The Model-driven Architecture in an Industrial Environment

Asadullah Shaikh*, Sheeraz Ali†, Jawed Ahmed† and Muniba Shaikh‡
*Universitat Oberta de Catalunya (Spain)
Email: ashaikh@uoc.edu
†Cursor Software Solutions (UAE)
Email: sheeraz@cursorsoft.net
†Cursor Software Solutions (Pakistan)
Email: jawed.ahmed@cursorsoft.net
‡Quaid-e-Awam University of Engineering
and Technology (Pakistan)
Email: hina04_muet@hotmail.com

***Abstract*—Model-driven Architecture (MDA) is a development methodology that helps designers to define and communicate a solution. It is becoming popular day by day due to its transformation techniques from one model to another. In an industrial environment such as software and Business, MDA can play a major role in the development of software/embedded systems. In this peace of research we are considering that how software development in the industrial environment is affected by the models and transformations. As a result of this, a literature study is being done to highlight some real time models and transformations in the industrial environment. Couple of transformations have been investigated and explained with its benefits and use.**
[1]

I. Introduction

The software development is mainly focused on embedded systems in the industrial environment, such as business, software etc. Because they are essential for popular functioning of a system. The embedded systems are small, electronic, and encapsulated with hardware device; they work as a component of the huge system. Embedded systems perform an important role in improving the capability and performance of the real time system [1].

In the software development methodology, Service Oriented Architecture (SOA) [15] [12] and MDA focuses on embedded systems that have been developed mostly through function-based analysis and with procedural languages like C for implementation purposes. There exists such tools that transforms the model automatically however, there is a need for tools that support the component-based implementation of target systems in an object-oriented methodology for a specific use. For example, some times the software systems are complicated and the industry requires a single component of that software system. At this point, an efficient tool is required that extracts the particular chunk or component in order to use in embedded systems. Model-driven Architecture (MDA) is one such process, and it is used in the industry. This process enables services for defining software models and facilitating transformations between different model types.

Apart from this, MDA reduces gap among modeling environments of different types, it shifts graphical models into code towards writing transformation rules for your models. Hence, code gets automatically generated and every time there is change in the model or in code, both views are synchronous. The purpose of this paper is to submit a method which states the importance of current MDA value in the industry to the people with basic knowledge in software engineering, and to the software designers that consider introducing MDA into their organization. We have considered few transformation that convert the UML Model into Constraint Satisfaction Problem (CSP).

II. Model-driven Architecture

Object Management Group (OMG) has been the leading carrier of interoperability, platform independence and flexible solutions. Successful application of CORBA, IIOP, IDL, and UML in the industry brought credit to OMG for providing platform neutral, interoperable, flexible solutions. Model-Driven Architecture (MDA) is the framework for next generation software development it contains a set of standards, like Meta Object Facility (MOF), XML Metadata Interchange (XMI), and UML. Apart from this, MDA's goal is to facilitate interoperability between different platforms in distributed and heterogeneous environments, by using models within the entire development process [3]. According to OMG specifications, MDA separates the work into development of two different types of models: Platform-independent Model (PIM) and Platform-Specific Model (PSM). To generate a model out of another, appropriate model transformations are to be applied. In order to better understand these concepts and how they are used in MDA framework, it is necessary to formerly explain these concepts.

978-1-4244-8003-6/10/$26.00

A. Platform-independent Model

A Platform-independent Model (PIM) does not contain any impurities of technical details. It focuses on capturing the concepts available in the domain; it is typically the first development step, once initial requirements analysis is finished[2]. A PIM manifests the entities present in domain and the relationships identified among them, like conceptual model in the case of a database design. All details belonging to the final plat-form on which the desired system would be implemented and/or deployed are not incorporated at this level.

A PIM could possibly be an input to next phase or it could be an output of a previous phase. It is common to have more than one implementations of the same business model due to variety of reasons. As such, development of PIM aims at developing the model that best describes the business entities and its processes free from the specifics of the deployment technologies like J2EE, .NET, C#, Oracle, etc.

B. Platform-specific Model

Platform-specific Model (PSM) is the result of applying proper transformation rules on PIM. The transformation rules exactly describe how and what platform-specific details are to be added to PIM and reflect the same as an application model. This model contains the peculiarities of a target implementation technology like what database platform the model would be deployed and/or target language details, e.g. which entities would act like entity or session beans in a J2EE implementation. The availability of transformation tools for required platforms is important for converting into PSM from PIM. Its possible with UML 2.0, to integrate the behavior in object diagrams which facilitates generation of more concrete code models. At present the availability and capability of such tools is limited, but after further advancements in this field,the situation will be change [3].

