UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <http://www.upgrade-cepis.org/>

Publisher
UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by Novática <http://www.ati.es/novatica/>, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática,* <http://www.ati.es/>)

UPGRADE monographs are also published in Spanish (full version printed; summary, abstracts and some articles online) by Novática

UPGRADE was created in October 2000 by CEPIS and was first published by Novática and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>)

UPGRADE is the anchor point for UPENET (UPGRADE European NETwork), the network of CEPIS member societies' publications, that currently includes the following ones:
- Informatica, journal from the Slovenian CEPIS society SDI
- Informatik-Spektrum, journal published by Springer Verlag on behalf of the CEPIS societies GI, Germany, and SI, Switzerland
- ITNOW, magazine published by Oxford University Press on behalf of the British CEPIS society BCS
- Mondo Digitale, digital journal from the Italian CEPIS society AICA
- Novática, journal from the Spanish CEPIS society ATI
- OCG Journal, journal from the Austrian CEPIS society OCG
- Pliroforiki, journal from the Cyprus CEPIS society CCS
- Pro Dialog, journal from the Polish CEPIS society PTI-PIPS
- Tölvumál, journal from the Icelandic CEPIS society ISIP

Editorial Team
Chief Editor: Llorenç Pagés-Casas
Deputy Chief Editor: Francisco-Javier Cantais-Sánchez
Associate Editor: Rafael Fernández Calvo

Editorial Board
Prof. Wolffried Stucky, CEPIS Former President
Prof. Nello Scarabottolo, CEPIS Vice President
Fernando Piera Gómez and Llorenç Pagés-Casas, ATI (Spain)
François Louis Nicolet, SI (Switzerland)
Roberto Carniel, ALSI – Tecnoteca (Italy)

UPENET Advisory Board
Matjaz Gams (Informatica, Slovenia)
Hermann Engesser (Informatik-Spektrum, Germany and Switzerland)
Brian Runciman (ITNOW, United Kingdom)
Franco Filippazzi (Mondo Digitale, Italy)
Llorenç Pagés-Casas (Novática, Spain)
Veith Risak (OCG Journal, Austria)
Panicos Masouras (Pliroforiki, Cyprus)
Andrzej Marciniak (Pro Dialog, Poland)
Thorvardur Kári Ólafsson (Tölvumál, Iceland)
Rafael Fernández Calvo (Coordination)

English Language Editors: Mike Andersson, David Cash, Arthur Cook, Tracey Darch, Laura Davies, Nick Dunn, Rodney Fennemore, Hilary Green, Roger Harris, Jim Holder, Pat Moody, Brian Robson

Cover page designed by Concha Arias Pérez
"Golden Ratio" / © ATI 2008
Layout Design: François Louis Nicolet
Composition: Jorge Llácer-Gil de Ramales

Editorial correspondence: Llorenç Pagés-Casas <pages@ati.es>
Advertising correspondence: <novatica@ati.es>

UPGRADE Newslist available at
<http://www.upgrade-cepis.org/pages/editinfo.html#newslist>

ISSN 1684-5285

Monograph of next issue (June 2008)

"Next Generation Technology-Enhanced Learning"

(The full schedule of **UP**GRADE is available at our website)

CEPIS UPGRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org

Vol. IX, issue No. 2, April 2008

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by **Novática**, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*) at <http://www.ati.es/novatica/>.

MDA Manifestations

Bran Selic

In 2004 the author, along with several colleagues, published an article titled "An MDA Manifesto", which outlined a strategic vision for Model-Driven Development (MDD). That article identified the key elements that characterized this approach to software development and its value proposition. The present article contains an assessment of the progress made since then towards fulfilling that vision, based on practical experience in applying MDD in industry. The key impediments that are hindering a more extensive realization of that vision are identified and categorized. Finally, a long-term strategy is outlined for overcoming these hurdles.

Keywords: Model-Driven Architecture (MDA), Model-Driven Development (MDD), Object Management Group (OMG).

1 Introduction

Almost four years ago, some of my colleagues at IBM Rational and I co-authored an article entitled "An MDA Manifesto", which was first published in the *MDA Journal* and then again in the eponymous book by Frankel and Parodi [1]. The primary intent was to articulate our shared vision of model-driven development (MDD). IBM and its Rational business unit in particular were pioneers in the application of modeling methods to software development.

Jim Rumbaugh and Grady Booch, both of Rational (and both of whom were authors of the Manifesto article), were the primary designers of the Unified Modeling Language (UML), much of it based on their industry-leading earlier work in model-based object-oriented methodologies. Rational's modeling tools were and still are market leading MDD tools.

