Tools for MDA Software Development: Evaluation Criteria and Set of Desirable Features

Tihomir Calic, Sergiu Dascalu, Dwight Egbert
CSE Department, University of Nevada, Reno
{calict, dascalus, egbert}@cse.unr.edu

Abstract

Model Driven Architecture (MDA) is a new approach to software development that moves standard code-centric software development to model-centric software development. The basic idea is to specify system functionality with a platform independent model and then translate this model into platform specific model(s) and fully executable source code. MDA provides interoperability capabilities between different technologies, simplifies the work of software engineers, reduces software development costs, and supports adaptation to rapid changes in technology. Based on the development of a software application, a Glossary Management Tool, this paper proposes an evaluation framework for MDA tools and outlines with a set of software requirements the "portrait" of an ideal MDA tool.

1. Introduction

MDA is currently one of the most challenging and innovative areas of research in the software engineering field. Essentially, the main idea of MDA is to abstract software applications on a higher level through visual models. That means that MDA shifts the traditional code-centric software development paradigm to the new model-centric software development paradigm. There are several reasons behind this initiative, as follows.

Today, in the software industry there is a wide variety of existing technology platforms that, due to the rapidly changing technology, become obsolete over time. Every time a new technology arrives, software companies have to redesign their business systems or build them from the scratch. However, this is expensive, so the problem is to find an approach that could protect our investments in technology against its changes. One such approach is the MDA, which separates system functionality from its implementation in a specific technological platform [1]. In order to accomplish this, MDA involves software modeling and defines two core models: the Platform Independent Model (PIM), and the Platform Specific Model (PSM) [1]. PIM models the system's functionality, while PSM models the system implementation details. MDA sees PIM as a universal and long-lived model that will survive changing technology and be reused in the future to adapt existing systems to new technologies. The PIM is used for the generation of PSM models, from which can be then generated source code for the target application. Those transformations are achieved using the MDA supported tools. The PIM can also be used to adapt an existing system to other existing platforms, if the system is initially built or adapted to support MDA principles.

The other envisioned advantage of using the MDA approach is increased productivity. Since this approach can generate a large amount of code from models, it saves time in the development of a process and it delivers software solutions faster. Moreover, MDA tools incorporate the design patterns, templates, and best practices of leading software experts, which means that by using the MDA tools it is possible to produce higher quality code than when applying traditional software development methods. Finally, MDA relies on open standards, hence there are no extra costs for MDA adoption[1].

The remaining of this paper, based on work presented in [2], is organized as follows: Section 2 provides background information on MDA; Section 3 presents the software specification and design of the Glossary Management Tool used as case study; Section 4 proposes a set of evaluation criteria for MDA tools; Section 5 presents the "portrait" of an ideal MDA tool; Section 6 outlines related research work, and, finally, Section 7 contains pointers to future work and presents the conclusions of the paper.

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.241

IEEE computer society

2. Background on MDA

In 2001, the Object Management Group (OMG) [1] proposed a new methodology called Model Driven Architecture (MDA) [3]. The OMG is a non-profit organization that creates and implements standards for software product development in the area of object-oriented technology. Model Driven Architecture is based on several standards defined by the OMG, specifically: the Unified Modeling Language (UML) [4], the Meta-Object Facility (MOF) [5], the XML Metadata Interchange (XMI) [6], and the Common Warehouse Metamodel (CWM) [7]. As indicated by [1], there are many industrial sectors where MDA can be applied including finance, e-commerce, manufacturing, healthcare, and other.

MDA relies on the separation of the business logic of a system from its implementation. To achieve this, MDA defines two types of models: the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM). PIM model captures system behavior and functionality, while PSM model captures information about details of system implementation. Both models should be described with a modeling language that relies on OMG's Meta-Object Facility (MOF) standard. The basic MDA concepts are briefly presented next, based on information from [1].

Model is an abstraction of the functionality and behavior of a system that we are observing [8]. Representation of a model in MDA must be formal, which means that it needs to be described by a language that has a clearly defined syntax [9]. In addition, that language should be based on OMG's MOF standard.

Platform is an environment where models will be executed. The platform is independent from the functionality of a system. Examples of platforms are Java2, CORBA, Microsoft .NET, Web Services, the operating systems Linux, Solaris, Windows, etc. [10].

