Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2010
Volume 43

On Foundation for Certification of Model Driven Architecture (MDA) Tools: Defining a Specification

Antons Cernickins, *Riga Technical University*, Oksana Nikiforova, *Riga Technical University*

***Abstract* - A modern approach to software development includes a wide variety of processes, which are used to regulate and manage the entire development life cycle. One of the main aspects to be considered during the preparation stage of software development is the selection of the right tools. Nowadays when software systems become more and more complex, an inappropriate selection of the tools may cause project not to succeed at all. That is why the main challenge of the industry is still aimed on the simplification, optimization, and automation of software development process. One of the ways to deal with an increasing complexity, as well as not to dig into the details of the source code, is to use modeling. A model-driven approach, mainly represented by OMG's Model Driven Architecture as the most advanced and popular one, purports to be the next evolutional milestone for the whole software industry. In fact, most of the tools currently available on the market are claimed as "MDA complaint," though have never been checked or analyzed for that compliance. Therefore, there should be a certification program defined to figure out the main features of each tool in accordance with OMG standards. The original article proposes a foundation for certification of MDA tools. In particular, this includes the specification of the most common features and options defined to clarify the accordance level of each tool from various perspectives.**

Keywords: **CASE tools, model driven architecture (MDA), model driven development (MDD), software development life cycle**

I. Introduction

Non-technical aspects aside, selecting the appropriate methodology is only a half step to succeed in software development. The other half of success relies on the appropriate tools. By no means, this is true for almost every field nowadays. In case of software development, the problem is how to make the right choice, i.e. how to select the appropriate tools that will automate all of the routines, which software developers do nowadays.

The original paper contains a research on Model Driven Architecture, reviewing the model-driven approach to software engineering within the variety of the CASE tools, which are proposed as supporting for MDA activities. The goal of the paper is to propose a specification for classification of MDA tools – the first milestone in the development of the certification program. This paper provides a summary of research [1].

The paper is organized as follows. Section II provides a brief overview of software development life cycle. This includes the ISO/IEC 12207 standard, the variety of software development methods, techniques, and approaches, as well as the most current trend – modeling. A scope for specification of MDA tools is defined in Section III, including a short review of the requirements, selected to classify the variety of MDA tools. Furthermore, the research concludes the importance of the specification and proposes a scope of problems to be reviewed in further research.

II. Software Development Life Cycle

In brief, a software development life cycle is considered as a foundation for any method (methodology) used on the design and development of a software product [2]. Besides, it is a continuous process, which starts with a decision to develop a software product and ends at a moment, when the developed software product is disposed [3].

In order to regulate the structure of the software development life cycle, as well as the processes and activities within it, the international standard ISO/IEC 12207 has been proposed [4]. This standard establishes a common framework for software life cycle processes, with well-defined terminology that can be referenced by the software industry.

According to ISO/IEC 12207, the structure of software development life cycle includes the following two groups of processes: System Life Cycle Processes and Software Specific Processes [4]. The former group of processes includes:

1. Agreement Processes;
2. Organizational Project-Enabling Processes;
3. Project Processes;
4. Technical Processes.

The latter group of processes is represented by:

1. Software Implementation Processes;
2. Software Support Processes;
3. Software Reuse Processes.

In turn, each group of processes is divided into subgroups, where every process features the performance of process-specific activities and solution methods [3].

However, this standard is not designed towards any specific life cycle model [3]. Therefore, the main emphasis of the standard is to provide regulations for any kind of software development models, methodologies, and approaches.

A. Background

The process of software development as such continues to evolve [5]. Since the actual beginning of the software industry dozens of various approaches, techniques, and models have been proposed. Some of them had a major impact on how the software is built nowadays.

The following models formed a basis for software development [3]:

- Waterfall model (1970 – 1985);
- Spiral model (1986 – 1990).

The waterfall model represents a sequential way for software development, where progress is seen as flowing like a waterfall through the stages of software development life cycle [3], [6]. To proceed from one stage to another the preceding stage should be completed and perfected. In order to provide an additional flexibility, various modifications of the waterfall model have been defined.

