MDA: The Vision with the Hole?

Ashley McNeile

Introduction

MDA (Model Driven Architecture) is a flagship initiative of the OMG (Object Management Group) presenting a new vision of how application systems should be developed and managed.

Launched in 2001, MDA is now having a major impact on the work of the application development methods and tools community. Currently, there are at least 40 tools that incorporate at least one of the major aspects of MDA (the OMG website has a page at http://www.omg.org/mda/committed-products.htm listing 42 of them) and there are perhaps another couple of dozen in the works. According to Jon Siegel, the OMG's vice president of technology transfer, MDA adoption has caught on faster than any other OMG standard. PriceWatersCoopers' Technology Centre recently predicted that MDA will be one of the most important methodologies in the next two years

As is normal in this industry, much of the literature on MDA is saturated with new terminology and hype. This makes it hard to get any insight into the real issues that are discussed feverishly in the meeting rooms of OMG conferences or the offices of tool vendors.

In this article I want to describe one of the key MDA debates and expose some of the implications arising from it. In particular I want to show that, if the development of the ideas and tools for MDA continues on its current trajectory, there is a danger that one of its key objectives will be missed.

The Vision

The vision of MDA is both simple and grand. It objective is to decouple the way that application systems are defined from the technology they run on. The purpose of this decoupling is to ensure that the investments made in building systems can be preserved even when the underlying technology platforms change.

The key buzz-words of MDA are PIM (Platform Independent Model) and PSM (Platform Specific Model). Figure 1 shows the overall scheme.

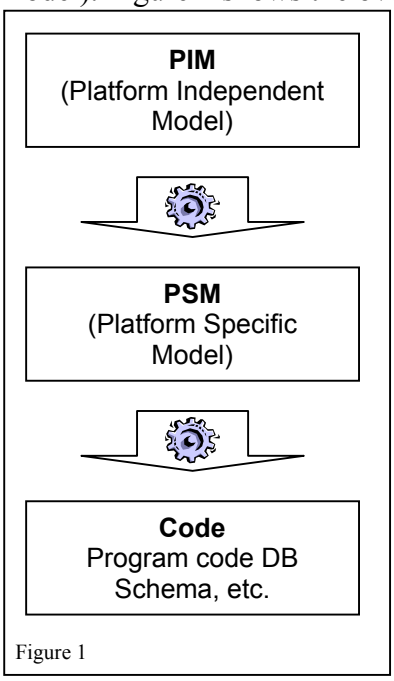

Figure 1

The PIM is (in the words of the OMG) "a representation of business functionality and behaviour, undistorted by technology details".

In order to implement a PIM on a specific platform, a tool is used to generate the PSM from the PIM. The tool understands the target technology and knows how to translate the logical constructs of the PIM into a suitable form for the chosen platform. These mappings can be complex, and a given element of the PIM may be mapped to multiple elements in the PSM. For instance a single business object of the PIM may be mapped to an SQL DB Table definition, an EJB Entity Bean and a Remote Interface.

The PSM is also a model, but now at the design level and reflecting the mapping to a target platform. The final stage is to generate from the PSM the code and other technical artefacts (SQL DDL, IDL Interfaces, deployment descriptors, etc.) needed to deploy and run the system.

ashley.mcneile@metamaxim.com
www.metamaxim.com

A central point about the MDA approach is that automation is used to produce the PSM and Code from the higher level artefacts, as indicated by the gear-wheel symbols in Figure 1. This automation not only speeds the development process but also ensures that the Code is a faithful representation of the higher level models, and therefore that the models are a faithful representation of the Code. That's the theory, anyway!

The benefit claimed for MDA is that the PIM, being technology independent, is the best way to capture and preserve the investment, intellectual and financial, made in the development of an application. If the application needs to be deployed on several platforms, or migrated from one platform to another as technology changes, the required code can be regenerated from the PIM. This, it is argued, is faster and cheaper than migrating the deployed code.