Platform Independent Model (PIM) captures the business logic of the system that we are building and must be independent of any implementation technology. Each PIM can be transformed to one or more Platform Specific Models (PSMs). Any system can be partitioned into one or more domains that represent different subject matter areas and each of them is represented by one PIM model. Since the same domain can be found in different systems, a PIM model can be reused with or without modifications in future modeling problems. This is the main reason why the MDA approach does not incorporate any implementation details in the PIM model. In this way, business logic captured in a PIM model lasts much longer then the logic expressed by a programming language that can eventually become obsolete over time. Furthermore, the platform-independency of the PIM model provides its portability across many different platforms and interoperability between different platforms [3], [8].

Platform Specific Model (PSM) contains the business logic that is already expressed in the PIM model and the information about platform implementation details. The PSM model is always generated from the PIM model. There are two basic types of PSM models: one is a UML model and the second is a source code. When the PSM model is expressed in a source code it is called Platform-Specific Implementation (PSI) [11]. Each time when business requirements need to be changed, those changes need to be made in the PIM model, not in the PSM model, as the PSM models will be regenerated from the modified PIM.

Mappings between models together with markings of the models are one of the key features that MDA tools need to provide to the developer. *MDA mapping* is a set of rules and techniques for translating a PIM model into another PIM model or into a PSM model. There are several different types of mapping in MDA [3].

Model transformation in MDA is a process of conversion of a PIM model to a PSM model. Both the model type mapping and the model instance mapping are supported in model transformation. OMG recently defined the standard for model transformation in MDA, called Queries/Views/Transformations (QVT) [12], but that standard is still under development.

Meta-Object Facility (MOF) is an OMG standard that defines an abstract language used to describe metamodels [1]. MOF is very important in the MDA approach, especially in the case of interchanging models between different vendors' MDA tools. Thanks to employing the MOF standard, it is feasible to manage the same PIM model by different vendors' MDA tools. Naturally, each MDA tool has to be MOF-compliant and the language for expressing the models has to be an MOF-based modeling language such as the UML. The MOF standard has also a major role in defining the mapping function between models, particularly from PIM to PSM.

Common Warehouse Metamodel (CWM) is an OMG standard that defines modeling metadata in a data warehouse environment. The main goal of CWM is to enable the exchange of metadata between data warehousing tools and data warehousing platforms. CWM is based on three OMG's standards: MOF, UML, and XMI [1]. As an MOF-compliant standard, CWM can use all OMG specifications that rely on MOF. One of these specifications is XMI, which provides interchange metadata expressed using the CWM metamodel. The basic modeling language for

representation of CWM metamodels is UML. In addition to UML, CWM uses the Object Constraint Language (OCL) [13] to represent additional constraints on the CWM metamodel [7]. The CWM specification covers the full life-cycle of design, deployment, and management of data warehouse applications. In MDA, CWM has the most important role in the mapping from PIM models to database schemes.

The *XML Metadata Interchange* (XMI) is an OMG standard that provides interchange of metadata information between modeling tools, repositories, and middleware supporting the MOF metamodel [6]. XMI is built upon the Extensible Markup Language (XML) [14], a standard developed by the World Wide Web Consortium (W3C) that defines data exchange between distributed environments [15]. In MDA, XMI is mostly used to interchange UML models between MDA tools and to map UML models to XML textual format. To achieve this, each MDA tool needs to support the import and export of the XMI file format. There are two kinds of information that need to be exchanged. One is the information about the elements that constitute the UML model (classes, attributes, associations, state transitions etc.) and the second is how these elements are represented in UML diagrams (positions of graphical symbols, shapes, colors, fonts, etc.). The XMI document can also be transformed into various programming language formats using the W3C Recommendation called Extensible Stylesheet Language Transformations (XSLT) [16].

MDA tools have a major role in MDA-based software development. They provide development of software applications by creating a PIM model of a system and its transformation and mapping it into the PSM model. MDA tools also provide further transformation of the PSM model to the fully executable program code in several programming languages. Currently, the OMG did not specify a document that indicates which features a modeling tool needs to incorporate to be MDA-compliant. However, according to the OMG website [17] today on the market there are over fifty MDA tools that support one or more major features of the MDA approach. Each of them has different strengths, so it is up to developers to choose a tool that best fits their needs.