In turn, the spiral model approach combines the features of the waterfall and prototyping models in an effort to emphasize the initial stages of software development—requirements specification and design [3], [7]. In spiral model, the process of software development is iterative, meaning that the actual is done without affecting the former one. The spiral model approach is typically intended for large projects.

The process of software development may also involve the other activities, not necessarily the ones that are closely associated with a development process itself (e.g., design or implementation stage) – risk management, human resource management, etc. In fact, this facilitated the establishment of various integrated approaches, such as Rational Unified Process (RUP) and Microsoft Solutions Framework (MSF) [8], [9].

RUP provides a disciplined approach to assigning tasks and responsibilities within a development organization [8]. The primary goal of RUP is to provide the development of high-quality software and meet clients' expectations within a predictable schedule and budget. In brief, RUP consists of four stages (phases), which may include several iterations, and nine disciplines (six engineering and three supporting disciplines). The purpose of initial iterations is to perform the requirements analysis and planning disciplines, while further iterations mainly are concerned with the actual development of the software product.

Microsoft Solutions Framework (MSF) is a deliberate and disciplined approach based on a defined set of principles, concepts, guidelines, and proven practices from Microsoft Corporation [9]. MSF does not force the developers to use a specific software development model (e.g., a waterfall model), providing a choice instead. The MSF Process Model – this model actually defines the software development life cycle – combines concepts from both waterfall and spiral models. As a result, the combination of both incorporates the benefits of milestone-based planning from the waterfall model with the incrementally iterating project deliverables from the spiral model.

However, when the primary goal of the development team is rapid software development and delivery, the Agile methods may be used [10]. In general, Agile methods promote a disciplined management process, where frequent inspection and adaptation of the final product is performed; tasks are divided into subtasks (increments), no direct long-term planning is performed. The whole development process remains iterative, through.

The list of the well-known agile methods includes XP (Extreme programming, considers responsiveness to frequent changes in client's requirements), Scrum (considers sets of practices and predefined roles), DSDM (Dynamic Systems Development Method, which considers the continuous user involvement during the whole development process), and many others.

B. Trend Towards Modeling

In 1970s and 1980s, most of the software systems were developed using structured methodology [3]. This methodology is based on a simple graphical approach: the actual model of the software system is described using schemes and diagrams in a formal way. Structured analysis also provides the ability for client to participate in the development process (in informal way).

However, the reality was challenging enough: it was hard to implement the structured methodology into the development process due to lack of tools [3]. Moreover, the structured methodology itself does not provide any flexibility in manual software development either. Therefore, a new division of tools was founded to satisfy the needs of software developers – Computer-Aided Software Engineering (CASE) tools [3].

The development of a new software product requires a variety of tasks to be organized and completed correctly [11]. CASE tools were developed exactly for these purposes – to automate these processes, to ease the coordination between them, as well as to provide a more efficient way to develop software.

Furthermore, the use of CASE tools increases the speed of the development [11]. Developers are always facing the problem of having business needs change prior to the system could be finished being developed – the actual affect of this problem was minimized. Staying on the leading edge in a highly competitive market can make the difference between success and failure. More specifically CASE tools [11]:

- ensure consistency, completeness and conformance to standards;
- encourage an interactive, workstation environment;
- speed up the development process;
- allow precision to be replicated;
- reduce costs, especially in maintenance;
- increase productivity;
- make structured techniques more practical.

C. Model Driven Architecture

As a manufacturer thinks of managing assets, expenses, and leverage, the same way of thinking should be considered by software industry [5]. While software becomes more sophisticated and complex, developers should still be able to maintain the quality level of their software products, providing reliability, and meeting client's expectations [12]. In turn, it is easy to notice the routines that are being done over and over again. Therefore, the consideration of modeling and model-based approaches may capitalize these routines (i.e. actually offering the optimization of the workflow).

