MDA-Based Methodologies: An Analytical Survey

Mohsen Asadi and Raman Ramsin

Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran
mohsenasadi@mehr.sharif.edu, ramsin@sharif.edu

Abstract. Model-Driven Development (MDD) has become a familiar software engineering term in recent years, thanks to the profound influence of the Model Driven Architecture (MDA). Yet MDD, like MDA itself, defines a general framework, and as such is a generic approach rather than a concrete development methodology. Methodology support for MDA has been rather slow in coming, yet even though several MDA-based methodologies have emerged, they have not been objectively analyzed yet. The need remains for a critical appraisal of these methodologies, mainly aimed at identifying their achievements, and the shortcomings that should be addressed. We provide a review of several prominent MDA-based methodologies, and present a criteria-based evaluation which highlights their strengths and weaknesses. The results can be used for assessing, comparing, selecting, and adapting MDA-based methodologies.

Keywords: Model Driven Architecture, Software Development Methodology, Evaluation Criteria.

1 Introduction

The Model-Driven Architecture (MDA) proposed by the Object Management Group (OMG) 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. The primary goals of MDA are portability, interoperability, and reusability of software. To achieve these goals, MDA raises the level of abstraction and strives to automate the software generation process.

There are a number of important OMG standards at the core of MDA: The Unified Modeling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), and Common Warehouse Metamodel (CWM) [1]. These standards define the core infrastructure of the MDA, and have greatly contributed to modern systems modeling. The core standards of the MDA (UML, MOF, XMI, and CWM) form the basis for building coherent schemes for authoring, publishing, and managing models within a model-driven architecture.

MDA provides an approach for specifying systems in terms of models; 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

I. Schieferdecker and A. Hartman (Eds.): ECMDA-FA 2008, LNCS 5095, pp. 419–431, 2008.

the other hand, describes the system design in the form of a platform-dependent model. Through its multi-layered modeling approach, MDA raises the abstraction level of traditional platform-dependent design approaches.

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 as to 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. This fact forces organizations willing to adopt the MDA 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.

This research presents an analytical review and evaluation of a select set of existing MDA-based methodologies. The research has been performed in three main steps: information gathering and methodology selection, development of Evaluation Criteria (EC), and criteria-based evaluation of the selected MDA-based methodologies – with results and observations presented in tabular form. The results can be used by software developers to select the MDA-based methodology best suited to their needs, and by method engineers to create MDA-based methodologies through making use of the strengths identified and addressing the deficiencies observed.

The information gathering step involves studying relevant MDA literature and identifying prominent MDA-based methodologies. An initial set of evaluation criteria is then defined; this initial set is refined and completed to satisfy a predefined set of suitability Meta-Criteria (criteria to evaluate evaluation criteria). The last step involves performing the evaluation based on the set of criteria, and tabulating and analyzing the results.

The rest of the paper is organized as follows: Section 2 provides a review of existing MDA-based methodologies; we present our evaluation results in section 3, and provide an analysis of the results in Section 4; conclusions and areas for furthering this research are presented in section 5.

2 Review of MDA-Based Methodologies

In this section, we present a review on MDA-Based methodologies using the process-centered template introduced in [2], which accentuates the processes of the methodologies. The main factor influencing the selection of these particular methodologies was the availability of proper resources and documentation on their processes.

2.1 ODAC Methodology

ODAC is an MDA based methodology specifically targeted at distributed systems. It provides a set of concepts and structure rules to create systems. The "viewpoint" is the main concept used in this methodology. A viewpoint is a subdivision of the complex specification of the system [3], used for organizing the modeling activities. ODAC

considers five viewpoints: *enterprise*, *information*, *computational*, *engineering*, and *technology*. It uses these concepts to define development steps by identifying the correspondences between analysis, design, and implementation activities and the viewpoints. ODAC identifies three categories of specifications for each system: *behavioral*, *engineering*, and *operational* [4]. ODAC phases are as follows (Fig. 1):

- *Analysis*: produces the behavioral specifications (PIM) of the system.
- *Design*: establishes the engineering specifications (analogous to MDA's Platform Description Model–PDM) and uses it to produce the operational specifications (PSM) via projection of the PIM onto a target environment.
- *Implementation*: generates the execution code from the PSM.

