AN APPROACH OF MODEL DRIVEN DEVELOPMENT

Poonam Mishra ¹, Reeta Kumari ²
¹Department of Computer Application
²Department of Information technology, Mrs. K.M.P.M. Vocational college, Jamshedpur, India

ABSTRACT:

The main aspect of model-driven-development is to move the focus from programming to modeling. In this paper we explain MDD approach in terms of the major concepts, the premises, benefits and characteristics which derive its adoption. It is a way of software development where primary software artifacts are models from which code and other artifacts are generated. In this approach the developer designs software to increase productivity and quality of the system under development.

Keywords: Unified modeling language, DSLs, Meta model.

[1] INTRODUCTION

Model-Driven Development (MDD) is software development process. It is an integrated approach to architecting, developing, testing and deploying complex, high performance systems that specifies programs in domain specific languages (DSLs). In this approach, software systems are documented and produced as models. MDD uses models to represent a program. A model is written in a DSL that specifies particular details of a program’s design. As an individual model captures limited information, a program is often specified by several different models. A model can be derived from other models by transformations, and program synthesis is the process of transforming high-level models into executables (which are also models) [1].

In a traditional software development process we generally start with gathering requirements for the system that we want to develop. According to the list of requirements we specify a number of scenarios and use cases to get a clear concept on the behavior of the system. Then we can create an initial conceptual model containing the classes of the system and their interrelations. The next step is to design a more detailed model, which acts as a blueprint for the system under development; this model contains all attributes and methods of classes in the system. At last, the blueprint model is used as a guideline to write the source code for the system.

In model-driven development (MDD) these steps are performed automatically. The UML(Unified modeling language) is used to create models for MDD. These models have well defined semantics and can be transformed into implementation of source code.
So in future, I will design UML for any abstract data type like linked list, stack and queue etc. Then it will proceed for source code.

[2] BASIC CONCEPTS

MDD provides three main concepts. These are the model, the meta-model and the model transformation. All are required for a proper MDD approach. We explain each of these concepts and how they are related to each other.

Models

A system and a relation is known as the RepresentationOf relation between the system and a model. The system is represented by a model, but it does not have to be a full description. A model is created with a particular goal in mind and therefore only information necessary to achieve this goal is included in the model [2]: all other information about the system is considered irrelevant. Models of software systems are often specified in the Unified Modeling Language (UML) [3]. A UML class diagram is an example of a model of a software system, as shown in [Figure-1].

![Figure: 1. Relationship between model and system.](image)

Classification of models

Models can be classified using different approaches:-

First classification approach is to make a distinction between specification models and description models. The first type of models is created before implementation of the system and is used to specify how the system under development should look like when it is implemented. The implemented system is only considered valid if it completely matches its specifications defined in the model. In this case the model is validated instead of the system under study. The model is valid if all its statements are true for the described system.

Another classification approach is to distinguish between sketchy models, blueprint models and executable models. In this approach sketchy and blueprint models can be used for both forward and reverse engineering. Sketchy models for forward engineering are used to communicate ideas with other developers; they present a rough sketch of some part of the system under development. Sketches therefore do not contain all the details of the system. In forward engineering the blueprint model specifies all details needed for a programmer to implement the system under development. Blueprint models for reverse engineering completely describe (a part of) the existing system. The main difference between sketchy models and blueprint models is that the
former are aimed to be explorative, while the latter are definitive. Executable models are directly compiled to executable code.

Meta-models

A specific kind of model is the meta-model. A definition of the concept meta-model, taken from [4] is given below.

A meta-model is a specification model for a class of systems where each system in the class is itself a valid model expressed in a certain modeling language.

A metamodel is a model of a language that captures its essential properties and features. These include the language concepts it supports, its textual and/or graphical syntax and its semantics (what the models and programs write).

It is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems. A meta-model typically defines the languages and processes from which to form a model[11].

If we explain this definition, we can say that a meta-model is a model of a set of models. In this set all the models are specified in a certain modeling language. i.e. , the UML meta-model is a model for all models that are specified in the UML.

The two basic metamodeling relationships is shown in [Figure-2].

![Figure: 2. Metamodeling relationship.](image)

A model represents a system and conforms to a metamodel

Model Transformations

Models can be transformed into other models using model transformation. Model transformations are the key challenge in MDD, because it generate source code from higher-level models. At higher levels of abstraction, these transformations between models can become very complex.

In the introduction of this chapter we described the traditional process of software development. If we would use MDD for this process, we would at least need four types of models: - a list of requirements, a conceptual model, a blueprint model, and the source code and three transformations. The process starts with a model at a very high level of abstraction i.e. the requirements and transforms these into models of lower level of abstraction i.e. up to source code.
Model transformations take at least one model as input to produce at least one output model, but it is also possible to have multiple inputs or output models [6]. Meta-models of the input and output models are needed to specify transformations between two models. The transformation between the source model and target model is defined in a number of transformation rules that make use of the meta-models of both the source and target model. These rules are specified in a transformation language and conform to a certain meta-model. The transformation rules are also a model, not of the software system, but of the transformation between a source and target model.

There are three different architectural approaches for defining transformations:

The direct model manipulation approach, the intermediate representation approach, or the transformation language support approach.

Direct Model Manipulation – With the help of this tool users access to an internal model representation and the ability to manipulate the representation using a set of procedural APIs.

Intermediate Representation – the tool can export the model in a standard form, typically XML. An external tool can then take the exported model and transform it.

Transformation Language Support – the tool offers a language that provides a set of constructs or mechanisms for explicitly expressing, composing and applying transformations.