C. Model Transformation

Model transformation is a process of converting graphical constructs into another model with the same semantics. Automation of model transformation gives a higher level of correctness and also gives surety of accurate translation of functionality through models. The tools which are currently available, make possible to generate code stubs from the class diagram models. We have used several transformations in order to show the conversion of one model to another and its usage in the industry. From platform-independent model, the transformation engine produces the platform-specific model applying certain transformation rules as shown in Figure 1. The transformation rules command the transformation engine that which transformations to apply to the models to generate the source code. These rules comprises of the arrangement of the produced code, the language and the implementation of the model assemble, i.e. classes etc. Transformation rules also comprises of the platform-specific view and limitation [10].

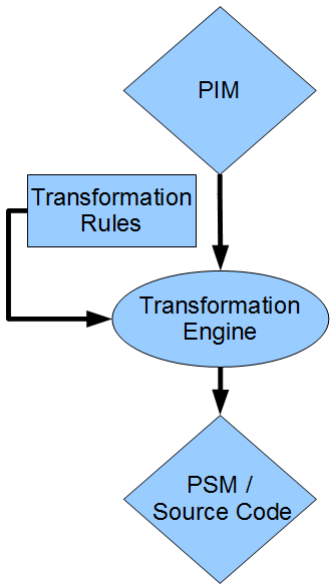

Fig. 1: High-level Description of PIM Transformations.

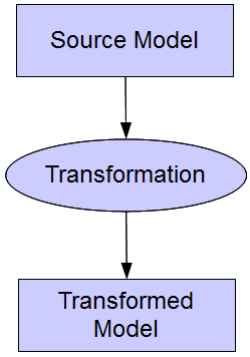

Fig. 2: Transformation From the Input Model to the Output Model.

III. Methodology

The methodology defines a framework which is used in this paper. A framework states a structure for describing a set of concepts, methods, technologies, that are necessary for a complete product design and manufacturing process. In this paper, both above concepts are complemented from a taxonomical and an industrial point of view respectively. This framework helps to organize the information and to provide an outline for the paper. The framework helps the reader/designer to understand and interpret the information contained in the paper. This framework is used for describing the different models and their transformations into other models. Figure 2 describes the general framework for model transformations. These models could be visualized as nodes and the transformations as edges. Each node can have several attributes or be located in a specific phase.

This framework is model oriented rather than process oriented and focuses on the models within the development phases. A set of an attribute and a domain is applicable for each model. As it is possible that certain models may

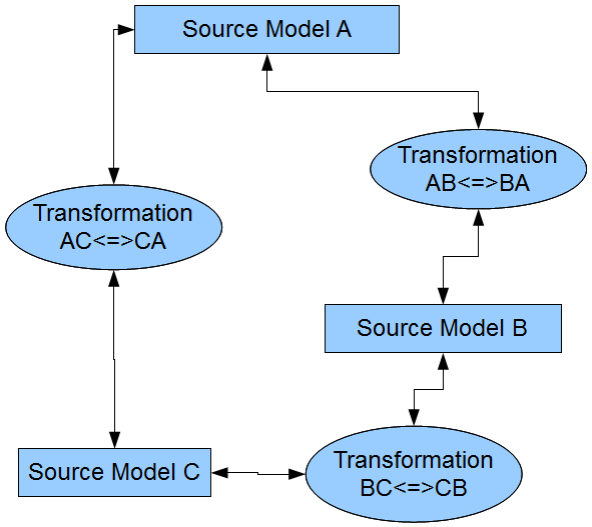

Fig. 3: Transformation From the Output Model to Another Output Model.

have common attributes, so it can be represented as a set of boundaries as shown in Figure 3. The transformations are unique edges, characterized by the tool to be used for them and attributes, like automation level, usage/maturity level can be linked to each other. It helps to analyse different paths independently for performing the transformations. Characteristics like supportability, maintain- ability, and portability could arise through the graph. Figure 3 shows how the graph is divided in two domains, one is transformation of AB, i.e. Step 1, and the other domain is transformation of BC or AC i.e. Step 2. Here Model A and Model B belong to the analysis phase, and Model C belongs to the design phase. Transformation BC is required in order to transform Model B into Model C.

For the framework, following parameters are concerned. These parameters are applicable to input and output model.