While dozens of model and code generating software systems have been offered to solve the problems regarding software productivity and quality, the problem still remained unsolved [5]. As CASE tools developed up that time were oversold on their "complete code-generation capabilities" [13], similar arguments are now exposed to Object

Management Group (OMG) Model Driven Architecture (MDA).

However, it is important to point out that MDA is not yet another technical approach, proposed as a new form of CASE for modeling and the generation of the source code [14]. In fact, MDA carries forward some of the ideas of CASE, as well as provides a solution for some problems related to the usage and application of the CASE tools (e.g. the promulgation of XMI standard to deal with data exchange problems) [15].

Organizations can benefit from the application of the model-driven approach to the existing activities of the software development life cycle, such as requirements gathering, business analysis, process modeling, systems design, service definition, integration, solutions design, source code generation, automatic transformations, etc. [5]. Moreover, considering MDA as a holistic approach would take the entire development to the next level.

Whereas the reuse on the application level may look complex, the following set of activities by MDA provides a far more convenient view on this problem [16]:

1. Choose model that corresponds with a problem domain;
2. Subset the model as necessary;
3. Choose models in accordance with the implementation technology platform;
4. Define the interconnection between models;
5. Generate the software system.

When it becomes necessary to modify the application, changes are being introduced to the application model only (1). Furthermore, when it becomes necessary to retarget an application to a different implementation environment, models for the new environment are being selected (3), and software is being regenerated (5) [16]. Thereby, there is no need to modify the application models, costs are lower, productivity is higher, and maintenance is cheaper. Thus, each model is built in the way it can be subsequently reused.

Notably, MDA consists of four layers, which are considered as milestones in software development [17] (Fig. 1):

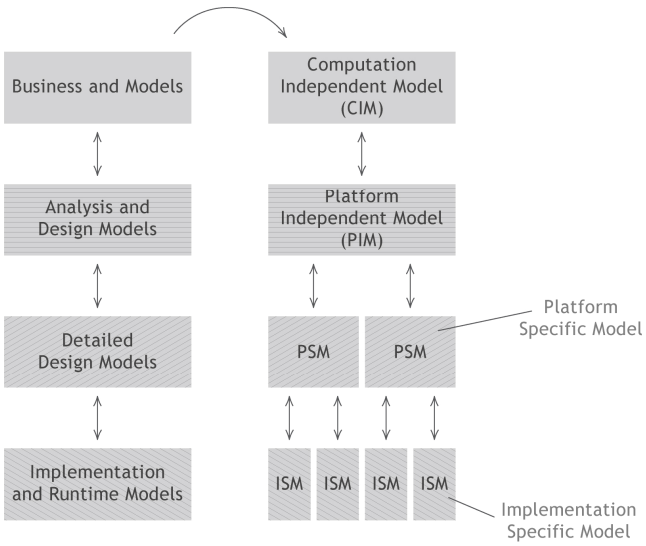

Fig. 1. The layers and transformations of MDA

Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), and Implementation Specific Model (ISM).

There are four principles that underlie the approach [17]:

1. Models expressed in a well-defined notation are the milestones of system representation;
2. System development is performed within a set of models and model transformations;
3. Models and model transformations are described in a formal way with the help of meta-models;
4. The broad adoption of MDA approach requires industry standards and openness.

To conclude, the fundamental idea of MDA approach is the separation of concerns—ability to separate the application architecture into several distinct models [18]. Then each model is evaluated and marked up, the source code is being generated. Whether the tool was released prior to or following the MDA specification, it should support this division and not just generate code from a class diagram.

III. Defining a Scope for Specification of MDA Tools

Modeling is not a new discipline in software development [19]. Hence, the same statement is true for any model-driven approach – it is "a different portion of the same image." That is why the essential idea of model-driven approaches should be considered as evolutionary.

An approach to classification of CASE tools, which are positioned as "MDA compliant," has already been proposed in [18]. Rather than trying to define a scope for specification of MDA tools from the blank, CASE tools should be investigated first.