It is both interesting and instructive to reflect on that vision in the light of subsequent practical experience with MDD since the article was written. Has anything fundamental changed in the vision? What are the current states of practice and adoption of MDD? What stands in the way and how serious is it? The purpose of this article is to examine some of these issues and also to investigate potential strategies that would enable broader application of MDD in industry and a more comprehensive realization of the vision behind it.

It would have been ideal if all of the original authors were involved in this assessment, but, due to a number of operational reasons this was not feasible (for one, I have since retired and have a bit more time at my disposal than my co-authors). Nevertheless, I have maintained close contact with all of them and, although I certainly cannot claim to represent their views, I am confident that we share pretty much the same vision and objectives outlined in the original article.

2 The MDA Manifesto Revisited

A "manifesto" is an explicit declaration of set of principles and a plan of action for reaching some objectives. The original article identified three keystones of the Model Driven Architecture (MDA) initiative from the Object Management Group (OMG) [2], as interpreted by IBM's technical team responsible for its MDD strategy. These were:

- Use of higher levels of *abstraction* in specifying both the problem to be solved and the corresponding solution, relative to traditional software development methods (NB: in the original article, this was referred to as "direct representation").
- Increased reliance on computer-based *automation* to support analysis, design, and implementation.
- Use of *industry standards* as a means facilitating communications, product interworking, and technological specialization.

The following is a brief summary of the nature and rationale of each of these key elements. Readers interested in a more in-depth description should refer to the Manifesto article itself.

2.1 The Issue

In essence, the primary problem that MDD is intended to address is the often overwhelming complexity involved in designing and implementing modern software. The magnitude of this problem just keeps growing, as our demands for more sophisticated functionality and more dependable software increase (as Grady Booch notes, in some ways "software runs the world" [3]). It is, therefore, critical for us to understand the sources of this complexity lies and what can be done about them.

Author

Bran Selic is currently President of Malina Software Corp. In 2007, Bran retired from IBM Canada, where he was an IBM Distinguished Engineer responsible for defining the strategic direction for software modeling tools for the Rational brand. He is currently the chair of the OMG task force responsible for the UML standard. Bran is also an adjunct professor of computer science at Carleton University in Ottawa, Canada. <bselic@ca.ibm.com>.

In his seminal work on software development, "The Mythical Man-Month" [4], Fred Brooks Jr. identifies two kinds of complexity: *essential complexity*, which is inherent to a particular problem and, consequently, unavoidable, and *arbitrary complexity*, which is due to the methods and tools used to address the problem. Brooks points out that software designers face more than their share of arbitrary complexity.

For example, they often have to cope with the idiosyncrasies of traditional programming languages, in which a single uninitialized variable or misaligned pointer can have disastrous consequences, whose impact can extend far beyond the localized context in which the error was made. Similarly, many crucial and difficult to detect errors can be introduced in the process of translating complex domain-specific concepts into corresponding computer-program implementations.

The basic motivation behind MDD can be reduced to the elimination of arbitrary complexity, through the definition of improved methods and tools.

2.2 Abstraction

Abstraction is a primary technique by which human minds cope with complexity. By hiding from view what is irrelevant or of little consequence, a complex system or situation can be reduced to something that is comprehensible and manageable. When it comes to software, it is extremely useful to abstract away technological implementation detail and deal with the domain concepts in the most direct way possible. For instance, it is typically easier to view and comprehend a state machine as a graph, rather than to see it in the form of nested "case" statements in some programming language rife with distracting low-level syntactical details.

The MDD approach to increasing levels of abstraction is to define domain-specific modeling languages whose concepts closely reflect the concepts of the problem domain whilst minimizing or obscuring aspects that relate to the underlying implementation technologies.

To facilitate communications and understanding, such languages use corresponding domain-specific syntactical forms. This often means using non-textual representations such as graphics and tables, which more readily convey the essence of domain concepts than text.

2.3 Automation

Automation is the most effective method for boosting productivity and quality. Software, of course, is an excellent medium for exploiting automation, since the computer is in many ways the ideal machine for constructing complex machines. In case of MDD, the idea is to utilize computers to automate any repetitive tasks that can be mechanized, tasks which humans do not perform particularly effectively. This includes, but is not limited to, the ability to transform models expressed high-level domain-specific concepts into equivalent computer programs, as well as into different models suitable for design analyses (e.g., performance analyses, timing analyses). In case of executable modeling languages computer-based automation can also be used to simulate high-level models to help evaluate the suitability of a proposed design in all stages of development.

