An MDA-based System Development Lifecycle

Mohsen Asadi
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
mohsenasadi@Mehr.sharif.edu

Mahdy Ravakhah
Azad University of Mashhad
Mashhad, Iran
ravakhah@gmail.com

Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
ramsin@sharif.edu

Abstract

OMG's Model Driven Architecture (MDA) has deeply influenced modern-day software development, not only by providing promising means for automating the software process, but also through revitalizing the role of modeling in software development, the importance of which had been neglected during the recent euphoria over lightweight development methods. However, MDA's need to remain reasonably abstract means that it is more a software development approach rather than a standalone methodology, and therefore needs methodology support to be practically useful. Several MDA-based methodologies exist today, yet the need remains for the definition of an instantiable MDA-based development process.

We propose a generic lifecycle for MDA-based software development that can be used as a basis for constructing MDA-based methodologies through a Method Engineering (ME) process. The phases and activities of the proposed lifecycle are described herein, with a number of prominent MDA-based methodologies assessed as to their degree of conformance to the proposed lifecycle.

1. Introduction

The *Model-Driven Architecture* (MDA) defines an approach to information systems specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. At the core of MDA there are a number of important OMG standards: The Unified Modeling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), and Common Warehouse Metamodel (CWM). These standards define the core infrastructure of the MDA, and have greatly contributed to modern systems modeling and development [1].

MDA strives to automate software development through providing an approach for specifying systems in terms of models, and then enacting the development process through performing model transformations. System requirements are specified in the *Computation Independent Model* (CIM). The *Platform Independent Model* (PIM) is the model that describes the system design independent of the implementation platform. The *Platform Specific Model* (PSM), on the other hand, describes the system design in the form of a platform-dependent model. These three levels of models in MDA raise the level of abstraction of traditional platform-dependent design.

Despite its influence on Model-Driven Development (MDD), MDA does not provide a concrete methodology for software development. A Software Development Methodology (SDM) is a framework for applying software engineering practices with the specific aim of providing the necessary means for developing software-intensive systems [2]. A methodology consists of two main parts: a set of modeling conventions comprising a Modeling Language (syntax and semantics), and a process which provides guidelines about the order of the activities and specifies the artifacts developed using the modeling language. According to the above definition, MDA is not a methodology, but rather an approach to software development. It has therefore become imperative for software developers to have MDA-based methodologies at their disposal for enacting and realizing MDA practices and standards. This fact has forced organizations willing to adopt the MDA for system development to either transform their software development methodologies into Model-Driven Development (MDD) methodologies, or use new methodologies that utilize MDA principles and tools towards the realization of MDA standards.

While several MDA-based methodologies have been introduced in recent years, a *Situational Method Engineering* (SME) approach [3] supporting the construction of a *custom* MDA-based methodology – i.e. one that has been tailored to fit the project situation at hand – has not been suggested yet. The main reason is that SME approaches require a set of resources before the method engineering effort can commence: *Assembly-based* SME requires a library

978-0-7695-3136-6/08 $25.00 © 2008 IEEE
DOI 10.1109/AMS.2008.19

of process components; *Paradigm-based* SME requires generic instantiable process metamodels/lifecycles; and the *Extension-based* approach relies on extension patterns [4]. None of these exist in an MDA-compliant context.

We propose a generic MDA-based system development lifecycle that can be instantiated and used as the basis for engineering tailored-to-fit MDA-based methodologies through a Paradigm-based SME approach. Generic lifecycles with similar usefulness have previously been proposed for Object-Oriented methodologies [5] and Agile methods [6]. Our proposed lifecycle is specifically aimed at engineering methodologies supporting and facilitating MDA-based software development, and can therefore be used as a general framework for MDD.

This paper is organized as follows: The proposed MDA-based System Development Lifecycle (MDA-SDLC) is presented and described in Section 2; Section 3 contains the results of assessing existing MDA-based methodologies as to the their degree of conformance with the proposed MDA-SDLC; conclusions and areas for furthering this research are presented in Section 4.

