Graph{
     name = n};
    when {
      STMToGraph (st, gr);}
  }
  top relation StateToNode{
    n, ns : String;
    check only domain STM s : PIM_STM::State{
     stm = st : PIM_STM::STM {name = n},
      name = ns};
    enforce domain GE nd : GE_PSM::Node{
     graph = gr : GE_PSM::Graph {name = n},
      shape = 'circle', nodeText = ns};
    when {
      STMToGraph (st, gr);}
  }
  top relation TransitionToArc{
    ng : String;
    check only domain STM s : PIM_STM::Transition{
     stm = st : PIM_STM::STM { },
      source = so : PIM_STM::State { },
      target = ta : PIM_STM::State{}};
    enforce domain GE a : GE_PSM::Arc{
     graph = gr : GE_PSM::Graph { },
      line = 'thin',
      source = sr : GE_PSM::ArcEnd{
        arrowHead = 'none',
        node = no : GE_PSM::Node { }
      }
    };
    target = tr : GE_PSM::ArcEnd{
      arrowHead = 'simple',
      node = nta : GE_PSM::Node{ }
    }
    arc = a : GE_PSM::Arc{ }
    when {
      STMToGraph (st, gr);
      StateToNode(so, no);
      StateToNode(ta, nta);}
  }
  top relation EventToArc{
    n : String;
    check only domain STM tr : PIM_STM::Transition{
     triggeredBy = e : PIM_STM::Event{name=n};
      enforce domain GE a:
        GE_PSM::Arc{arcText=n};
    when {
      TransitionToArc(tr, a);}
  }
}
```

Figure 8. QVT-R Code [29]
5. General Methodology

We take metamodels for PIM (source model) and PSM (target model) prepared with a CASE tool of UML such as MagicDraw[30]. Using the previous tool, we export each metamodel to EMF [31] XMI [32] format, which is used later to create the model instance for the PIM. Then we need to pass these three inputs (PIM XMI file, PSM XMI file and PIM model instance) to the QVT-R code to generate the PSM model instance in XMI format. Later we will process the generated PSM XMI file with an XSLT [33] processor to generate the appropriate code.

6. Result and Discussion

The result from the code above will be a number of GE instances and many features that have been applied to generate these instances. The bottom part of Figure 9 shows the report after executing an example of QVT code (there are 23 generated instances and 63 features applied).

Figure 9. An example of QVT code and its output using MediniQVT [34]

The output after executing the QVT code should be in an XMI file format. We can use any tool (like MagicDraw) to graphically display this kind of file format.

We can’t say that our code is the perfect, but any code may be error prone, or contain logical error that will not affect the execution, while it is definitely affecting the output.

Later we should pass this output file to generate the final code. As the final product, generated code may need minor manual modification also.

7. Conclusion

In this paper we provided a complete case study to highlight the important steps required if we try to create a system using MDA approach. We can conclude that the idea behind MDA is both simple and grand [10], because it is trying to decouple the definition of application systems functionality (in PIM) from technology platform that run on (in PSM). The objective from this decoupling is to ensure that the functionality of the system can be preserved even there is changing affect the underlying technology platform.

References

oriented programming, systems, languages, and applications - OOPSLA ’03, 2003, p. 124.


Mohammed Abdalla Osman Mukhtar was born in Khartoum Sudan 1975; he graduated from Sudan University of Science & Technology in Sudan, Faculty of Computer Sciences & Information Technology in 1998. He got his Master degree in Computer Sciences from University of Khartoum in Sudan 2002. He is Currently Ph.D. student in Universiti Teknologi PETRONAS (UTP), Computer & Information Sciences Department. His domain of work is: Development of Web Applications Using Model Driven Architecture Approach.