1) *Design Tools:* Software tools that are used for the implementation of input model.
2) *Notation:* A notation is a system of signs representing unique things. For high level computer languages, the notation is defined by syntax. For domain–oriented development, the domain terminology defines the domain notation [6]. The notation is classified as being either graphical or textual, and either standardized or unstandardized. There are few types of notation such Graphical Notation, Textual Notation, Standardized Notation and Unstandardized Notation
3) *Level of Implementation:* It is a method applied over models upon the level of detail or amount of information contained by them. For example, level of abstraction by which a system can be used fluently. Normally, the lower-level details are easily visible as compared to higher-level because of less complications. In this paper certain level of implementations are mapped in order to show the transformations in a more feasible way. The levels of implementation for this paper are: (1) Platform-independent Model (PIM) and (2) Platform-specific Model (PSM).
4) *Application of Transformations and its Usage:* The application of transformation describes the current status of the model, according to its usage in the industry i.e software or business.

IV. Results Obtained from Step 1

The process of results begins with collection and analyzation of the set of requirements given by the client. In this phase an overall architecture of the target system functionality is defined and module sketches are constructed which depicts the breakdown of the system into discrete parts of functionality. This specifies different objects of the overall system and clearly shows how they interact and collaborate.

The first transformation is based on sequence diagram that explains how groups of objects work together in achieving some system behavior. Using the transformation, sequence diagram contains one main object for business industry. From this component the industry can extract useful information in order to manufacture the desired system. The transformation helps to generate code automatically rather than implementing it manually. It is easy to draw a sequence diagram and generate the code without any extra time.

The hardware and the software components are identified and their requirements for each component are described. Mostly, these sub parts are provided by a third party manufacturer who specializes in building these smaller systems. This saves the cost of carrying out a whole independent activity and resource utilization; also these subsystems are tested and designed specifically according to car manufacturer's requirements.

A. System Sequence Diagram into JAVA

Input Model: System Sequence Diagram (SSD) has a number of different modes that effect its operation. These modes and the transitions between them can be illustrated with objects that work together. Figure 4 shows the sequence diagram.

1) *Design Tools:* Rational Rose for the design of sequence diagram [8].
2) *Notation:* Graphical based notation.
3) *Level of Implementation:* Sequence diagram is mostly used on PIM level.
4) *Application of Transformations and its Usage:* Sequence diagram is used in business industry for defining the sequence of the system. This model can be transformed in to PSM i.e JAVA code.

Output Model: JAVA Source Code: The output model is JAVA source code, which can be used to design the system as per sequence is given however, it can save the time of the developer. The sequence diagram focuses on entire system. Business industry can generate the code and use the specific component in to their design.

1) *Design Tools:* BOUML [4].
2) *Notation:* Code based.
3) *Level of Implementation:* Code is used on PSM level.

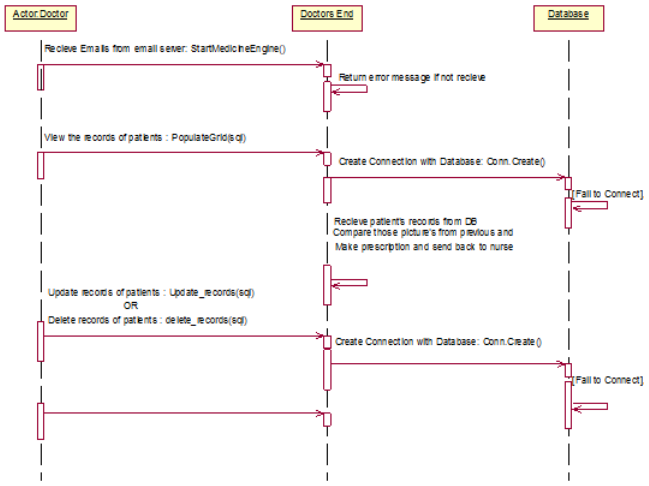

Fig. 4: Sequence Diagram of Tele-Wound^{TM} Application [14].

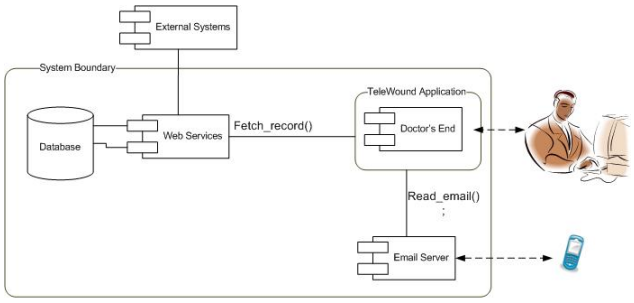

Fig. 5: Component of Tele-Wound^{TM} Application [14].