2. Proposed MDA-based System Development Lifecycle

Our proposed MDA-SDLC has been developed using ideas from the generic software development lifecycle and the development approach prescribed by the MDA. As mentioned before, a software development methodology consists of two main parts: a Modeling Language (syntax and semantics) and a Process. Naturally, MDA-based methodologies use UML as their modeling language. The generic lifecycle is therefore mainly focused on the process part of the methodology.

Our proposed MDA-SDLC is not a concrete methodology, but a general process that defines the phases and activities expected to be present in an MDA-based methodology. It can therefore be specialized (instantiated) to fit the project situation at hand. MDA-SDLC consists of five phases (Figure 1): *Project Initiation*, *PIM Development*, *PSM & Code Development*, *Deployment*, and *Maintenance*. As shown in Figure 1, returns to previous phases are usually necessary. The PIM Development phase and the PSM & Code Development phase are typically performed in an iterative-incremental fashion. The phases are described in detail in the following sections.

2.1. Project Initiation Phase

As with all other methodologies, MDA-based methodologies require a project initiation phase. Figure 2 shows the activities performed in this phase and the artifacts produced. The following activities are performed in this phase:

CIM definition*:* the scope of the software system is clearly defined through problem domain analysis. At the end of this activity, an unambiguous black-box definition of the system, its objectives, and its scope should be produced.

Requirements specification*:* system requirements are specified using requirements engineering and requirements gathering techniques, and a Requirements Model is produced. The Requirements Model spans functional requirements (use cases and activity diagrams), non-functional requirements (forces), and atomic requirements.

Obtain funding and support*:* mainly concerning justifying the project via feasibility analysis, and

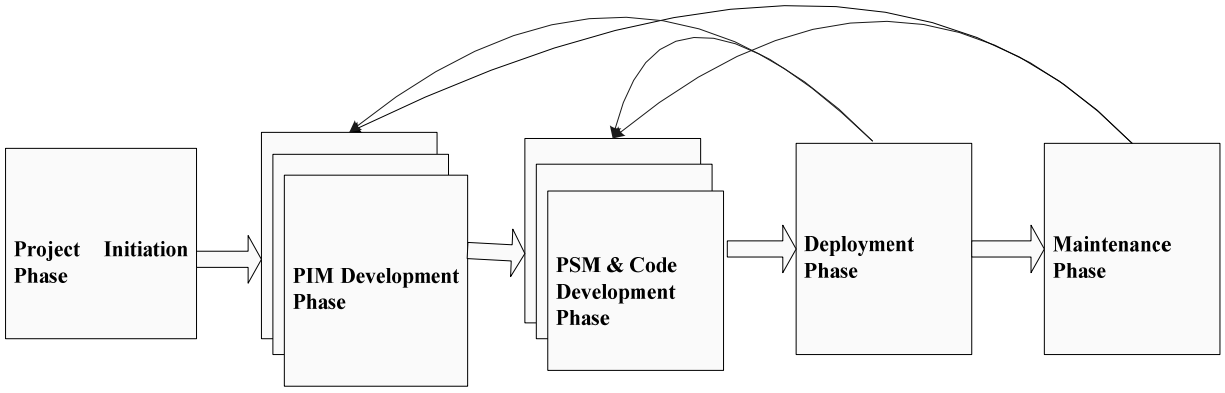

Figure 1. Proposed MDA-based SDLC

obtaining the necessary resources. Some activities may have already been achieved via portfolio management efforts.

***Initiate the team**:* the development team is organized. According to [7] the users of MDA technology are classified in three categories: *Knowledge builders*, whose responsibility is to build knowledge repositories, and consist of Architects, Platform Experts, Quality Engineers, and Methodology Experts; *Knowledge facilitators*, who assemble, combine and deploy knowledge, consisting of Project Managers and Quality Engineers; and *Knowledge users*, who apply knowledge, and consist of Designers and Software Engineers. The development team is composed of all the required people from all these three categories. It is not necessary to have a complete team from the start, as the team can be reorganized and completed during the development process.