3. Case Study: Project Glossary Management Tool

As a case study for MDA tool exercising we used a relatively simple web application called Glossary Management Tool (GMT). GMT is supported by a database and provides the regular database CRUD (Create/Report/Update/Delete) functions. By using the GMT a user can add, delete, search or modify various terms from the GMT's database. While universally necessary and highly practical, the GMT application is not very difficult to develop using the standard, code-centric software development techniques, but in our work it has been developed by using MDA tools only. Specifically, GMT has been implemented in three versions using three different MDA software environments (tools), two proprietary and one open source. The purpose of this development was to identify, classify, and evaluate features needed in MDA tools.

GMT is based on a three-tier client-server architecture that consists of three layers: presentation layer, application processing layer, and data layer (Figure 1).

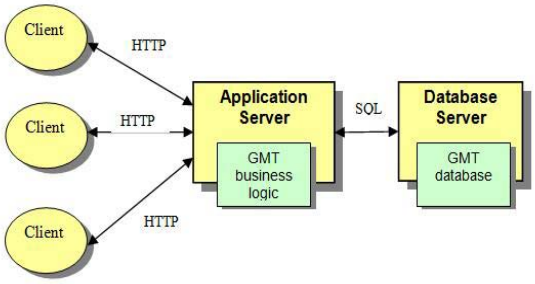

Figure 1: GMT architecture

All these layers are logically separated. The first layer, the presentation layer, is responsible for presenting the data to the users and for managing the interactions with the users. This layer allows the users (clients) to interact with GMT through any standard web browser. The second layer, the application processing layer, is concerned with providing the business logic of the GMT and is supported by an application server. The third layer, the data management layer, is responsible for storing and retrieving data from the database. This layer is provided by the database server [18].

The essential UML class diagram of the GMT is shown in Figure 2. More details of the GMT software model and its variant MDA-based implementations are available in [2].

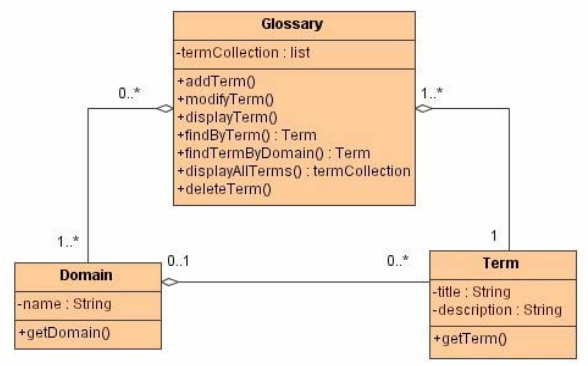

Figure 2: GMT class diagram

4. Evaluation Criteria for MDA Tools

After the repeated implementation of the GMT web application using three different software tools that support the MDA approach we put together a set of criteria for evaluating MDA software environments. To achieve this, we combined evaluation criteria from other reported works [21, 22, 23, 24, 25, 26, 27] and, in addition, we included several other criteria that we found useful based on our own experience. As a result, we came up with the following classification that consists of six main criteria groups:

1. **MDA features** - This criteria group evaluates to what degree MDA tools are compliant with the OMG's MDA specification.
2. **Tool capabilities** - This criteria group evaluates tool capabilities including the selected tool features and tool's environment.
3. **Quality** - This criteria group evaluates the overall quality of the MDA tool, including its efficiency, understandability, ease of implementation, and other.
4. **Usability** - This criteria group evaluates the quality of interaction between users and the MDA tool.
5. **Productivity** - This criteria group evaluates to what degree the expected MDA benefits are actually achieved by using the MDA tool.
6. **Documentation** - This criteria group evaluates the supporting documentation of the MDA tool such as available tutorials, samples, on-line help, and other.

Each criteria group is expanded into a set of sub-criteria. All sub-criteria together with their references, where applicable, are presented in Table I.