Two Interpretations of MDA

A closer look at what tool developers and methodologists are doing reveals that there are two rather different interpretations of how the MDA vision might be achieved. These two schools of thought have been termed "elaborationist" and "translationist" (terms that were, I believe, originally coined by Stephen Mellor).

I will describe each of these approaches in turn. A schematic of the first approach, the elaborationist, is shown in Figure 2.

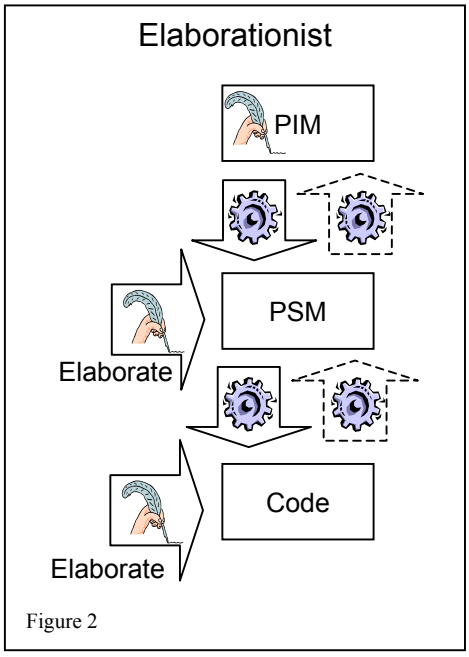

Figure 2

In the elaborationist approach, the definition of the application is built up gradually as you progress through from PIM to PSM to Code.

Once the PIM has been created, the tool generates a skeleton or first-cut PSM which the developer can then "elaborate" by adding further information or detail. Similarly, the tool generates the final code from the PSM, and this may also be elaborated. (Whether elaboration is required at both PSM and Code levels depends on the tool and the circumstances.)

As a result of elaboration, it is possible that the lower level models get out of step with the higher ones. For this reason, tools generally support the capability to regenerate the higher level models from the lower, shown in the diagram as upward pointing arrows. The ability to generate downwards, modify, and re-synchronise by generating upwards is known as "round-trip engineering".

Clearly, in this approach, the generated artefacts (the PSM and the Code) must be understandable to the developer, otherwise modification (elaboration) would be not be possible. Contrast this with, say, the output from a language compiler where the generated code is not intended to be touched and is generally not human readable.

The elaborationist approach to MDA probably represents the mainstream. It is essentially a mechanisation of the familiar OO development approach, where first an analysis model (OOA) is produced, followed by design model (OOD) and finally code. In the traditional approach, however, there was no automation of the production of models, either forwards or backwards. MDA brings this automation.

ashley.mcneile@metamaxim.com
www.metamaxim.com

The second interpretation of MDA is the translationist, and a schematic of this approach is shown in Figure 3.

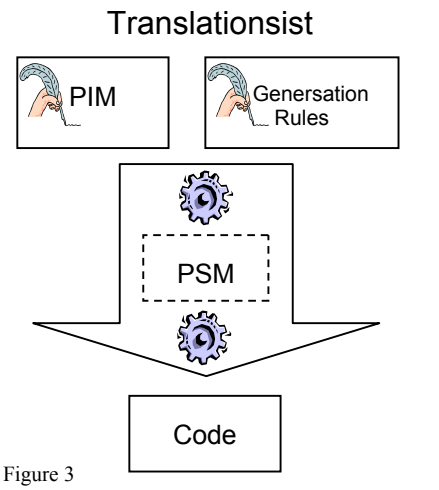

Figure 3

In this approach, the PIM is translated directly into the final code of the system by code generation.

The transformation (or translation) of the PIM into the final code is performed by a sophisticated code generator, sometimes called a "Model Compiler", and symbolised by the large arrow. It is driven by Generation Rules that describe how the elements of the PIM are to be represented in the final code.

The PSM is an intermediate stage in the generation and is internal to the code generator. It is generally not visible or editable by the developer and that is why I have shown it with a dashed outline.