***Define general plan**:* the development risk of each of the requirements and the development effort needed is estimated, based on which the requirements are prioritized. The general plan is then created. The plan is revised and completed during the development process.

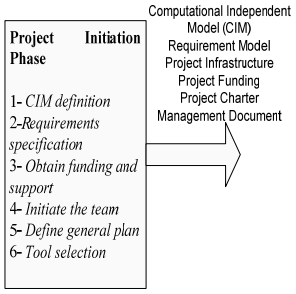

Figure 2. Main activities of the Project Initiation phase

***Tool selection**:* the tool typically handles many activities in an MDA-based methodology, such as the definition of models and metamodels, model transformations and code generation performed based on model information, and the definition of constraints and rules to verify model compliance [7]. In this activity, one or more tools are selected to support the activities in the development process. For the selection of appropriate tools, tool requirements are identified from a software engineering perspective and mapped to the capabilities of tools available on the market.

The outputs of this phase are: CIM, Requirements Model, project infrastructure, project funding, project charter and management documents.

2.2. PIM Development Phase

The objective of this phase is to create a complete and precise model of the structure and behavior of the system. This model is derived from the Requirements Model. The model must be platform independent, i.e. during the creation of this model nothing about the implementation platform is taken into account. The model typically contains activity diagrams, sequence diagrams, state machines, and class diagrams.

Figure 3 shows the activities and products of this phase. As shown in Figure 3, the inputs of this phase are from the project initiation, deployment, and maintenance phases. The activities of this phase are as follows:

***Produce analysis PIM**:* in this activity, a platform-independent analysis model is defined through analyzing the requirements model. System functionalities are described in the analysis PIM while maintaining traceability to the requirements model. Developers may use appropriate model elements stored in a model repository to produce some parts of this PIM. This model is not the final PIM, but forms the foundation for producing the final version. Conventional OO analysis techniques can be used for this activity, which is typically executed in an iterative and incremental fashion.

***Architectural design**:* in this activity, the system architect designs the system architecture. The general plan may be reviewed if necessary. The product of this activity is the main framework of the system.

***Produce design PIM**:* in this activity, the analysis PIM model is refined to produce the design PIM, which models the detailed structure and behavior of the solution (software application). Conventional OO design techniques can be used in this activity. The design PIM is derived from the analysis PIM in an iterative-incremental fashion. Constraints, preconditions, postconditions, and invariants are defined using UML and OCL mechanisms. Reusable domain-dependent design model elements can also be retrieved from model repositories for composing the design PIM.

***Verification/Validation**:* in this activity, we check whether the products of the modeling activity are free from defects and in compliance with the requirements established in the requirements modeling activity. We execute this step mainly to correct design PIM errors before transforming it into the PSM.

***Generalization**:* it is necessary to execute the generalization activity after creating the models in the previous activity. In this activity, tasks are performed to make models more reusable. Reusable domain-specific model elements are uploaded and categorized into repositories for reuse in future projects.

The outputs of this phase are: verified design PIM, project documents, management documents, and the requirements allocation matrix.

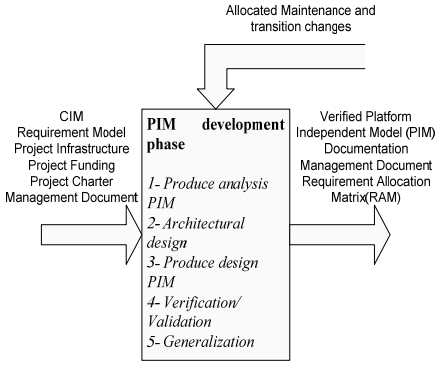

Figure 3. Main activities of the PIM Development phase

2.3. PSM & Code Development Phase

During the activities of this phase, the PIM produced in the previous phase is transformed into the PSM model. Most of the activities of this phase are performed by MDA tools. Figure 4 shows the activities and products of this phase. As shown in Figure 4, the inputs of this phase are from the PIM development, deployment, and maintenance phases. The activities of this phase are as follows:

Transform PIM to PSM*:* the PSM is produced from the PIM using MDA tools. Conventional guidelines are used to guide the developer in performing the transformation using the selected tool. These guidelines are provided by the tool or by the methodology itself.