5. Portrait of an Ideal MDA Tool

Based on our research exploration and experience with MDA tools acquired during the implementation of the GMT case study application using three different MDA software environments, below is outlined the "portrait" of an "ideal" MDA tool. For this purpose, the concise and practical style for writing functional requirements proposed in [19] has been used. The ideal MDA tool shall:

1. Provide a UML graphical editor that supports UML modeling of all types of UML diagrams. UML extensions (profiles) shall also be supported.
2. Provide support for modeling the Platform Independent Model (PIM) of a system that will contain only system logic and will not contain any information about system implementation. The dynamic behavior of the system shall be expressed using the supported UML diagrams and the specific language that describes the system behavior (such as OCL).
3. Automatically generate a Platform Specific Model (PSM) for the chosen target platform directly from the modeled PIM. The tool shall support the generation of PSMs for all the leading platforms available on the market and in the open source community.
4. Support the modification of PSM models in order to meet specific user requirements.
5. Support the validation and verification of models in order to check their consistency with the system specification and to verify the correctness and completeness of the designed models.
6. Support both model-to-model and model-to-code transformations via plug-in cartridges. The tool shall provide a graphical development environment for the development of new cartridges and for the customization of existing ones.
7. Include a graphical user interface designer component that will allow easy development of the application's user interface and will be able to meet the most ambitious user interface requirementss.
8. Generate the entire application code and project infrastructure required for running the application. The tool shall provide the implementation of the application in any desired programming language.
9. Support the debugging process for both model-to-model and model-to-code transformations.
10. Provide automatic deployment of the generated application to all the leading production servers available on the market and in the open source community. The tool shall provide testing support

for all versions of the leading web and application servers and databases available on the market and in the open source community.

11. Support automatic generation of the project documentation.
12. Support exporting of models into the XMI format in order to exchange the models with other MDA and UML tools. The tool shall support importing of models from all leading MDA and UML tools available on the market and in the open source community without losing any information during the import process.
13. Support reverse-engineering of legacy software systems in order to create UML models from any

Table 1: Evaluation criteria for MDA tools

	FEATURES		REFERENCES
1. MDA Features	1.1 PIM support 1.2 PSM support 1.3 Multiple target platforms 1.4 Marking 1.5 Mapping 1.6 Transformations	1.7 Model import/export 1.8 Reverse engineering 1.9 Standardization 1.10 UML profiles 1.11 UML diagrams types 1.12 OCL support	[3]
2. Tool Capabilities	2.1 UML graphical editor [21] 2.2 Managing model complexity [20] 2.3 Zooming [20] 2.4 Automatic GUI creation 2.5 Modeling GUI 2.6 Validation and Verification of Models [20] 2.7 Defining New Transformations [23] 2.8 Debugging [21]	2.9 Customization of Generated Code 2.10 Protected Areas [21] 2.11 Deployment [20] 2.12 Testing [20] 2.13 Execution of Generated System [22] 2.14 IDE Integration [21] 2.15 Currency of Supported Software 2.16 Automatic Report Generation	[20], [21], [22], [23], own experience
3. Quality	3.1 Efficiency [24] 3.2 Simplicity 3.3 Robustness [24] 3.4 Quality of generated GUI	3.5 Understandability [24] 3.6 Ease of implementation [24] 3.7 Completeness [24] 3.8 Ability to produce expected results [24]	[24], own experience
4. Usability	4.1 Learnability [25], [26] 4.2 Visibility [25], [26] 4.3 Feedback [25], [26] 4.4 Constraints [25], [26]	4.5 Mapping [25], [26] 4.6 Consistency [25], [26] 4.7 Affordance [25], [26]	[25], [26]
5. Productivity	5.1 Reduced development time [27] 5.2 Reduced complexity of implementation [27] 5.3 Reduced level of skills [27]	5.4 Code quality [27] 5.5 Cost effectiveness	[27], own experience
6. Documentation	6.1 Organization [24] 6.2 Samples 6.3 On-line help [24]	6.4 Understandability [24] 6.5 Completeness [24] 6.6 Quality [24]	[24], own experience

legacy application written in any programming language.
14. Provide integration with all leading programming software development environments available on the market and in the open source community.
15. Provide teamwork support to allow multiple users to work simultaneously on the same project.
16. Include complete documentation and samples that effectively explain to users how to use the tool.
17. Provide support for maintenance and evolution of the developed application.
18. Run on multiple operating systems.
19. Have good performance (CPU, memory and disk usage) on an average PC workstation.