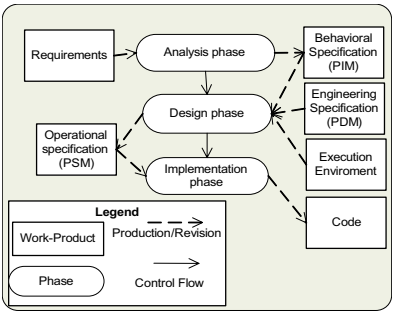

Fig. 1. The ODAC Process

2.2 MASTER Methodology

MASTER was developed as part of a European information project of the same name. The methodology includes an MDD process and a set of system family engineering methods to adapt the MDD process according to customer requirements [5]. The activities and roles of this methodology are defined based on the *Software Process Engineering Metamodel* (SPEM) [6]. MASTER phases are as follows (Figure 2):

- *Capture User Requirements*: covers requirements elicitation and documentation.
- *PIM Context Definition*: describes the domain scope of the software system to be developed. The output of this phase is a clear definition of the system, its goals, and its domain.
- *PIM Requirements Specification*: develops a clear and complete requirements model. The main activity of this phase includes specifying *capabilities* (use cases) and *enforcers* (nonfunctional requirements) of the system.
- *PIM Analysis*: models the internal view of the system regardless of the technological constraints.
- *Design*: models the detailed structure and behavior of the system.
- *Coding and Integration*: develops and verifies the execution code. The code can be generated from the platform-specific model by means of MDA tools.
- *Test*: verifies and validates the final system.
- *Deployment*: transitions the system to the user environment.

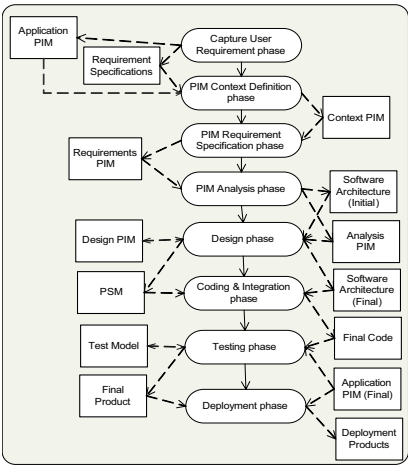

Fig. 2. The MASTER Process

2.3 C³ Methodology

The C³ methodology uses principles of Business Object Oriented Software Technology for Enterprise Reengineering (BOOSTER) to develop business applications [7]. The name C³ is derived from the three concepts of inter-organization *Collaboration*, *Concurrent* software engineering and *Component* development. Concurrent software engineering for both system architectural design and component design is realized through Model-Driven Development and XMI-based techniques.

The phases of this methodology are as follows (Figure 3):

- *Standardization*: downloads the required model elements needed to develop the target business software from the project repository.
- *Software Development*: defines the application's overall architecture.

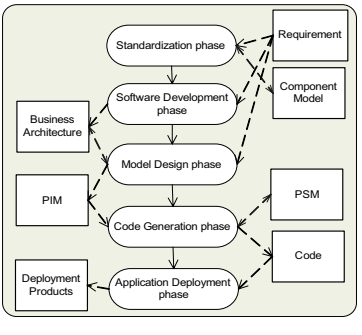

Fig. 3. The C³ Process

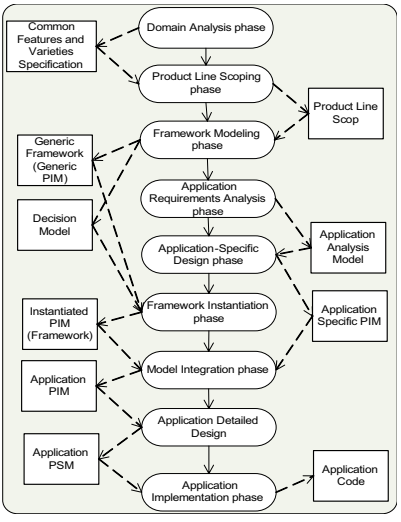

Fig. 4. The DREAM Process

- *Model Design*: refines the business application architecture. The PIM is the output of this phase.
- *Code Generation*: transforms the PIM to PSM and deployable components.
- *Application Deployment*: prepares the software for deployment into the operational environment based on the architectural framework.

