On the use of models transformation to specify the dynamic behavior of production system

A. Dekhinet¹, A. Chaoui², K. Khoualdi³

¹ Al Jouf University KSA, abddekhet@ju.edu.sa
² University of Constantine Algeria, achaoui@hotmail.com
³ King Abdulaziz University KSA, kamel_khoualdi@yahoo.com

Abstract— The increasing complexity and the diversity of domains lead to view the system designing or analyzing at high level of abstraction. The last decades have seen the emergence of the MDE (Model-Driven Engineering) approach. The main purpose of this approach is to define a framework for model handling: Metamodeling, Modelling, Transformation between models. The metamodeling defines a Domain Specific Modelling Language (DSML) which capture the domain concepts, the relationships among them, and the rules that constrain the manipulation of the model elements. The definition includes the abstract syntax, concrete syntax, static semantic, dynamic semantic and the visualization of the language. In this paper we define a metamodel (DSML) for production systems using the GME (Generic Modeling Environment) framework. The DSML customizes the GME which allows the modeller to create a production system model. Therefore, the model can be used for analysis, testing and operation. In addition, the model can be transformed to a well-defined semantic model (petri net, abstract state machine, …) to perform certain operation such as formal verification.

Keywords— Model, DSML, Production system, GME, Transformation rule

I. INTRODUCTION

In MDE we consider a system at high level of abstraction using the concept of metamodeling and handle the model of model concepts instead of the real world model itself. The model handling is mainly based on the transformation of models. Model transformation consists of transforming a source model to a target model according to a set of rules representing the metamodel of the transformation. This transformation depends on the purpose of the metamodeling or the definition of the DSML. Figure 1 depicts the principle of models transformation, where DSML1 and DSML2 can be the same (endogenous or In-place) or different (exogenous). A source model MS representing a real system and conforms to DSML1 is transformed to a target model MT conforms to DSML2 according to a transformation metamodel MTT. A variety of transformation approaches has been developed, but generally the transformation problem is formulated as graph transformation problem where a set of rules guides the transformation process [1]. A graph transformation rule has the form p: LHS —→ RHS, where the LHS (left-hand side) specify the pre-condition and the RHS (right-hand side) the post-condition. The transformation process transforms a graph G (MS) into graph H (MT)
by the sequencing of the application of rules. Each application cycle consists of three steps:

- Locate an occurrence of LHS in G
- Delete from G the part matched by LHS
- Paste of RHS to the result, giving H

The variety of the transformation approaches has led to the development of various tools. According to the domain of application and the transformation technology, we can distinguish [2]:

- General-purpose tools include AGG and PROGRESS;
- Reengineering tools include FUJABA;
- Model-To-Model transformation tools include GReAT, ATOM3 and MOLA;
- Model checking and verification tools include VIATRA, GROOVE and CheckVML.

A production system is a specific domain; however we can model its concepts and the relationship between them with an appropriate DSML. The purpose of the modelling is to specify the behaviour of a production system where the DSML1 and the DSML2 are the same. Otherwise the suitable framework must provide the support for Model-To-Model transformation. The research products based on the model-to-model transformation approach include ATOM3, DOME, Moses and GME. For several reasons we opted for GME, in addition it has an important distinguishing property that it is based on UML class diagrams. GME (Generic Modelling Environment) is a configurable metamodeling environment developed at the Institute for Software Integrated Systems at Vanderbilt University. It provides a framework for creating metamodel (DSML) environments. In conjunction with GME, the language used to specify and implement the model to model transformation is GReAT (Graph Rewriting and Transformation), which is developed using GME.

![Fig. 1 Models transformation](image-url)
II.  METAMODELING WITH GME

Based on domain's modelling paradigm, GME [3] tool allows the generation of the DSML environment used to create domain specific model. The syntactic part of the DSML is specified using the pure UML class diagram and the static semantic part is specified in the form of constraints using the Object Constraint Language (OCL). As required it's possible to extend UML to specify a certain visualisation objects of the metamodel. Once all the metamodel elements are specified, GME interprets it to generate the new domain specific modelling environment.

The metamodel is built using a palette of predefined UML stereotype provided by the framework (Fig. 2). The richness of the palette allows the creation of large-scale and complex models. The palette includes atomic stereotype such as Atom, composite stereotype such as Model, association stereotype such as Connection and other stereotypes.

Fig. 2 GME Framework

III.  MODEL-TO-MODEL TRANSFORMATION

GReAT is the language used in GME for the specification of model-to-model transformation or the dynamic semantic of the DSML [4]. Based on the type graph, GReAT supports the exogenous transformation and allows the specification of
cross-domain transformations, where models belonging to different domains can be transformed simultaneously. GReAT consists of three sub-languages: the pattern specification language, the transformation rule language and the control flow language.

1. **Pattern Specification Language**

   The pattern specification language is used to specify the graph patterns that will form the LHS and RHS of the rule. Unlike some other approaches in GReAT both the LHS and RHS is specified together as a single pattern.