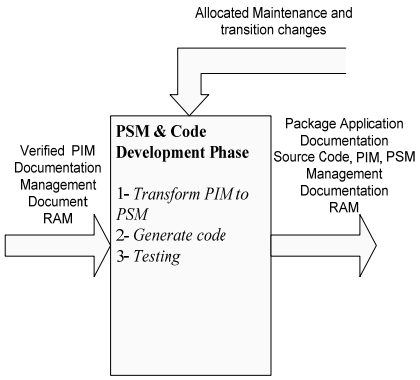

Figure 4. Main activities of the PSM & Code Development phase

Generate code*:* in this activity, the execution code is generated from the PSM using MDA tools. Current MDA tools cannot generate complete code from the PSM. As with the previous activity, conventional guidelines are used to guide the developer in performing the transformation using the selected tool. The developers then have to manually complete the generated code. This activity defines the organization of the code, executes unit tests, and integrates all components and subsystems.

Testing*:* this activity includes standard testing tasks such as: plan tests, prepare test model, prepare test cases and test scripts, execute tests, correct defects and document test results. Test cases are defined and applied on the code. Automatic testing is possible to some extent, but manual testing is usually necessary in order to complement the testing activities.

The outputs of this phase are: packaged application, documents, source code, PSM, management documents and the requirements allocation matrix.

2.4. Deployment Phase

The objective of this phase is to successfully deliver the developed system to the final user. Figure 5 shows the activities performed in this phase and the artifacts produced. As shown in Figure 5, the input of this phase is from the PSM & code development phase. The activities of this phase are as follows:

Final testing of the system*:* in this activity, final system and acceptance testing are performed. The developer may choose to pilot/beta-test the system with a subset of the actual users of the system. We intend to find and act on defects. To fix the defects, we may have to return to previous phases.

Finalizing system and user documentation*:* during the development cycle, some documentation may have been written. These should be finalized when the system is being prepared to be released. User manuals are also prepared during this activity. We may also gather and document the experience acquired during the current project to be used in future development efforts.

Transition of the system to the user environment*:* during this activity, the system is installed in the user environment, and related tasks (such as data conversion) are executed.

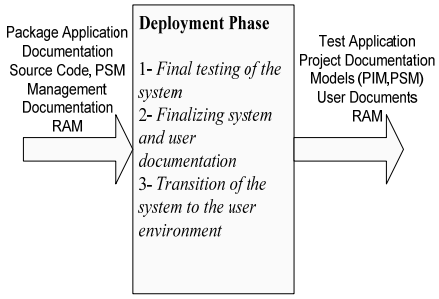

Figure 5. Main activities of the Deployment phase

The outputs of this phase are: tested application, project documentation, models (CIM, PIM, PSM), user documents, and the requirements allocation matrix.

2.5. Maintenance Phase

The objective of this phase is to keep the system in production after delivery to the user community. This process is continued until system retirement or support termination. Figure 6 shows the activities performed in this phase. As shown in Figure 6, the input of this phase is from the Deployment phase.

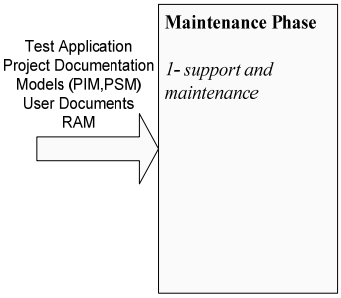

Figure 6. Main activities of the Maintenance phase

3. Evaluation Based on Existing MDA-based Methodologies

In this section, we evaluate the MDA-SDLC presented in the previous section through assessing existing MDA-based methodologies as to their degree of conformance with the proposed lifecycle. We show how phases and activities of existing methodologies correspond with MDA-SDLC phases and activities, and thereby show that the proposed lifecycle does indeed cover existing methodologies.