2.4 DREAM Methodology

The DRamatically Effective Application development Methodology (DREAM) combines the key activities of Product Line Engineering (PLE) with the model transformation features of MDA [8]. DREAM phases are as follows (Fig. 4):

- *Domain Analysis*: captures the features of several organizations in the same domain, and analyzes the Commonality and Variability (C&V). The output is the specification of common features and differences between organizations.
- *Product Line Scoping*: determines the scope of the target product line.
- *Framework Modeling*: realizes the C&V in a framework, presented as a PIM. The framework defines the general architecture for the desired members of the product line, together with the relationships and constraints.
- *Application Requirements Analysis*: analyzes the application requirements and identifies the features related to the application at hand. The output of this phase is the application analysis model.
- *Application-Specific Design*: realizes the application analysis model as a platform-independent design model. The output is the application-specific PIM.
- *Framework Instantiation*: instantiates the framework for the specific application by setting the variants accordingly. The output of this phase is the instantiated framework PIM.
- *Model Integration*: integrates the specific application PIM and the instantiated framework PIM into one model.

- *Application Detailed Design*: refines the integrated model by considering platform-specific issues, thereby producing the PSM.
- *Application Implementation*: generates the execution code and its related implementation complements – such as the database – from the PSM.

2.5 MODA-TEL Methodology

The MODA-TEL methodology is mainly targeted at distributed applications [9]. The activities and roles of this methodology are defined based on SPEM [6].

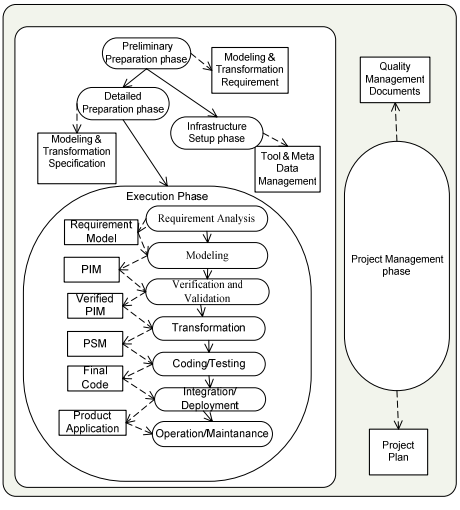

Fig. 5. The MODA-TEL Process

As shown in Fig. 5, the MODA-TEL process consists of five phases:

- *Project Management*: manages and monitors the project.
- *Preliminary Preparation*: identifies modeling and transformation requirements.
- *Detailed Preparation*: determines modeling and transformation specifications.
- *Infrastructure Setup*: provides the tool support and metadata management facilities to be used in the execution phase.
- *Execution*: aims at developing the software artifacts and executable code. The activities include: Requirement Analysis, Modeling (producing the PIM), Verification/Validation, Transformation (PIM to PSM), Coding/Testing, Integration/Deployment, and Operation/Maintenance.

2.6 DRIP-Catalyst Methodology

DRIP-Catalyst is an MDA-based methodology for developing complex, fault-tolerant distributed families of software [10]. DRIP stands for Dependable Remote Interacting Processes. The methodology makes use of the notion of "Atomic Action" as a recovery technique that permits programmers to apply backward and forward error

recovery. A Coordinated Atomic Action (CAA) consists of distributed transactions and an atomic action. The DRIP framework embodies the CAAs in terms of a set of java classes. It builds on the notion of Dependable Multiparty Interaction (DMI). DRIP Catalyst includes a process, a UML profile and a set of transformations, all of which have been integrated into a tool. DRIP-Catalyst phases are as follows (Fig. 6):

- *Problem to Solution Transition*: maps the requirements to the solution through sketching nested CAA diagrams.
- *Platform-Independent Architectural Design*: categorizes the CAAs generated in the previous phase in a coherent package of UML class diagrams.
- *Platform-Independent Detailed Design*: details the modeling elements related to each CAA identified in the previous phase, using UML activity diagrams.
- *Formal Verification*: automatically checks dependability properties using formal methods, and verifies that the models satisfy the requirements, thus producing a verified Platform-Independent Detailed Design Model (PIM2DM).
- *PIM to PSM Transition*: maps the PIM2DM to a platform-specific model through transformation provided by MDA tools, producing the PSM.
- *PSM to Code*: maps the PSM to execution code.
- *Completion*: produces code that can be compiled.
- *Deployment*: defines a configuration set that realizes the deployment of the application, through producing deployment guides and configuration files.