2. **Transformation Rule Language**

   The transformation rule language is used to specify the transformation rules. Each object in the graph pattern plays a specific role in the transformation. There are three types of roles:

   - *Bind*: Used to match objects in the graph
   - *Delete*: After match the object is deleted
   - *New*: After match the object is created

   According to the general form of a the transformation rule, the objects marked as “Bound” can be considered as the LHS and the objects marked “New” or “Delete” are the RHS. In certain case, the pattern itself is not enough to take decision of the matching, so we need additional non-structural constraints on the pattern. A C++ code can be placed in *Guard* to restrict the execution of the rule if certain conditions hold. The rule execution implies the creation or deletion of objects and the values of objects attributes are affected by the transformation, so the *Attribute mapping* specifies the update by a set of assignment statements. The application of a rule depends on the objects (Packets) supplied by the previous rule, the *In* and *Out* ports are used to pass objects between rules.

---

![Fig. 3 Rule with patterns, guard and attribute mapping](image)
3. Control Flow Language

The model to model transformation is the result of sequencing the application of rules, thus the control flow language is used to specify the flow of the application of rules and manage the complexity of the transformation process. The language supports the following features:

- Sequencing: Rules can be connected and executed in sequence
- Non-Determinism: Rules are executed in parallel where the order of execution is non-deterministic
- Hierarchy: High-level or compound rule can contain other compound rules or primitive rules
- Recursion: High-level rule can call itself
- Conditional execution: Test/Case construct used to choose between different control flow paths.

IV. PRODUCTION SYSTEM METAMODEL

A production system can be considered as a domain specific and it's possible to define its DSML using the GME. Succinctly, a production system is composed of several workshops, in turn the workshop contains various stations each performs a specific task in the production chain. Consider a workshop that manufactures a specific product PR which includes material-handling machines and manufacturing machines. The material-handling machine can be tray or conveyor, and the manufacturing machine can be production machine or assembly machine. The final product PR is made of several parts. Manufacturing machines are connected to conveyors; assembly machine is connected to tray and conveyors are connected to trays. The product pieces moved between manufacturing machines are held by material-handling machines. The piece denotes any product parts included the product itself. Thus the metamodel or the DSML that reflects this description is shown in figure 5.

![Fig. 4 Production system metamodel](image-url)
The metamodel of the production system can be used as a paradigm to create a model for real world system. The metamodel allows configuring the GME as an environment for production system modelling. The Figure 6 presents a model created using the new GME environment. There is a two production machines of P1 and another of P2. The pieces are transported by conveyors to a tray. The assembler takes the pieces and produce PR. This one is moved by the conveyor and the tray to the final destination.

Fig. 5 Production system model

V. TRANSFORMATION RULES

The definition of the production system DSML aims to specify the dynamic behavior of the system. The definition alone is not sufficient to describe all aspects of the system, and it must be complemented by the specification of transformation rules that model the dynamic behavior of the system.

The production line of PR consists of a set of operations that represent the dynamic behavior of the system. Each operation is modelled by a transformation rule. The transformations rules (Fig. 6) and their corresponding operations are:

Rule 1 : Manufacturing P1 by ProdP1 and its deposit on conveyor
Rule 2 : Manufacturing P2 by ProdP2 and its deposit on conveyor
Rule 3,4 : Transferring the product P1 and P2 from conveyors to trays
Rule 5 : Assembling P1 and P2 by the assembly machine and produce the final product PR
Rule 6 : The deposit of PR on conveyor
Rule 7 : Transferring the product PR from conveyor to tray. This rule is similar to rule 3 and rule 4

As GReAT allows the Non-Determinism execution of rules, the execution order is as follows:

Seq 1 : Seq2 // Seq3
Seq 2 : Rule1, Rule3
Seq 3 : Rule2, Rule4
Seq 4 : Rule5, Rule6, Rule7
Rule 1

Guard

Return ProductionM.Nhr=0;

ProductionM.Nhr=ProductionM.Nhr-1;

Rule 2

Guard

Return ProductionM.Nhr=0;

ProductionM.Nhr=ProductionM.Nhr-1;

Rule 3

Guard

A. Dekhinet, A. Chaoui, K. Khoualdı
Rule 4

![Rule 4 Diagram]

Rule 5

![Rule 5 Diagram]

Rule 6

![Rule 6 Diagram]

Fig.6 Transformation rules
VI. CONCLUSION

The MDE introduces a new vision and approach for modelling production systems. A dedicated production system framework can be generated based on the ability of the model handling frameworks to generate a specific environment. This approach allows the modeller to create his own environment, which allows him to handle the concepts and constraints of his expertise domain.

In this paper a new framework was created based on the proposed DSML. The set of rules have been defined to model the dynamic behavior of production system. The definition of DSML and rules depends on the purpose of the transformation. For example, if the purpose of transformation is the performance evaluation or the QoS (Quality of Service) of production system, each rule can be divided into more detailed rules and the DSML can be enriched by adding new attributes and objects.

The GME framework and GReAT are powerful enough to model complex systems. However, the metamodeling process depends on the ability of the modeller to describe the syntax and the semantic of his system and their mapping in terms of UML stereotype and transformation rules.

VII. REFERENCES