Six prominent MDA-based methodologies were selected for this purpose: MODA-TEL, MASTER, C^3, ODAC, DREAM, and DRIP-Catalyst. The MODA-TEL methodology is the MDA-based development methodology that is being developed and applied in the MODA-TEL project [7], mainly targeted at distributed applications. The MASTER methodology was developed as part of a European information project of the same name, and prescribes a MDD process together with a set of system family engineering methods to adapt the MDD process according to customer requirements [8]. C^3 is a component-based methodology enhanced with advanced MDD techniques [9], which adopts and elaborates on many principles from the Business Object Oriented Software Technology for Enterprise Reengineering (BOOSTER) approach. The ODAC methodology is based on RM-ODP (Reference-Model on Open Distributed Processing) [10, 11]. DREAM is a product line engineering methodology, which integrates key activities of PLE with model transformation features of MDA [12]. DRIP-Catalyst is a MDA-based methodology for the development of complex, fault-tolerant distributed families of software [13].

Table 1 shows the phases and activities of theses methodologies. Table 2 shows how each phase and activity of the proposed MDA-SDLC covers the phases and activities of the selected MDA-based methodologies.

4. Conclusions

We propose a MDA-based SDLC for developing software systems using the model-driven approach. The lifecycle is detailed enough to define a framework for MDD, and general enough to be instantiated through a paradigm-based method engineering process, thereby producing MDA-based methodologies tailored to fit specific project situations. The lifecycle builds upon generic software engineering activities as well as MDA practices and standards for model-driven development. We have shown that the proposed lifecycle does indeed cover prominent MDA-based methodologies.

This research can be furthered by identifying process patterns recurring in MDA-based methodologies. These can then be defined as method chunks and used for assembly-based method engineering of MDA-based methodologies, ideally in conjunction with (and as a complement to) the proposed MDA-SDLC. A strand of research is also being directed at defining a MDA-based extension framework that allows existing methodologies to be extended into MDA-based methodologies [14].

Acknowledgment

We wish to thank the Research Vice-Presidency of Sharif University of Technology for sponsoring this research.

References

[1]. I. Mukerji, J. Miller, MDA Guide Version 1.0.1, *OMG*, 2003.

[2]. R. Ramsin, R. F. Paige, "Process-Centred Review of Object-Oriented Software Development Methodologies", *ACM Computing Surveys* (to be published).

[3]. A. F. Harmsen, *Situational Method Engineering*, Moret Ernst & Young, 1997.

[4]. J. Ralyté, R. Deneckére, C. Rolland, "Towards a generic model for situational method engineering", *Proc. of CAiSE2003*, 2003, pp. 95-110.

[5]. S. W. Ambler, *Process Patterns: Building Large-Scale Systems Using Object Technology,* Cambridge University Press, 1998.

[6]. S. W. Ambler, "The agile system development lifecycle", *published on the web at: http://www.ambysoft.com/essays/agileLifecycle.html*, 2006.

[7]. A. Gavras, M. Belaunde, L. Ferreira Pires, J. P. Andrade Almeida, "Towards an MDA-based development methodology", *Proc. First European Workshop on Software Architecture (EWSA2004)*, Enschede, Netherlands, 2004, pp. 71-81.

[8]. X. Larrucea, A. B. G. Diez, J. X. Mansell, "Practical Model Driven Development process", *Proc. Second European Workshop on Model Driven Architecture (MDA)*, Canterbury, UK, 2004.

[9]. T. Hildenbrand, and A. Korthaus, "A Model-Driven Approach to Business Software Engineering", *Proc. 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004), Volume IV Information Systems, Technologies and Applications*, Orlando, Florida, USA, 2004, pp. 74-79.

[10]. M. Gervais, "ODAC: An Agent-Oriented Methodology Based on ODP", *Journal of Autonomous Agents and Multi-Agent Systems*, *7*(3), 2003, pp. 199-228.