First of all, the selection of the appropriate CASE tools is influenced by software product itself: goals, user requirements, constraints, skills, etc. – all these capabilities make sense [3]. Secondly, the analysis methodology of the problem and its implementation should be considered as well. Thirdly, the whole process of software development should be driven by CASE tools, meaning that software engineers should be able to combine tools, where each tool serves different purposes (i.e. a tool chain is formed).

To sum up, CASE tools should be selected in accordance with the following requirements [3]:

1. Support for software development life cycle, modeling, multiple platforms, and integration capabilities;
2. Project management and integrity control;
3. Platform and database independence;
4. Team support;
5. Client/server application development;
6. Openness and import/export capabilities;
7. Vendor's brand, loyalty, total cost of ownership, expenses;
8. Usability, easiness of implementation;
9. Quality of documentation;
10. Compliance with standards and notations.

The estimation of software quality is another consideration of this scope. In order to determine, which requirements should be used for this type of estimation, the international

standard ISO/IEC 9126 has been used. The objective of this standard is to deal with some of the well-known human biases that may affect the delivery and perception of a software product [3]. The standard is divided into four parts, with quality model of particular interest. ISO/IEC 9126 quality model includes the following requirements [3]:

1. Functionality – an existent set of functions and their specified properties;
2. Reliability – capability to maintain the appropriate level of performance under certain conditions for a certain period of time;
3. Usability – usage efforts and individual assessments of such usage;
4. Efficiency – the amount of resources needed to maintain the appropriate level of performance under certain conditions;
5. Maintainability – efforts needed to make specified modifications;
6. Portability – ability of software to be transferred to another environment.

As a fundamental part of MDA, modeling incorporates the notion of creating various models at different levels of abstraction, linking them together and forming an implementation [18]. Some of these models are platform specific, while the others are independent upon the software platforms. In turn, each model represents a combination of text and multiple complementary and interrelated diagrams [18]. That is why another perspective that can be used in this scope defines the role of modeling. In fact, the role of modeling is defined using Modeling Maturity Levels (MMLs), which are the following [18], [20]:

0. No specification – the specification of software is kept in the mind of a developer;
1. Textual specification – the specification of software usually is written down in one or more documents;
2. Text with models – this level enhances textual specification with several models to show the main components and objects of the system;
3. Models with text – the specification is written down in one or more models;
4. Precise models – shares the idea of specification to be written in one or more models;
5. Models only – models are detailed enough to allow complete code generation.

Whether a tool is commercial or open source, the licensing options is very important to many developers [18]. The typical benefits of open source software are affordability and availability of a source code. On the other hand, if the commercial software is purchased, the customer may also get such features as additional support, access to software upgrades, etc. Currently, the most known licensing models are commercial, open source, and commercial open source [18].

The most important group of requirements proposed in this scope is the software development life cycle of Model Driven Architecture (analyzed in [18]).

Table 1 provides an overview of the specification of MDA tools. This includes 7 categories, specified in a hierarchical way (hierarchical view is flattened in the table): categories are divided into subcategories, subcategories are divided into groups, and groups are divided into single entries, accordingly. Whereas some of the categories are detailed enough (i.e., have groups and entries), the others do not need to be detailed so precisely, meaning that subcategories should be considered as entries. A more detailed look on the specification itself is provided in [1].