4) *Application of Transformations and its Usage:* It is used in business Industry.

B. Component Diagram into C++ Code

Input Model: Component Diagram has a number of different components of the system. The Component diagram of Tele-Wound represents the components of the tele-medicine application. Doctor's component represents the doctor's activities of an application. It also make contact with the external email server for receiving emails. The method read_emails() request the email from the email pop3 server and give it back to doctors component where the doctor's component receive all emails and separate patient's images and text descriptions and save it into databases. Furthermore, the component also need patients records so that the doctors can be able to treat their patients after viewing and analyzing their previous history records, fetch_record() will fetch the records from database to the doctors components by using web services. Figure 5 presents the component diagram of the tele-medicine health care system [13].

1) *Design Tools:* Microsoft Visio.
2) *Notation:* Graphical based notation.
3) *Level of Implementation:* Component diagram is mostly used on PIM level.

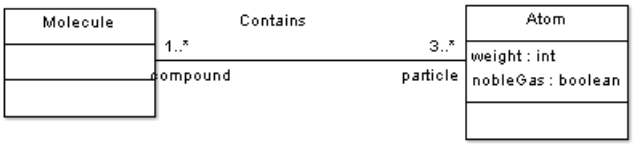

Fig. 6: UML/OCL Class Diagram (Atom-Molecule).

context Molecule **inv** InertNobleGases: self.weight $\geq$ 100

TABLE I: OCL Constraint (Atom-Molecule) class diagram.

4) *Application of Transformations and its Usage:* Component diagram is used in business/software industry for defining the components of the system. This model can be transformed in to PSM i.e C++ code.

Output Model: C++ Code: The output model is C++ source code, which can be used in system code. It will be component based therefore, each component of the system will be different and can be used independently in any software product.

1) *Design Tools:* BOUML [4].
2) *Notation:* Code based
3) *Level of Implementation:* Code is used on PSM level.
4) *Application of Transformations and its Usage:* It is used in business/software industry.

V. Results Obtained from Step 2

In the step 2 results, we have considered suitable algorithms for the functional requirements. Modern design packages and tools support for the creation of functionality models and evaluate functionality through object diagrams. Object diagrams are designed and evaluated with the help of UML class diagrams. The end result is a set of object diagram which is generated using Constraint Satisfaction Solver ECL^iPS^e [16]. Tools like UMLtoCSP [5], USE [7], Alloy [9] supports the transformations of UML class diagrams into object diagram using several transformation techniques.

A. UML/OCL Model to Object Diagram Using CSP

Input Model: UML/OCL models are designed in order to provide a high-level description of a software system which can be used as a piece of documentation or as an intermediate step in the software development process. In the context of Model Driven Development (MDD) and MDA, the need for correct specifications arises because the whole technology is based on model transformation. This transformation shows model correctness in terms of verifiability. Figure 6 shows the class diagram of Atom-Molecule while Table I represents the OCL constraints of Atom-Molecule class diagram. After the verification, this model can be used in any software product based on the requirements of the software industry.

1) *Design Tools:* ArgoUML for the design of UML class diagram [2] and any text editor for OCL [11] constraints.
2) *Notation:* Graphical based notation.
3) *Level of Implementation:* Class diagram used for PIM level and OCL invariants on PSM level.

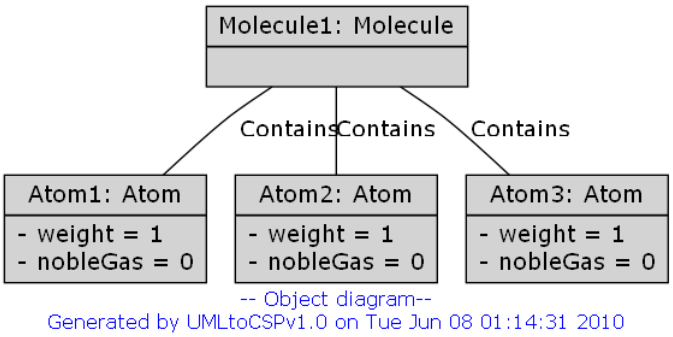

Fig. 7: Object Diagram (Atom-Molecule).

4) *Application of Transformations and its Usage:* Class diagram is used in software industry for defining the high level design. The model can be transformed into PSM and PIM.