[11]. M. Gervais, "Towards an MDA-Oriented Methodology", *Proc. 26th Annual International Computer Software and applications Conference (COMPSAC'02),* Oxford, England, 2002.

[12]. S. Kim, H. G. Min, J. S. Her, S. H. Chang, "DREAM: A practical product line engineering using model driven architecture", *Proc. ICITA '05,* Australia, 2005, pp. 70–75.

[13]. N. Guelfi, R. Razavi, A. Romanovsky, S. Vandenbergh, "DRIP Catalyst: an MDE/MDA Method for Fault-tolerant Distributed Software Families Development", *Proc. OOPSLA & GPCE 2004 workshop on best practices for Model Driven Development*, 2004.

[14]. F. Chitforoush, M. Yazdandoost, R. Ramsin, "Methodology support for the Model-Driven Architecture", *Proc. APSEC'07*, Nagoya, Japan, 2007, pp. 454-461.

Table 1. Phases and activities of the selected MDA Based methodologies

MODA-TEL		**MASTER**	**C3**	**ODAC**	**DREAM**	**DRIP-Catalyst**
Phase	Activity in phase					
project management phase	project organization, quality management	*capture user requirements* phase	*standardization* phase	*application deployment* phase	*domain analysis* phase	*problem to solution transition* phase
		PIM context definition phase	*software development* phase		*product line scoping* phase	*platform-Independent Architectural Design* phase
preliminary preparation phase	platform, transformation, modeling language, and traceability strategy identification	*PIM requirements specification* phase	*model design* phase	*design* phase	*framework modeling* phase	*Platform-independent detailed design* phase
					application requirements	*formal verification* phase
detailed preparation phase	modeling languages and transformations specification	*PIM analysis* phase	*code generation*		*application-specific design* phase	
		design phase			*framework instantiation* phase	*PIM to PSM transition* phase
infrastructure setup phase	tool support and metadata management					
		coding and integration phase	*application deployment* phase	*implementation* phase	*model integration* phase	*PSM to code* phase
execution phase	requirements analysis, modeling, verification, coding and testing, integration and deployment, operation and maintenance	*test* phase			*application detailed design*	*completion* phase
		deployment phase			*application implementation* phase	*deployment* phase

Table 2. Results of assessing existing MDA-based methodologies

MDA-SDLC		MODA-TEL	MASTER	C^3	ODAC	DREAM	DRIP Catalyst
Project Initiation Phase	*CIM definition*	×	PIM Context Definition	×	×	×	×
	Requirements specification	Requirements Analysis	Capture User Requirements & PIM Requirements Specification	×	×	Domain Analysis, Product line scoping	×
	Obtain funding and support	Project Management & Tool selection	×	×	×	×	×
	Define general plan		×	×	×	×	×
	Tool selection		×	×	×	×	×
	Initiate the team		×	×	×	×	×
PIM development phase	*Produce Analysis PIM*	Modeling	PIM Analysis	Standardization	Analysis	Requirements Modeling	Problem to Solution Transition
	Architectural Design		×	Software Development		Framework modeling	Platform-independent Architectural design
	Produce Design PIM		Design	Model Design	Design	Application specific Design, Framework instantiation, Model Integration	Platform-independent Detailed Design
	Generalization	×			×	Framework Modeling	×
	Verification/ Validation	Verification /Validation	×	×	×	×	Formal Verification
PSM & Code development phase	*Transform PIM to PSM*	Transformation	Design	Code Generation	Design	Application Detailed Design	PIM To PSM Transition
	Generate code	Code/Testing	Coding & Integration		Implementation	Application Implementation	PSM to Code Transition, Code Completion
	Testing		Testing	×	×	×	×
Deployment Phase	*Final testing of the system*			×	×	×	×
	Finalizing system and users documentation	×	Deployment	Application Deployment	×	×	Deployment
	Transition of the system to the user environment	Integration/ Deployment			×	×	
Maintenance phase	*support*	Operation/ Maintenance	×	×	×	×	×
	maintenance		×	×	×	×	×