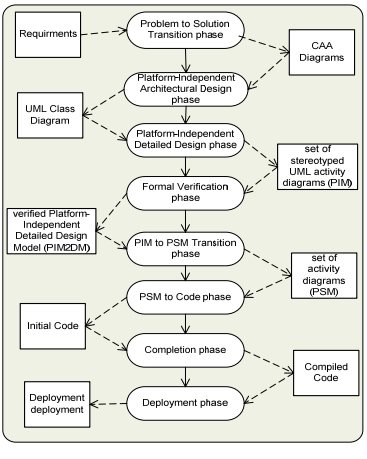

Fig. 6. The DRIP-Catalyst Process

3 Evaluation of MDA-Based Methodologies

We have evaluated the MDA-based methodologies reviewed in the previous section using a method similar to the Feature Analysis approach [11]. The Feature Analysis approach was developed in 1996 under a collaborative project between academia and industry. The outcome of this project was a method to evaluate software engineering

methods and tools. Feature Analysis provides two ways for evaluating any product in terms of results: a simple form and a scale form. In its simple form, the approach presents a list of "yes/no" responses against the existence of some feature in a product. In the scale form, instead of "yes/no" responses, a number between -1 to 5 is used which represents the degree of conformance of the product to a feature.

Since the selected MDA-based methodologies are evaluated with a set of Evaluation Criteria (EC), the development of the criterion set is an important feature of our research. The collected criteria are of two types: (a) Scale type, where a scale represents the degree of presence of a criterion in the methodology (we use scales with three levels for each such criterion); and (b) Narrative type, where the degree of the implementation of the criterion in the methodology is described in narrative form.

The criterion set used for evaluation was developed through gradual refinement: an initial set of general criteria – addressing software development processes and MDD-related issues – was compiled through studying relevant resources, such as official MDA specifications and survey/analysis reports on software development methodologies [2]; the set was then refined, using a set of meta-criteria (criteria to evaluate the EC set) to guide the refinement process towards a reasonably complete and precise set of criteria. The following meta-criteria were defined for this purpose:

(I) *Existence of tool-related criteria*: used to ensure that the EC set provides tool evaluation, as most MDA practices are enacted through specialized tools. This meta-criterion ensures the existence of criteria that measure how much of a task is governed by tools and how much by the methodology itself; that is, whether the methodology participates in such activities or leaves them to tools.

(II) *Existence of MDA-related criteria*: used to evaluate the completeness of the EC set as pertinent to MDA aspects. This meta-criterion ensures that the EC set covers the MDA aspects of methodologies. MDA-related criteria are applied to the methodologies only in an MDA-related context.

(III) *Existence of general criteria*: used to evaluate the completeness of the EC set from general aspects. This meta-criterion ensures that the EC set covers the general aspects of the methodologies; general criteria can be applied to all methodology types: plan-driven, agile and MDA-based methodologies alike.

The refinement process proceeded by categorizing the initial set of criteria into *tool-related*, *MDA-related*, and *general* criteria (according to the above meta-criteria). For each category, relevant resources were then searched iteratively for new criteria and ideas for refining the existing ones. For instance, in striving to complete the criteria belonging to the general category, general software engineering resources and existing documentation on methodologies (such as plan-driven, agile and component-based) were consulted; as an example, since most methodologies (especially agile methods) include activities for customizing and adapting their processes, adaptability was added as a criterion in order to cover this need in MDA-based methodologies; Reusability was also added, based on the observation that MDA and most component-based methodologies consider it as essential. Tables 1, 2 and 3 show the resulting evaluation criteria; the three tables correspond to meta-criteria I, II and III respectively. We have strived to produce a useful, relevant, and meaningful criterion set, while keeping it small and practical. We have therefore focused on addressing features that are particularly important and significant in MDA and MDD.

As a basic requirement, evaluation criteria targeted at software development processes are expected to satisfy certain validity meta-criteria; one such set has been defined in [12]. Our evaluation criteria satisfy the four validity meta-criteria of [12], in that they are: 1) *general* enough to be applicable to all MDA-based methodologies; 2) *precise* enough to help discern the similarities and differences among MDA-based methodologies; 3) *comprehensive* enough to cover all significant features of MDA-based methodologies; and 4) *balanced*, i.e. adequate attention has been given to all three major types of features in a methodology: *technical*, *managerial* and *usage* [12].