2.4 Standards

MDA is OMG's initiative to support MDD with a set of open industry standards. Such standards provide multiple benefits, including the ability to exchange specifications between complementary tools as well as between equivalent tools from different vendors (thereby avoiding vendor lock-in). Standards allow tool builders to focus on their principle area of expertise, without having to recreate and compete capabilities provided by other vendors. Thus, a performance analysis tools need not include a model editing capability. Instead, it can interact with a model editing tool using a shared standard .

As part of MDA, the OMG has defined a basic set of MDD standards for modeling languages (e.g., UML, MOF), model transform definition (MOF QVT), MDD process definition (SPEM), and a number of other areas. It is somewhat ironic that MDA is sometimes viewed as an approach to MDD that is contrary to the domain-specific languages approach, this is not the case, since many of the MDA standards are specifically designed to support specialization for domain-specific needs. The MOF language, for instance, is a language for defining domain-specific languages. Furthermore, UML can also be used to define different domain-specific languages by taking advantage of its profile mechanism. This not only allows reuse of the effort and ideas that went into the design of UML but also enables the reuse of existing UML tools. In many ways, this approach to domain-specific language design overcomes one of the major barriers that has impeded such custom approaches in the past: the lack of adequate tooling as well as the cost of maintaining it and evolving it. Thus, it is possible to reap the benefits of both standardization and customization.

3 The State of the Practice in MDA

While one can argue against concrete realizations of the MDA idea, such as the technical features of UML or MOF, it is hard to argue with any of its basic premises. Increasing the levels of abstraction and automation and the use of standards, executed properly, are all undeniably useful. Furthermore, there have been numerous examples of successful applications of MDA in large-scale industrial projects (cf. [5] [6])[1]. Yet, there is still a significant amount of controversy about whether or not MDA is useful. It is fair to say that the dominant perception among today's software practitioners is that MDA has yet to prove itself, or, at the ex-

[1] On the other hand, many enterprises that have achieved successes with MDD are inclined to keep them confidential in the belief that their use of MDD is an important advantage they hold over competitors.

treme end of the opinions scale, that it is a distracting academic fairy tale, concocted by software theologians who are disconnected from any practical reality[2].

I am unaware of any published results, but my personal estimate from numerous discussions with software development teams in industry is that the penetration of model-based methods hovers around 10%. If this stuff is really as good as claimed, why isn't everyone using it?

It turns out that there are numerous and varied reasons for this glacial pace of adoption. They can be roughly classified into technical, economic, and cultural.

3.1 Technical Hurdles

A major problem that plagues many software products these days is *usability*. In the case of MDA, it is mostly manifested in MDA tools. Although often endowed with diverse and very powerful functionality, such tools almost invariably tend to be extremely difficult to learn and to use. Users are typically faced with a bewildering spectrum of menu items arranged in seemingly arbitrary groupings. Common operations that should be easy to use often require complex and counterintuitive tool manipulations. One of the reasons for this is that, ironically, many of the tool designers and implementers are not themselves practitioners of MDD and, therefore, do not have a sense for how the tools should look and behave.

Consequently, poor tool usability is one of the biggest current impediments to greater penetration of MDD methods in practice.

A second major technical problem is that there is still very little theoretical underpinning for MDD. Much of the MDD technology that is available today was developed in ad hoc ways by industrial teams who were trying to solve specific problems in circumstances that do not afford the luxuries of reflection and generalization. As a result, when it comes to supporting MDD, we do not yet know precisely what works, what does not, and why. The result is not only gratuitous diversity but also substandard and inadequate technologies. In contrast, traditional programming-oriented methods and technologies have been studied amply and one can rely on a sound body of theory to ensure that common problems are avoided and solid and proven solutions are chosen.

The lack of a sound theory of MDD is also manifested in interoperability problems between MDD tools that often result in highly undesirable vendor lock-in for users. This is true even in the presence of standards.

3.2 Cultural Hurdles

Despite the availability of hard evidence of the success of MDA in practice, there is still insufficient awareness of its potential and its capabilities among practitioners who could be exploiting it. However, even in cases where a project team might be fully aware of the potential benefits of applying MDD, there is still a problem in adopting it due to the inevitable overheads whenever new methods and tools are introduced into a running production environment. It takes time to learn and adjust to new ways of working (not to mention that it may be necessary to support the old and the new methods and tools during phased cutovers). In today's highly competitive environment, where time-to-market is a fundamental driver of development, this overhead is difficult to accept, since the investment payback is generally deferred to subsequent projects.