A feature of the translationist approach is that the downstream artefacts (PSM and Code) are not further elaborated or amended by hand. The PIM (plus the Generation Rules) are the full source of the generated system and there is no need to look at or amend the generated artefacts, in the same way that there is no need to look at or amend the output from a programming language compiler. Because changes are made upstream and propagated by regeneration, round-trip engineering is not required to keep everything synchronised.

The translationist approach derives, in the main, from work on real-time and embedded systems, in particular that of Sally Shlaer and Stephen Mellor in the 1980s. Although it is now being repositioned as a mainstream MDA technique, suitable also for business information and transactional systems, current use of the translationist approach is still mainly in this domain.

Over the past year or so about half a dozen books about MDA have been published (the OMG has a page listing these books at http://www.omg.org/mda/reading-room.htm). I want to mention two of the books, shown in Table1, one giving the elaborationist view and the other the translationist view. These books were written by people deeply involved in developing MDA and currently represent the definitive accounts the two approaches.

Table 1 includes brief quotes showing what the authors of each say book say about the other approach. It would not be true to say that the two camps are war, because the MDA umbrella is large enough at present to accommodate both. But is clear that the proponents of each have robust opinions about their superiority!

TWO BOOKS ABOUT MDA		
TITLE	**MDA Explained. The Model Driven Architecture: Practice and Promise**	**Executable UML. A Foundation for Model Driven Architecture.**
AUTHORS	Anneke Kleppe Jos Warmer Wim Bast	Stephen Mellor Marc Balcer
APPROACH DESCRIBED	Elaborationist	Translationist
WHAT IT SAYS ABOUT THE OTHER. APPROACH	"Executable UML [= Translationist] is suitable within specialized domains, but even there the benefits are less than you would expect …" (Page 36)	"Because elaboration is stupid" (Page 303)

Table 1

ashley.mcneile@metamaxim.com
www.metamaxim.com

Modelling Behaviour

Whatever interpretation of MDA is held, the modelling approach used in the PIM must address the behaviour of the application. The OMG has the following statement on its website:

"Fully-specified platform-independent models (*including behaviour*) can enable intellectual property to move away from technology-specific code, helping to insulate business applications from technology evolution and further enable interoperability."
(http://www.omg.org/mda). [The italics are mine.]

Moreover, if behaviour is fully captured there is the potential to execute the model as kind of prototype, which can be used to validate that the model works as expected and that it meets the requirements of the users. Richard Soley, CEO of OMG, in his presentation "MDA: An Introduction" says that one of the aims of MDA is that "Models are *testable and simulatable*". Oliver Sims, an active member of various OMG Task Forces who served for several years on the OMG Architecture Board, says in his presentation "MDA – The Real Value" that "The aim [of MDA] is to build computationally complete PIMs". Here the term "computationally complete" means executable, and therefore testable. (Both these presentations are available at http://www.omg.org/mda/presentations.htm).

This brings us to the key question of how behaviour is specified in a PIM, and whether a PIM can indeed be executed and tested. While there is general agreement within the MDA community that the structural aspects of a system are captured using the structural modelling techniques of UML, there are different views about how the behaviour or dynamics of a system should be represented. This divergence of view is the source of the hole mentioned in the title of this article.

The statements shown in Table 2, taken from the two books mentioned earlier, illustrate this difference of thinking.

PIM BEHAVIOUR DEFINITION		
APPROACH	Elaborationst	Translationist
VIEW OF BEHAVIOUR	"The dynamics of the system are represented by pre- and post-conditions on operations." (MDA Explained: page 36)	"The behaviour of a system is driven by objects moving from one stage in their lifecycle to another in response to events. " (Executable UML: page 6)
METHOD OF MODELLING BEHAVIOUR	Pre- and post-conditions on operations in class definitions.	Statemachines and Activities.
LANGUAGE USED	Object Constraint Language (OCL)	Action Language (AL)

Table 2

In the elaborationist view, behaviour is specified by defining the conditions (pre-conditions) that must pertain for an operation to take place, and the effects (post-conditions) that result from its execution. The pre- and post-conditions are specified in the Object Constraint Language (OCL), which is a formal language for specifying assertions (formally, an "expression language"). The actual operation itself is not specified in the PIM, although in some cases it can be inferred. To quote again from the book, MDA Explained: "For relatively simple operations the body of the corresponding operation might be generated from the post-condition, but most of the time the body of the operation must be written in the PSM" (page 36). This means that, in general, the PIM is not an executable artefact.