Table 1. Tool-related evaluation criteria (satisfying meta-criterion I)

Criterion Name	Criterion Type	Description of Levels
PIM to PSM Transformation	Narrative	Involved: The Methodology explicitly participates in the activity and provides precise techniques/guidelines. Devolved: The activity is devolved to the tools and the methodology does not prescribe the steps that should be performed by the tools.
PSM to Code Transformation	Narrative	
Metadata Management	Narrative	
Automatic Test	Narrative	
Traceability between Models	Narrative	

Table 2. MDA-related evaluation criteria (satisfying meta-criterion II)

Criterion Name	Criterion Type	Description of Levels
Tool Selection/Implementation	Scale Form	A: The methodology does not provide a specific tool and there are no explicit guidelines as to how to select an appropriate alternative tool. B: The methodology does not provide a complete toolset, or only general guidelines are provided for selecting alternative tools. C: The methodology provides a complete toolset, or provides precise guidelines for selecting appropriate alternative tools.
CIM Creation	Scale Form	A: Production of the model is not addressed by the methodology. B: The methodology provides general guidelines for creating the model; creation steps are not determined precisely. C: The methodology explicitly describes steps and techniques for creating the model.
PIM Creation	Scale Form	
PSM Creation	Scale Form	
Verification/ Validation	Scale Form	A: The activity is not defined and is devolved to the developers. B: The activity is defined by the methodology, but not in detail. C: The methodology provides explicit and detailed guidelines and techniques for performing the activity.
Extension of Rules	Scale Form	
Round-trip Engineering	Scale Form	
Source Model and Target Model Synchronization	Scale Form	
Use of UML Profiles	Narrative	

Tables 4, 5, and 6 show the results of applying the evaluation criteria to the selected set of MDA-based methodologies. It should be noted that the interpretation of the results is largely dependent on the usage context. The evaluation results can be used for selecting a suitable process from the set of surveyed ones based on a set of predefined requirements, or for identifying shortcomings in these processes in order to improve them. The evaluation framework (criteria) and the results can also be used in a Method Engineering (ME) context; i.e. for guiding the adaptation, extension, meta-modeling/instantiation, and decomposition/assembly of MDA-based processes.

4 Analysis of the Results

The following subsections contain analyses of the evaluation results shown in tables 4, 5, and 6. Of the methodologies reviewed herein, MODA-TEL and MASTER are the methodologies that satisfy most of the criteria.

Table 3. General evaluation criteria (satisfying meta-criterion III)

Criterion Name			Criterion Type	Description of Levels
Coverage	Generic Life Cycle	Requirements Engineering	Scale Form	A: The methodology does not provide coverage for the phase. B: The methodology provides general guidelines for the phase. C: The methodology provides detailed directives for the phase.
		Analysis	Scale Form	
		Design	Scale Form	
		Implementation	Scale Form	
		Test	Scale Form	
		Deployment	Scale Form	
		Maintenance	Scale Form	
	Umbrella Activities	Project Management	Scale Form	A: The methodology does not provide coverage for the activity. B: The methodology provides general guidelines for the activity. C: The methodology provides detailed directives for the activity.
		Quality Assurance	Scale Form	
		Risk Management	Scale Form	
Problem Domain Analysis			Scale Form	A: Problem Domain Analysis has not been addressed. B: Problem Domain Analysis is implicit and confined to requirements engineering. C: Problem Domain Analysis is explicitly addressed by the methodology, and traceability is maintained.
Reusability			Scale Form	A: The task is devolved to the developers; the methodology does not prescribe techniques/guidelines. B: The methodology explicitly prescribes techniques to create potentially reusable artifacts. C: In addition to B, the methodology prescribes techniques to record syntactic/semantic features of reusable aspects for future reuse.
Adaptability			Scale Form	A: No techniques are prescribed for adapting the methodology. B: The methodology provides extensible notations. C: In addition to B, the methodology prescribes explicit techniques for configuring the process and/or modeling language.
Completeness of Definition			Scale Form	A: Some phases of the methodology are not completely specified. B: All phases are completely specified (in breadth) but details are lacking in some phases. C: All phases are completely specified at an adequate level of detail.
Methodology Type			Scale Form	Extended: The methodology is the result of extending an existing methodology to support MDA-based development. MDA-based (Genuine): The methodology has been created from scratch aimed at supporting MDA-based development.
Application Scope			Narrative	