However, perhaps the most difficult issue to overcome of all is the conservative mindset of many software practitioners. Because they tend to invest vast amounts of time and effort in mastering specific implementation technologies (which, due to their often arbitrary complexity, do require significant investment), many practitioners define their expertise in terms of computing technologies they have mastered rather than the domain in which they are working. For instance, they are much more likely to view themselves as, say, J2EE experts rather than as financial software experts. Consequently, there is often major resistance to technological change, even if the new technology could lead to better solutions for the specific domain problem on which they are working. This same technology-centered culture means that many software developers have a very superficial interest in and understanding of how the products they implement are to be used, which, in turn, leads to poor product usability discussed above.

One major barrier in overcoming all such cultural issues is the sheer number of software practitioners, which is estimated between 10 and 15 million [7]. This is, of course, a huge inertial mass that is very difficult to shift from its present cultural base.

3.3 Economic Hurdles

Today's prevailing business environment is focused on relatively short-term return on investment (ROI). Public companies report their results on a quarterly basis and a failure to meet profit expectations in a given quarter is likely to result in a falling stock prices and a shift of stockholders to other apparently more immediately profitable businesses. This has the unfortunate effect that most investment in technological development tends to be short-term. Consequently, it is hard to justify longer-term investments in new software development methods and tools, particularly if the payback is not guaranteed. And, to be fair, switching to MDD does not guarantee success, partly because of the other issues discussed above. For example, the risks of introducing MDD into a software development organization can be greatly mitigated if it is led by individuals with prior experience. Unfortunately, such expertise is still quite difficult to find and secure. And, with the aforementioned absence of a systematic foundation for MDD, organizations are of-

[2] Steve Mellor tells the following anecdote that epitomizes the current state of affairs in MDA: When he was asked over ten years ago about when he expected MDA to become mainstream, he suggested that it would likely happen within the following year and a half to two. And he has been giving the same answer to that question ever since.

ten left to fend for themselves through a risky process of trial and error.

Clearly, these are all substantial barriers to overcome and it seems likely that the pace of introduction of MDD in industrial practice will remain a slow for several years to come. In the next section, we describe some initiatives that could help accelerate this trend.

4 The Way Forward

There are at least three possible areas in which to address the problem of increasing the penetration of MDD in practice:

- Education
- Research
- Standardization

4.1 Education

Given the difficulties of changing the dominant technology-centric culture noted above, it is necessary to initiate such change through education, starting at the undergraduate level. This means instilling an understanding and respect for users among software engineering students. A primary need is comprehending the value that the product to be developed has for its customers and users. That, in turn, typically requires an understanding of the economic and business context of the product. In other words, what is needed is insight that extends beyond the immediate technological issues. Software engineering graduates must have an understanding and working knowledge of economic and business factors that influence what they design and build[3]. As Charles Babbage put it: "It is doubly important for the man of science to mix with the world".

In addition, designing software products that are used by people requires an understanding of human psychology. At present, the prevailing attitude among software developers seems to be that human factors constitute a second-order concern, to be addressed by user-interface design specialists once the main system architecture has been finalized. Often this is viewed as a mere matter of designing suitable graphical interfaces and menu items. The understanding that usability requirements might have a fundamental impact on the architecture of a software system is still rare among software professionals.

Last but not least, it is necessary to increase the introduction of MDD methods into software engineering education. Most current undergraduate curricula already include some basic elements of model-based engineering, such as courses on UML. But, with no systematic theoretical foundation on which to base this, the results are often haphazard and inadequate. To address that, more research is needed into the theory behind MDD.

[3] One additional benefit of a broader education is an understanding when technological solutions are appropriate and when they are not. There are many examples when technological solutions have created more problems than they solved.

4.2 Research

The research community has embraced the notion of MDD, partly because they see it is an opportunity to effect a dramatic sea change in software technology. For example, modeling languages can be designed to avoid the arbitrary complexity of current programming languages. This complexity is sometimes a barrier to the application of highly-effective engineering methods, such as the use of mathematical analyses to predict key system characteristics before the system is constructed. All too often in software, such characteristics remain unknown until the complete system is designed and built, at which point the cost of redesign can be prohibitive. New modeling languages can be designed that are specifically designed to support such analyses.