6. Related Work

In order to put together a reasonably comprehensive set of criteria for evaluating MDA software environments we first researched several existing common methods for evaluating software engineering tools in general. In particular, we found very useful the research work conducted by Dr. Barbara Kitchenham at the University of Keele, UK [24]. Her research project is known as DESMET. Among nine different types of evaluation that are identified by DESMET the type named Feature Analysis Case Study is most similar to the approach that we used in our work. In her work, Kitchenham explains how to generate a set of features for a software tool or a method, and identifies some of the features common to software tools and methods. In a similar direction, the research work conducted within the MODA-TEL project coordinated by Eurescom identified MDA tool requirements and separated them into four groups: business modeling, model transformation, artifact generation, and legacy integration [21]. The requirements proposed within the mentioned groups are primarily designed to meet MDA-specific features and some specific tool capabilities. Another valuable related research work was conducted by Tariq and Akhter [22]. Their work identified features from current MDA specification and some specific general tool's features. Furthermore, research groups at King's College London and York University identified sixteen properties that a tool needs to have to be MDA compliant [27]. Besides the related works mentioned above, there are several other sources that we have consulted [23], [28], [29].

Based on this work by other researchers, we have expanded and further organized the evaluation criteria software professionals are likely to need when deciding what MDA tools to use. Furthermore, based on practical experimentation and application of the criteria, we have proposed a set of desirable features for MDA tools, as detailed in Section 5 of this paper. For both classification of evaluation criteria and set of useful features that we suggested it is important to point out that specific weights or priorities can be assigned on case by case, depending on various factors such as: project goals, project constraints, type of application, developers' experience, and project management priorities and approach.

7. Conclusions and Future Work

In this paper we have explored the MDA approach for software development. Our exploration was meant to investigate the feasibility and usefulness of the MDA approach. Throughout the process of exploration we had two major tasks. First, we have implemented a fully functional web application in three different versions using three different software tools that support and implement the MDA approach and, based on this experience as well as the review of related literature, we compiled a set of criteria for evaluating MDA software environments. Second, we outlined a "portrait" of a most desirable MDA tool. In addition to proposing a set of evaluation criteria and suggesting features for an "ideal" MDA tool we also identified a number of directions of future work that could be beneficial to pursue. Some suggested areas for future research and development are as follows:

- Experimenting with a larger number of MDA environments, both from commercial and open-source domains.
- Exploration of MDA environments (three or more) using a more complex case study application than the Glossary Management Tool used in our exploration.
- Exploration of MDA environments (three or more) using an extended version of our set of evaluation criteria (along the lines indicated in Section 4).
- Specification and design of the proposed "ideal" MDA tool.
- Building the prototype of the proposed "ideal" MDA tool.
- Exploration of MDA tool support for PIM and PSM models on more details.
- Experimentation with customizing and extending the model transformation rules within existing particular MDA tool cartridges.
- Development of a new cartridge for a programming language not currently supported for a particular MDA tool (e.g. for Python [30]).

- User group evaluation of MDA tools that involves a large group of developers that apply the evaluation criteria proposed in this paper. The outcome results will be more objective since they will result from combining developers' opinions.

In summary, the main contributions of this paper are as follows:

- Proposal of an evaluation framework consisting of six groups of assessment criteria and 54 detailed criteria.
- Summary of the most relevant and desirable features of an "ideal" MDA tool.
- Suggestions of several potentially rewarding directions of future research and development.
- Design and implementation of the Glossary Management Tool (GMT) web application as a case study for exercising MDA tools and for exploring various aspects of software modeling and implementation using MDA.

References

[1] Object Management Group, accessed September 1, 2007 at http://www.omg.org

[2] Calic, T., *Exploration of Model Driven Architecture Capabilities via Comparative Utilization of MDA Tools*, MS thesis, University of Nevada, Reno, USA. December 2006.