TABLE 1
SPECIFICATION OF MDA TOOLS

Category	Subcategory	Group	Entry
Accordance with MDA-oriented life cycle	Knowledge formalization (CIM)	Generalization of requirements	
		Abstraction of knowledge	
		System model	
		Verification of user requirements	
	System model refinement (PIM)	Modeling language	
		Definition of infrastructure and constraints	
		Profile specification	
		Model verification (formalized knowledge)	
	Mapping (PIM-to-PSM)	Specification of mapping functions	
		Marking	
		Mapping verification	
	System model implementation (PSM)	Model verification (system model)	
	Transformation support	Refactoring capabilities	
		Model-to-Model transformation support	
		Model-to-Code transformation support	
		Customized transformation support	

TABLE 1 (CONTINUED)

Category	Subcategory	Group	Entry
Functional capabilities	Environment	Life cycle	
		Hardware requirements	
		Software requirements	
		Plug-in, add-in, extension support	
		Team work capabilities	
	Modeling	Role of modeling	
		Diagram creation	UML diagrams
			Tool-specific diagrams
		Data modeling	
		Process modeling	
		Model management	Import/export
			Meta-modeling
		Graphical analysis capability	
		Simulation capability	
		Prototype development	
		User interface generation capabilities	
		Traceability	
		Project-specific syntax and semantic management	
	Implementation	Syntax-driven editing	
		Maintainability analysis	
		Source code compilation	
		Source code analysis	
		Debugging	
	Testing	Testing description	
		Automation of operator’s actions	
		Automation of test cases	
		Regressive testing	
		Automation of result analysis	
		Testing coverage analysis	
		Performance analysis	
		Analysis of exceptional cases	
		Support for dynamic environments	
	Documenting	Textual and graphical mode support	
		Form support	
		Capabilities of publishing systems	
		Hypertext function support	
		Support for documenting standards	
		Automation capabilities	
	Configuration management	Authorization rights management	
		Tracking of changes	
		Version management	
		Management options	
		Version and modification generating	
		Archiving capabilities	
	Project management	Resource management	
		Evaluation capabilities	
		Test management	
		Quality management	
		Tracking of changes	
		Additional capabilities	

TABLE 1 (CONTINUED)

Reliability	Repository management
	Automatic backup capability
	Data access management
	Error processing capabilities
	Fault analysis capabilities
Usability	User interface
	Localization options
	Licensing options
	Ease of use
	Usage capabilities
	Quality of documentation
	Quality and availability of training
	Level of knowledge
	Integrity of user interface
	Web resources
	Diagnostic options
	Response time
	Ease of update implementation
Efficiency	Technical requirements
	Workload efficiency
	Performance
Maintainability	Distributor options
	Update traceability
	Update compatibility
	Maintainability of the final product
Portability	OS compatibility
	Data portability
	Support for data portability standards

IV. Conclusions and Future Work

The original paper contains a brief overview of the model-driven approach to software engineering within the variety of the CASE tools, which are proposed as supporting for MDA activities. It also proposes a specification for classification of MDA tools, which is one of the key components in foundation for certification of MDA tools.

The problem of compatibility among tools is not fictitious. The reality is, developers have to check the compliance of various tools as if it was done in "a code and fix" way, not in an effective way. The list of MDA committed companies and products [21] found on OMG website is useless, as it does not guarantee the compliance between the tools. Once again, there should be something (e.g., guidelines) defined to figure out the main features of each tool in accordance with OMG standards, how each tool should behave, what output each tool should provide, etc. In other words, the representatives of the approach should lean towards the side of the developers; the actual transition should be eased, instead of being complicated.

One of the options on how provide compatibility among the variety of MDA tools is tool certification. For example, Microsoft has created especial certification program in order to verify that specific software is compatible with Windows operating system. One of the directions the authors propose for the future work is the actual evaluation of the tools (example is shown in [1]), as well as the development of the online version of specification. As for the next milestone in certification of MDA tools, the methodology for the certification program should be defined.

References

[1] A. Čerņičkins, "Modeļvadāmās arhitektūras rīku analītisks apskats," Maģistra darbs. Rīga: RTU, 2009.
[2] I. Sommerville, *Software Engineering*, 8th ed. Wokingham: Addison-Wesley, 2006.
[3] А. Вендров, *CASE-Технологии. Современные Методы и Средства Проектирования Информационных Систем*. Москва: Финансы и статистика, 1998.
[4] *ISO/IEC 12207. Systems and Software Engineering – Software Life Cycle Processes*, 2nd ed. ISO, 2008.
[5] O. Nikiforova, A. Cernickins, and N. Pavlova, "Discussing the difference between model driven architecture and model driven development in the context of supporting tools," in *Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA) 2009*, September 2009, pp.1-6.
[6] W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of WESTCON*. San Francisco: IEEE Press, 1970, pp. 1-9.
[7] B. Boehm, "A spiral model of development and enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no.4. New York: ACM, 1986, pp. 14-24.