In the translationist view, behaviour is specified as statemachines and Action Language. The statemachines specify the states that an object can have in its lifecycle, and how an object moves

ashley.mcneile@metamaxim.com
www.metamaxim.com

from state to state as events take place. Activities (specified in Action Language) specify what an object does when its state changes. Action Language, unlike OCL, is an imperative language. This approach results in models that are executable, and therefore testable.

Object Constraint Language and Action Language are both OMG standards, associated with the overall UML standard but essentially separate from each other.

To summarise. Behaviour in a PIM is specified differently in the elaborationist and translationist approaches. PIMs constructed according to the former approach are not executable or testable, whereas PIMs constructed according to the latter approach are.

The Hole

The title of this article speaks of a hole. This hole is a consequence of the likely adoption pattern of the competing MDA approaches.

As noted earlier, the translationist approach derives from work in real-time/embedded systems. The use of statemachines for specifying behaviour in this type of system has a long history, easily pre-dating UML. There is reason to suppose, therefore, that the translationist approach, with its reliance on statemachines, will fit well with current practice in this domain.

In business information and transactional systems, statemachines are seldom used. In this domain, behaviour is normally modelled using the Use Case and Collaboration diagrams of UML. These diagrams, however, are not suitable for code generation or model execution and do not feature in MDA as mainstream artefacts, although they may, of course, still be produced. To people working in this domain, it is likely that the translationist approach will be seen as alien. Indeed, to date there is no evidence that translationist tools are making any penetration into this market.

At least in the short to medium term, it is likely that the approaches and associated tools will be positioned by vendors to address the domains that represent their historical origins: elaborationist for business systems and translationist for real-time. For builders of business systems, this would mean that MDA will fall short of one of its key objectives – it will not be possible to execute, test or simulate a PIM.

Where do we go from here?

If this prognosis is correct, and builders of business systems will not be able to execute and test their PIMs, something has to give. Perhaps the notion that the "PIM is testable and simulatable" will have to be quietly dropped as a feature of MDA and instead be viewed as a nice bonus if it can be achieved. My personal opinion is that this would be a pity, for two reasons:

1. The ability to test a PIM is valuable. It not only provides the developers with the assurance that the dynamics they have modelled are coherent but, more importantly, gives users and other non-technical stakeholders a means of interacting with the model and testing it themselves during its development. This would offer real benefits where the requirements are complex or uncertain, by allowing significant testing to take place in the context of the PIM where making corrections is easier and cheaper.
2. If executing the PIM is not possible, the MDA process becomes a kind of mechanised waterfall. Of course the development will be iterative but, however good the round-trip engineering facilities, the temptation will be not to go back to the top with each iteration. Synchronisation of the (non-executable) behavioural modelling in the PIM with the executable will then tend to evaporate, and trust in the PIM as a true representation of what has been built will be lost. This will make MDA easy prey for the Agile Modelling brigade

ashley.mcneile@metamaxim.com
www.metamaxim.com

who will argue that the process is tool-centric busy work of dubious value, and that we should go back to using a whiteboard.

If the notion of a testable PIM is to become a reality for all users of MDA, the translationists will need to back up the claim that their approach works well for business applications by establishing a track record of successful projects. Until this happens the elaborationists will continue to say, as they do now, that the approach is only suitable for specialised domains.

ashley.mcneile@metamaxim.com
www.metamaxim.com