4.1 Tool-Related Evaluation Results

The results seem to show that most of the methodologies examined do not offer any guidelines as to how MDA tools should be used in coherence with the methodology, thus leaving all tool-related issues to the tools themselves. The only counterexamples are MODA-TEL and MASTER, and even these do not provide full coverage.

4.2 MDA-Related Evaluation Results

Since tools have a key role in MDD, MDA-based methodologies are expected to incorporate activities aimed at selecting or implementing appropriate tools. While DRIP-Catalyst and MASTER incorporate suitable tools themselves, MODA-TEL provides guidelines for selecting the tool from existing commercial and open source MDA toolsets. DREAM, C^3 and ODAC are at the other end of the spectrum: they do not even offer any guidelines as to how an appropriate alternative tool can be selected.

All of the MDA-based methodologies reviewed incorporate activities for creating the PIM and PSM; creation of the CIM, however, is only addressed by MASTER and C^3. Due to the model-centric nature of MDD, syntactic and semantic accuracy of the models is essential, as is their traceability to requirements; however, most of the processes reviewed do not provide adequate support for model verification/validation.

All the methodologies reviewed (except for MASTER) are weak in providing other important MDA features; i.e., support for extension of rules, round-trip engineering, and source-model and target-model synchronization.

4.3 General Evaluation Results

Most of the methodologies reviewed cover the analysis, design, and implementation phases of the generic software development life cycle, either by prescribing specialized techniques, or through making use of existing object oriented techniques; however, the requirements engineering, test, deployment, and maintenance phases are not adequately supported in most of them. For instance, only MODA-TEL supports maintenance, whereas MDA-based maintenance requires special techniques that cannot be simply borrowed from existing methodologies. Another area where MDA-based processes need improvement is support for umbrella activities; of the processes reviewed, only MODA-TEL and MASTER provide support for project management and quality assurance, while risk management is not supported by any methodology.

Table 4. Results of applying the Tool-related evaluation criteria

Criterion \ Methodology	MODA-TEL	MASTER	C^3	ODAC	DREAM	DRIP-Catalyst
PIM to PSM Transformation	Involved	Involved	Devolved	Devolved	Devolved	Devolved
PSM to Code Transformation	Involved	Involved	Devolved	Devolved	Devolved	Devolved
Metadata Management	Involved	Involved	Involved	Devolved	Devolved	Devolved
Automatic Test	Devolved	Involved	Devolved	Devolved	Devolved	Devolved
Traceability between Models	Involved	Devolved	Devolved	Devolved	Devolved	Devolved

Table 5. Results of applying the MDA-related criteria

Criterion \ Methodology	MODA-TEL	MASTER	C^3	ODAC	DREAM	DRIP-Catalyst
Tool Selection/ Implementation	B	C	A	A	A	C
CIM Creation	A	B	B	A	A	A
PIM Creation	B	C	B	C	B	C
PSM Creation	B	C	B	B	B	B
Verification/ Validation	B	A	A	A	A	B
Extension of Rules	C	B	A	B	B	A
Round-trip Engineering	B	A	A	A	A	A
Source Model and Target Model Synchronization	B	B	A	A	A	A
Use of UML Profiles	Used for Requirements Representation	Used for Annotating PIM with Management Information	Not Used	Used for Describing Development Steps.	Used for Defining Well-Structured Models	Used for Defining Fault-Tolerant Transactions