[8] "Rational unified process. Best practices for software development teams," IBM. [Online]. Available: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf. [Accessed: September 2009].
[9] "Microsoft Solutions Framework version 3.0 overview," Microsoft Corporation. [Online]. Available: http://www.microsoft.com/downloads/details.aspx?familyid=a71ac896-1d28-45a4-880c-8b0cc8265c63. [Accessed: September 2009].
[10] D. Lubiņa, "Programmu izstrāde ar Agile žiglāka un ražīgāka," Exigen Services. [Online]. Available: http://www.exigenservices.lv/newsroom/exigen-services-in-the-news/09-01-08-2021. [Accessed: September 2009].
[11] S. Barclay and S. Padusenko, "CASE tools," Queen's University. [Online]. Available: http://educ.queensu.ca/~compsci/units/casetools.html. [Accessed: September 2009].
[12] R. Bendraou, P. Desfray, M. Gervais, and A. Muller, "MDA tool components: a proposal for packaging know-how in model driven development," *Software and Systems Modeling*, vol.7, no.3. Berlin: Springer, 2008. pp. 329-343.
[13] J. Krogstie, "Integrating enterprise and IS development using a model driven approach," in *Proceedings of 13th International Conference on Information Systems Development – Advances in Theory, Practice and Education*. New York: Springer Science+Business media, 2005, pp. 43-53.
[14] M. Guttman and J. Parodi, *Real-Life MDA: Solving Business Problems with Model Driven Architecture*. San Francisco: Morgan Kaufmann Publishers, 2007.
[15] "MDA guide 1.0.1." Object Management Group. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. [Accessed: September 2009].
[16] S. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled: Principles of Model-Driven Architecture.* San Francisco: Addison-Wesley, 2004.
[17] A. Brown, J. Conallen, and D. Tropeano, "Models, modeling, and model driven development," in *Model-Driven Software Development.* Berlin: Springer, 2005, pp. 1-17.
[18] A. Cernickins and O. Nikiforova, "An approach to classification of MDA tools," in *Scientific proceedings of Riga Technical University*, 5th Series, Computer Science, Vol. 38. Riga: RTU, 2009, pp. 72-83.
[19] O. Nikiforova, A. Cernickins, and N. Pavlova, "On the tool chain for model driven architecture and model driven development: drawing a distinction," in *Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, September 2009. Riga: JUMI Publishing House Ltd., 2009, pp. 408-415.
[20] A. Kleppe and J. Warmer, "Getting started with modeling maturity levels," DevX.com. [Online]. http://www.devx.com/enterprise/article/26664. [Accessed: September 2009].
[21] "Model driven architecture. Committed companies and their products." [Online]. Available: http://www.omg.org/mda/committed-products.htm. [Accessed: September 2009].

Antons Cernickins, born in 1985. Earned a degree of Bachelor of engineering sciences in 2007, a degree of Master of engineering sciences in 2009 – both at Riga Technical University, Latvia. Now, a doctoral student at Institute of Applied Computer Systems, Riga Technical University.

He works as Information Systems Engineer at Department of Applied Computer Science, Riga Technical University. Fields of interests: computer science. Special interests: modeling, model-driven development.

Oksana Nikiforova has received a Doctor's degree (Dr.sc.ing) of information technologies in engineering science (system analysis, modeling, and designing sub-sector) from Riga Technical University, Latvia, in 2001.