Yet, despite the eagerness with which researchers have accepted MDD, there are some issues with the current research efforts. One of them is that much of the research focuses on particular point problems, in large part because research funding is provided by industrial partners who are primarily interested in solving their immediate problems. Consequently, there is insufficient exploration of the theoretical foundations of MDD. What is needed, therefore, is an overall map of the MDD research space in which the various areas of exploration are clearly identified as are the relationships between them. Only when this is properly understood will we be able to talk of a comprehensive and systematic *theory of MDD.*

One recent initiative is intended to deal with this issue: the newly formed Centre of Excellence for Research in Adaptive Systems (CERAS) [8]. This is a multi-year research effort organized and funded by the Ontario Centres of Excellence (specifically, its Centre for Communications and Information Technology) and the IBM Center for Advanced Studies, with the objective of exploring the foundational issues behind a number of related emerging computing technologies, including MDD. One of its primary objectives in the domain of MDD is to define a comprehensive framework for MDD research (the research map mentioned above). Another key objective is to provide a communal focal point and a kind of clearinghouse for MDD research worldwide. In addition to exploring the foundations of MDD, CERAS will be doing research in the following areas (and, likely, others):

- Modeling language semantics and design (including domain-specific languages).
- Model transformations (including model-to-model and model-to-code).
- Model analysis (safety and liveness property checking).
- Model-based verification.
- Model management.
- MDD methods and processes.
- MDD tooling.

4.3 Standards

The role of standards, *de facto* or *de iure*, is key to the success of any widely used technology. Standardization will not only allow the distilling of proven results in a vendor-neutral manner, it will also facilitate specialization by providing a common foundation for interworking between specialties. Standardization bodies, such as the OMG are not only useful, but necessary. However, given the highly competitive nature of the industry and the unprecedented flexibility of software, it is difficult to expect software vendors to voluntarily conform to standards. Therefore, users of MDD tooling and software professionals in general should do their utmost to pressure vendors to contribute and adhere to software standards

Clearly, there should be a close link between research, industry, and standards bodies. It is critical that only things that are both well understood and proven in practice be standardized. Premature standardization can be counterproductive.

5 Summary and Conclusions

MDD has the potential to provide significant improvements in the development of software. It is based on sound and time-proven proven principles: higher levels of abstraction, higher levels of automation, and standardization. Furthermore, there are numerous verifiable examples of successful applications of MDD in industrial practice, that are existence proofs of its viability and value. Yet, the use of MDD is still an exception rather than the norm. This is due to the not insignificant hurdles that need to be overcome. Although many of these are technical in nature, what may be surprising to some is that the most difficult hurdles to overcome are the ones stemming from the current idiosyncratic culture of software development. This culture places far too much emphasis on technology and not enough on technology users and their needs. It is a very pervasive culture that is sustained in part by the current business climate that is heavily focused on short-term gain and, thus, discourages investment in new methods and tools.

In such circumstances the best that can be expected is a gradual introduction of MDD, facilitated primarily through changes in educational curricula and investment in foundational research. Software engineers must be much better educated in human factors and the workings of the marketplace; they should view technology more as a tool rather than as an end unto itself. At the same time, we need to research and develop a systematic theory of MDD, to ensure that the corresponding technology and methods are well understood, useful, and dependable.

References

[1] G.. Booch et al. "An MDA Manifesto", en Frankel, D. and Parodi, J. (eds.) The MDA Journal: Model Driven Architecture Straight from the Masters. Meghan-Kiffer Press, Tampa, Florida, 2004 (pp. 133-143).

[2] OMG. MDA Guide (version 1.0.1), OMG document number omg/03-06-01, <http://www.omg.org/docs/omg/03-06-01.pdf>, 2003.

[3] G. Booch. Saving Myself. <http://booch.com/architecture/blog.jsp?archive=2004-00.html>, July 22, 2004.

[4] F. Brooks. The Mythical Man – Essays on Software Engineering (Anniversary edition). Addison-Wesley, 1995. ISBN: 0201835959.

[5] OMG. MDA Success Stories (web page). <http://www.omg.org/mda/products_success.htm>.

[6] N.J. Nunes et al. Industry Track papers in UML Modeling Languages, and Applications – 2004 Satellite Activities (Revised Selected Papers), Lisbon, Portugal, October 2004, Lecture Notes in Computer Science, vol. 3297, Springer-Verlag, 2005 (pp. 94-233). ISBN: 3540250816.

[7] IDC. The 2007 Worldwide Professional Developer Model. IDC document number #207143. <http://www.idc.com/getdoc.jsp?containerId=207143>, 2007.

[8] Ontario Centres of Excellence (OCE). Centre of Excellence for Research in Adaptive Systems (CERAS). <https://www.cs.uwaterloo.ca/twiki/view/CERAS/CerasOverview>.