[3] TYPES OF UNIFIED MODELING LANGUAGE

UML gives numerous options to developers for specifying software systems. A UML model can graphically depict the structure and/or behaviour of the system under discussion from a certain viewpoint and at a certain level of abstraction. It is used to generate code in various languages using UML diagrams.

There are two broad categories of diagrams and then are again divided into sub-categories:

1) Structural Diagrams
   2) Behavioral Diagrams

1) Structural diagrams represent the structures which are used extensively in documenting the software architecture of software system.

The four structural diagrams are:

   a) Class diagram
   b) Object diagram
   c) Component diagram
   d) Deployment diagram

a) Class Diagram models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance,
associations and others. Class diagrams are the most common diagrams used in UML. Class diagrams basically represent the object-oriented view of a system which is static in nature.

Class diagram represents the object orientation of a system. So it is generally used for development purpose. This is the most widely used diagram at the time of system construction.

b) Object diagrams can be described as an instance of class diagram. So these diagrams are more close to real life scenarios where we implement a system

c) Component Diagram: Component diagrams represent a set of components and their relationships. These components consist of classes, interfaces or collaborations. So Component diagrams represent the implementation view of a system. During design phase software artifacts (classes, interfaces etc) of a system are arranged in different groups depending upon their relationship. Now these groups are known as components. Finally, component diagrams are used to visualize the implementation.

d) Deployment Diagram: Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed. Deployment diagrams are used for visualizing deployment view of a system.

2) Behavioral diagrams represent the behavior of the system which are used extensively to describe the functionality of software system. UML has the following five types of behavioral diagrams:

   a) Use case diagram
   b) Sequence diagram
   c) Collaboration diagram
   d) State chart diagram
   e) Activity diagram

   a) Use case diagrams are a set of use cases, actors and their relationships. They represent the use case view of a system. A use case represents a particular functionality of a system.

   So use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.

   b) Sequence Diagram: A sequence diagram is an interaction diagram. From the name it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another. Interaction among the components of a system is very important from implementation and execution perspective. So Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

   c) Collaboration Diagram: Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.
The purpose of collaboration diagram is similar to sequence diagram. But the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.

d) State chart Diagram: Any real time system is expected to be reacted by some kind of internal/external events. These events are responsible for state change of the system. State chart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface etc. State chart diagram is used to visualize the reaction of a system by internal/external factors.

e) Activity Diagram: Activity diagram describes the flow of control in a system. So it consists of activities and links. The flow can be sequential, concurrent or branched. Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system. Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

[4] BENEFITS OF MODEL DRIVEN DEVELOPMENT

The advantages of an MDD approach are as follows:

1. Increased productivity: MDD reduces the cost of software development by generating code and artifacts from models, which increases developer productivity.
2. Maintainability: In MDD high-level models are kept free of irrelevant implementation detail. These implementation details make it easier to handle changes in its technical architecture. A change in the technical architecture of the implementation is made by updating a transformation. The transformation is reapplied to the original models to produce implementation an artifact following the new approach. This flexibility also means that it is possible to try out different ideas before making a final decision.
3. Decrease of development time: Since most of the code is generated instead of implemented manually, which decreases development time.
4. Reuse of legacy: You can consistently model existing legacy platforms in UML. If there are many components implemented on the same legacy platform, you can develop reverse transformations from the components to UML. Then you have the option of migrating the components to a new platform.
5. Adaptability: Adaptability is a key requirement IT systems. When adding or modifying new function, we only develop the behavior specific to that capability. The remaining information needed to generate implementation artifacts was captured in transformations.
6. Raise of abstraction: When the developers design the application, the abstraction hides implementation details from developers, which leads to a reduction of complexity of the software artifacts that developers use to design the application. Since current platforms and frameworks are getting more and more complex, this reduction is most welcome [8].
7. Consistency: Manually applying coding and architectural decisions is an error prone activity. MDD ensures that artifacts are generated consistently.
[5] CHARACTERISTICS OF MODEL

The following five key characteristics of an effective model are:

Abstraction: Abstraction is the most important attribute of a model. Through abstraction, a model hides detail that is irrelevant for a given viewpoint. Software systems get more and more complex; if models hide the complexity of the underlying system, the developers get a better overview on the system’s functionality, which gives higher quality and productivity.

Understandability: Just hiding non-relevant information is not enough; a model should present its contents in a clear and understandable way. Models can be presented in both textual and graphical form. Usually, system requirements are presented in structured text documents, while an object model of a software system is often in the form of a graph.

Predictiveness: The model should present the system in such a way that it is possible to correctly predict the system’s non-obvious characteristics.

Accuracy: The important features of the modelled system should be represented accurately in the model.

Inexpensiveness: A model should be significantly cheaper to construct than the actual system.

[6] EVOLUTION OF MODEL DRIVEN DEVELOPMENT

Model-Driven Development (MDD) is a software development process. Software systems need to evolve, and systems built using model-driven approaches are no exception. The development and maintenance effort can be reduced by working at the model instead of the code level. Models define what is variable in a system, and code generators produce the functionality that is common in the application domain.

Software systems are continually required to increasing demands of correctness. To achieve these requirements, software development has evolved into a process of reusing existing software rather than constructing a new software system completely.

It required various types of evolution:

In regular evolution, the modeling language is used to make the changes. In meta-model evolution, changes are required to the modeling notation. In platform evolution, the code generators and application framework change to reflect new requirements on the target platform. Finally, in abstraction evolution, new modeling languages are added to the set of (modeling) languages to reflect increased understanding of a technical or business domain.

[7] CONCLUSION

In this paper, we presented the applicability of model-driven development to increase productivity of the software development process of small and simple endevour applications.
As this type of application is mainly data-driven, it can be used as a model an initial model. It can also be concluded the features that we described will be useful for initial stage researcher.

REFERENCES