She is presently an Associated Professor in the Department of Applied Computer Science of Riga Technical University, where she has been working since 1999. Her current research interests include object-oriented system analysis and modeling especially its issues in the framework of Model Driven Architecture. In these areas, she has published extensively and has been awarded several grants. She has participated and managed several research projects related to the system modeling, analysis, and design, as well as participated in several industrial software development projects.

Antons Čerņičkins, Oksana Ņikiforova. Modeļvadāmās arhitektūras (MDA) atbalsta rīku sertifikācijas pamatojums: specifikācijas definēšana
Vispārīgajā gadījumā, programmatūras izstrāde iekļauj sevī vairākus procesus izstrādes dzīves cikla uzturēšanai un pārvaldībai. Viens no svarīgākiem aspektiem, kuru ir jāņem vērā sakot programmatūras sistēmas izstrādi, ir piemērotāko izstrādes rīku izvēle. Ievērojot mūsdienas tendenci, kas ir saistīta ar kopējā programmatūras sarežģītības līmeņa pieaugumu, nepiemērota rīka izvēle var kļūt par noteicošo projekta īstenošanas neveiksmēs. Tādējādi, viens no galvenajiem IT industrijas mērķiem, kas ir saistīts ar programmatūras izstrādes procesa vienkāršošanu, optimizēšanu un automatizēšanu, paliek nesasniegts. Par vienu no paņēmieniem kā tikt galā ar pieaugošo sarežģītību, turklāt arī vienkāršojot programmatūras izstrādes procesu, var uzskatīt modelēšanas pieeju. Modeļvadāmā izstrāde, kas, galvenokārt, ir pārstāvēta ar organizācijas OMG piedāvāto modeļvadāmo arhitektūru (MDA), pretendē uz to, lai kļūtu par nākamo evolūcijas posmu programmatūras izstrādē. Lielākā daļa rīku, kas pašlaik ir pieejami tirgū, tiek pozicionēti kā tādi, kas ir „saderīgi ar MDA pieeju". Turklāt, šis saderīguma līmenis nekad arī netika pārbaudīts. Tātad, ir nepieciešams definēt attiecīgo sertificēšanas programmu, ar kuras palīdzību kļūtu iespējams novērtēt rīku atbilstību OMG standartiem. Šajā rakstā ir piedāvāts pamats MDA rīku sertificēšanas procedūras īstenošanai: tas ietver sevī visvispārīgāko īpašību un iespēju specifikāciju, kas ir domāta rīku atbilstības līmeņa noteikšanai.

Антон Черничкин, Оксана Никифорова. Основа для сертификации средств разработки основанной на моделях архитектуры (MDA): определение спецификации
Современный подход к разработке программного обеспечения (ПО) включает в себя широкий спектр процессов, используемых для управления всем жизненным циклом разработки. В данном контексте, одним из важнейших аспектов в разработке ПО является выбор подходящих средств разработки. Сейчас, когда программные системы становятся всё более и более сложными Выбор неподходящих средств разработки может стать одной из причин провала проекта. Из этого следует, что основная проблема всей индустрии — упрощение, оптимизация и автоматизация процесса разработки ПО — всё ещё остаётся актуальной. Одним из способов упрощения разработки ПО является использование моделирования. Основанный на моделях подход к разработке ПО, главным образом, представленный разработанной консорциумом OMG основанной на моделях архитектурой (MDA) — как наиболее популярной и передовой — претендует на то, чтобы стать следующей эволюционной вехой для всей индустрии ПО. Большинство средств разработки, доступных сегодня на рынке, их разработчики позиционируют как «совместимые с MDA», хотя они никогда не проверялись или анализировались на предмет такой совместимости. Исходя из этого, следует определить специальную программу сертификации, благодаря которой станет возможным оценивать основные характеристики средств разработки на предмет соответствия стандартам OMG. Данная статья предлагает основу для сертификации средств разработки MDA: в частности, в ней определена спецификация общих особенностей и возможностей средств разработки MDA, которую возможно использовать для того, чтобы прояснить уровень соответствия каждого средства разработки стандартам OMG.