[3] "MDA Guide Version 1.0.1", OMG's website, published June 2003, accessed September 4, 2007 at http://www.omg.org/docs/omg/03-06-01.pdf

[4] "Unified Modeling Language (UML), Version 2.0", OMG's website, published August 2005, accessed September 4, 2007 at http://www.omg.org/technology/documents/formal/uml.htm

[5] "Meta Object Facility (MOF) Core Specification", OMG's website, published January 2006, accessed September 4, 2007 at http://www.omg.org/docs/formal/06-01-01.pdf

[6] "MOF 2.0/XMI Mapping Specification, v2.1", OMG's website, published September 2005, accessed September 4, 2007 at http://www.omg.org/docs/formal/05-09-01.pdf

[7] "Common Warehouse Metamodel (CWM) Specification", OMG's website, published March 2003, accessed September 4, 2007 at http://www.omg.org/docs/formal/03-03-02.pdf

[8] Mellor, S., Scott, K., Uhl, A., and Weise, D., *MDA Distilled,* Addison-Wesley Professional, 2004.

[9] Dascalu, S., *Combining Semi-Formal and Formal Notations in Software Specification: An Approach to Modelling Time-Constrained Systems,* PhD thesis, Dalhousie University, Halifax, NS, Canada, 2001.

[10] Arlow, J., and Neustadt, I., *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML,* Addison-Wesley, 2004.

[11] Raistrick, C., Francis, P., Wright, J., Carter, C., and Wilkie I., *Model Driven Architecture with Executable UML*, Cambridge University Press, 2004.

[12] "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification", OMG's website, published November 2005, accessed September 17, 2007 at http://www.omg.org/docs/ptc/05-11-01.pdf

[13] "Object Constraint Language Specification", OMG's website, published May 2006, accessed September 17, 2007 at http://www.omg.org/docs/formal/06-05-01.pdf

[14] "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C's website, published February 2004, accessed September 17, 2007 at http://www.w3.org/TR/2004/REC-xml-20040204/

[15] "W3C", World Wide Web Consortium website, accessed September 17, 2007 at http://www.w3.org

[16] "XSL Transformations (XSLT) Version 1.0", W3C's website, published Nov. 1999, accessed Sept. 17, 2007 at http://www.w3.org/TR/1999/ REC-xslt-19991116

[17] "Committed Companies and Their Products", OMG's website, published May 2006, accessed Sept. 17, 2007 at http://www.omg.org/mda/committed-products.htm

[18] Sommerville, I., *Software Engineering*, Addison-Wesley, 2004.

[19] Arlow, J., and Neustadt, I., *UML and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley Professional, 2001.

[20] "Model Driven Architecture (MDA)", OMG's website, published July 2001, accessed October 8, 2007 at http://www.omg.org/docs/ormsc/01-07-01.pdf

[21] Renuncio, L.,E., *MODA-TEL (Deliverable 3.4) MDA Foundations and Key Technologies*, MODA-TEL Consortium, July 2004.

[22] Tariq, N. A., Akhter, N., *Comparison of Model Driven Architecture (MDA) Based Tools*, MS thesis, Royal Institute of Technology, Stockholm, Sweden. 2005.

[23] Janssen, J. W., *Evaluation of current tool support for the Model Driven Architecture*, MS thesis, University of Twente, Enschede, Netherlands. January 2004.

[24] Kitchenham, B., *DESMET: A Method for Evaluating Software Engineering Methods and Tools*, Technical Report TR96-09, University of Keele, UK. 1996.

[25] Norman, D., *The Design of Everyday Things*, Basic Books, 2002.

[26] Preece, J., Rogers, Y., Sharp, H., *Interaction Design,* Wiley Publishing Inc., 2002.

[27] "An Evaluation of Compuware OptimalJ Professional Edition as an MDA tool", King's College London and University of York, published September 2003, accessed October 8, 2007 at http://www.lcc.uma.es/~av/MDD-MDA/publicaciones/P_13kings_mda.pdf

[28] Najim, M., *A Study of Model Driven Architecture Approaches*, MS project, University of California, Riverside, Riverside, US. June 2005.

[29] Chatterjee, S., "MDA Tools Evaluation", Jax Magazine, published February 2006, accessed October 15, 2007 at http://www.jaxmagazine.com/itr/column/psecom,id,4,nodeid,354.html

[30] "Python Programming Language", Python official website, accessed Oct. 2007 at http://www.python.org/