Table 6. Results of applying the General criteria

Criterion \ Methodology			MODA-TEL	MASTER	C^3	ODAC	DREAM	DRIP-Catalyst
Coverage	Generic Life Cycle	Requirements Engineering	B	C	B	A	B	A
		Analysis	B	C	A	C	B	C
		Design	B	C	B	C	B	C
		Implementation	B	B	B	B	B	B
		Test	B	C	A	A	A	A
		Deployment	B	B	B	A	A	B
		Maintenance	B	A	A	A	A	A
	Umbrella Activities	Project Management	B	C	A	A	A	A
		Quality Assurance	B	B	A	A	A	A
		Risk Management	A	A	A	A	A	A
Problem Domain Analysis			A	B	B	A	B	A
Reusability			B	B	B	A	B	A
Adaptability			B	B	A	A	A	A
Completeness of Definition			B	C	A	B	B	A
Methodology Type			MDA-Based	MDA-Based	Extended	Extended	Extended	Extended
Application Scope			Distributed Applications	Information Systems	Business Software	Agent-Oriented Systems	Product Line Engineering	Distributed Fault-Tolerant Applications

Most of the MDA-based methodologies reviewed provide techniques for creating and applying reusable artifacts. Support for reusability, however, is not comprehensive enough: Methodologies do not prescribe techniques for recording the syntactic- and semantic features of reusable artifacts in order to facilitate future reuse.

Developers prefer methodologies which lend themselves to customization and adaptation; but of the methodologies reviewed, only MODA-TEL and MASTER provide adaptability (in the form of extensible notations). Furthermore, methodologies need to be properly defined in order to be usable; however, some of the methodologies reviewed herein (e.g., C^3) suffer from cursory definitions of activities.

5 Conclusions and Future Work

MDA cannot be useful without software development methodology support and the tools that implement its main concepts and standards. We have surveyed several prominent MDA-based methodologies and have evaluated them using a predefined set of evaluation criteria. According to the evaluations results, we can conclude that:

- The MDA-based methodologies studied herein are not mature enough, especially as pertaining to providing support for standard software engineering activities.
- Definitions of methodologies are not complete.
- Umbrella activities are not adequately addressed in most of these methodologies.
- Most of the methodologies do not participate in the activities that are supported by tools, and do not even provide guidelines for tool usage.
- PIM and PSM production is supported by the majority of these methodologies; the CIM, however, is mostly neglected.
- Most of the methodologies use conventional OOA and OOD techniques to produce PIMs.

We aim to further this research by identifying a set of process patterns showing recurring activities in different MDA-based methodologies, thereby producing a generic and instantiable process for such methodologies.

References

1. Mukerji, I., Miller, J.: MDA Guide Version 1.0.1. OMG (2003)
2. Ramsin, R., Paige, R.F.: Process-Centered Review of Object-Oriented Software Development Methodologies. ACM Computing Surveys~40(1), 1--89 (2008)
3. Gervais, M.: Towards an MDA-Oriented Methodology. In: 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), pp. 265--270. IEEE Press, Oxford (2002)
4. Gervais, M.: ODAC: An Agent-Oriented Methodology Based on ODP. Journal of Autonomous Agents and Multi-Agent Systems~7(3), 199--228 (2003)
5. Larrucea, X., Diez, A.B.G., Mansell, J.X.: Practical Model Driven Development process. In: Second European Workshop on Model Driven Architecture (MDA), UK (2004)
6. Object Management Group: Software Process Engineering Metamodel v1.0 (SPEM). OMG (2002)

7. Hildenbrand, T., Korthaus, A.: A Model-Driven Approach to Business Software Engineering. In: 8th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida. Information Systems, Technologies and Applications, vol.~IV, pp. 74--79 (2004)
8. Kim, S., Min, H.G., Her, J.S., Chang, S.H.: DREAM: A practical product line engineering using model driven architecture. In: ICITA 2005, Australia, pp. 70--75 (2005)
9. Gavras, A., Belaunde, M., Ferreira Pires, L., Andrade Almeida, J.P.: Towards an MDA-based development methodology. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol.~3047, pp. 71--81. Springer, Heidelberg (2004)
10. Guelfi, N., Razavi, R., Romanovsky, A., Vandenbergh, S.: DRIP Catalyst: an MDE/MDA Method for Fault-tolerant Distributed Software Families Development. In: OOPSLA \& GPCE workshop on best practices for Model Driven Development, Portland (2004)
11. Kitchenham, B., Linkman, S., Law, D.: DESMET: a methodology for evaluating software engineering methods and tools. Computing and Contrological Engineering Journal~8, 120--126 (1997)
12. Karam, G.M., Casselman, R.S.: A cataloging framework for software development methods. IEEE Computer~26(2), 34--45 (1993)