Output Model: CSP Code and Object Diagram: The output model is CSP code and object diagram, which is verified and therefore, can be used to the components of the software product.

1) *Design Tools:* UMLtoCSP [5].
2) *Notation:* Code based and graphical based.
3) *Level of Implementation:* Class diagram used for PIM level and CSP code on PSM level.
4) *Application of Transformations and its Usage:* It is used in software industry. Figure 7 shows the object diagram of Atom-Molecule as a verified model.

VI. Testing in Model-driven Architectures

Testing of PIMs needs a systems approach, it can be done during the initial stages while mapping into a platform, integration, and through subsequent reuse. In most cases, white box testing is used for PIM behavior in the development environment as well as the target platform.

Initially, the methods verifies the internals of a PIM for correctness as it executes, solving the observability and controllability obstacles. Following is the junit test code sample used for testing the PIM and PSM models.

```
import junit.framework.TestCase;
public class BankEJBimplement
sessionBean extends TestCase {

        public static void main(String[] args) {}

        public BankEJBimplement sessionBean String name) {
                super(name);
        }

        protected void setUp() throws Exception {
                super.setUp();
        }

        protected void tearDown() throws Exception {
                super.tearDown();
        }
    public void sayTest()
    {
        BankEJB test = new BankEJB();
        assertNull("Successful!", test.sayTest());}
```

In order to test the entire model, we have to make a separate test cases for each class. The results will be generated once the test cases are executed.

VII. Conclusion

In this paper, we have presented that how MDA can be used in an industrial environment. For that purpose, we have shown few transformations widely used in the industry. Through these transformations, following benefits can be achieved:

1) Time-to-market and management costs both can be reduced: designers and developers can work on multiple projects at a time.
2) Automatic conversion or transformation of design into source code minimizes the risks that can cause by the loss of coding or personnel working on the project.
3) These transformation might generate the testing processes that are less time-consuming and less demanding.
4) Testing is more easier in Model-driven Architecture as it supports both white and black box testing. If the transformation is component-based, it would be more easier to test rather than entire system.

As a future research, we want explore several projects involving complex methodologies of MDA to observe the results. Few possible research could be Verification and Validation of models through transformations.

References

[1] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son. Quantifying and suppressing the measurement disturbance in feedback controlled real-time systems. *Real-Time Systems*, 40(1):44–76, 2008.
[2] ArgoUML. Argouml. http://argouml.tigris.org/.
[3] R. Bendraou, S. Bouzitouna, and M.-P. Gervais. From mda platform-specific model to code generation: Coupling of rm-odp and uml action semantics standards. In *Software Engineering Research and Practice*, pages 407–416, 2004.
[4] BOUML. Bouml. http://bouml.free.fr/.
[5] J. Cabot, R. Clarisó, and D. Riera. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In *ASE'2007*, pages 547–548. ACM, 2007.
[6] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, 2006.
[7] M. Gogolla, J. Bohling, and M. Richters. Validating UML and OCL models in USE by automatic snapshot generation. 4(4):386–398, 2005.
[8] IBM. Rational rose enterprise. http://www.developers.net/ibmshowcase/view/249.
[9] D. Jackson, I. Schechter, and H. Shlyahter. Alcoa: the alloy constraint analyzer. In *Proc. of the 22nd Int. Conf. on Software Engineering(ICSE'00)*, pages 730–733, New York, NY, USA, 2000.
[10] Model-driven Archiecture. An introduction to mda. http://www-128.ibm.com/developerworks/rational/library/apr05/brown/.
[11] OMG. Object constraint language specification. http://www.omg.org/technology/documents/formal/ocl.htm.
[12] A. Shaikh, M. Memon, N. Memon, and M. Misbahuddin. The role of service oriented architecture in telemedicine healthcare system. *Complex, Intelligent and Software Intensive Systems, International Conference*, 0:208–214, 2009.
[13] A. Shaikh, M. Misbahuddin, and M. S. Memon. A system design for a telemedicine health care system. In *IMTIC*, pages 295–305, 2008.
[14] A. Shaikh and M. Muhammad. A system design for a tele-medicine health care system, 2007.
[15] A. Shaikh, A. Soomro, S. Ali, and N. Memon. The security aspects in web-based architectural design using service oriented architecture. *Information Visualisation, International Conference on*, 0:461–466, 2009.
[16] The ECL^{i}PS^e Constraint Programming System. http://www.eclipse-clp.org, mar 2007. version 5.10.