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

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/>.

A Brief History of MDA

Andrew Watson

On 8th March 2000 Object Management Group (OMG) announced that its Architecture Board had voted to adopt the Model-Driven Architecture (MDA) as both the strategic approach to developing OMG's own integration standards, and as its recommended application development technique. MDA was devised before the term "Service-Oriented Architecture" (SOA) became fashionable, and when many Business Process Management (BPM) techniques and languages were in their infancy. However, through a combination of foresight and good fortune MDA techniques are, if anything, more relevant today in the world of SOA and BPM than they were in 2000. This short history of MDA charts the influences that led to its creation, shows how its has evolved, and outlines the contributions it can make in the future.

Keywords: Business Process Management (BPM), Model-Driven, Model-Driven Architecture (MDA), Object Management Group (OMG), Service-Oriented Architecture (SOA), Unified Modelling Language (UML).

Author

Andrew Watson is Vice President and Technical Director at OMG. Andrew has overall responsibility for OMG's technology adoption process, and also chairs the Architecture Board, the group of distinguished technical contributors from OMG member organisations which oversees the technical consistency of OMG's specifications. Previously Andrew researched service oriented architectures and their type systems with the ANSA core team in Cambridge, wrote Lisp compilers at Harlequin, and worked on distributed systems and software engineering at HP Laboratories. <andrew@omg.org>.

1 Origins

OMG came into being in the late 1980s as an independent, not-for-profit industry organisation to specify object-based middleware that could help solve the growing problem of integrating IT systems that spanned multiple platforms. The resulting Common Object Request Broker Architecture (CORBA®) middleware and its related specifications became very widely used, and by 1999 an analyst survey [1] found that "*70 percent of respondents cited CORBA compliance as 'important' or 'very important' to integration, outpacing every other factor in the survey*".

From the mid-1990s OMG also began developing specialised middleware-based interoperability standards for application domains ranging from finance through telecoms to healthcare. In each of these areas, groups of highly-qualified domain experts devoted several man-years of effort to specifying standards for domain application components, using CORBA's Interface Definition Language (IDL) to specify the service interfaces that these components would provide and use. These standard services and the CORBA middleware they used to communicate formed the basis of OMG's Service-Oriented Architecture, known as the Object Management Architecture (OMA).

By the late 1990s OMG had used CORBA and IDL to specify several families of domain-specific services for different industries, and in the process had identified two limitations with this purely middleware-based approach to creating integration standards:

- IDL provides a precise way to specify the structure of the data that application components exchange with each other. However, since the CORBA middleware doesn't need to know or constrain the order in which the data are exchanged or the semantic relationships between the data fields, IDL doesn't provide any way of specifying these parts of the design. These important application-level constraints could only be captured using imprecise natural language, which was becoming more and more of a difficulty as the domain specifications became more sophisticated.
- The application component designs created by OMG's Domain groups were often equally usable with other middleware architectures; for instance, the Java Transaction Service (JTS)[2] is a translation into pure Java interfaces of the functions defined by the CORBA Object Transaction Service (OTS)[3]. However, converting OMG's specifications from IDL to another platform involves knowledge of both the source CORBA environment and the chosen target, and not all designers have this detailed knowledge. Furthermore, where there are multiple options for translating an interface element, multiple mappings are possible; hence different designers would likely generate different (and incompatible) translations.

It became clear that each of OMG's domain groups incorporated a large pool of priceless domain expertise, and in the process of creating domain interoperability specifications were actually creating valuable models for standard subsystems. However, the difficulty of precisely capturing non-structural aspects of the interfaces or translating the interfaces into other notations were preventing this valuable work being used to its full potential.

2 MDA is Born

To address these concerns, OMG decided to switch from a middleware-based approach to specifying SOA services to a platform-independent approach which could capture

behavioral as well as structural aspects of interoperability. These Platform-Independent Models (PIMs) could then be translated via standardised transformation rules into interface specifications for any particular application platform, such as CORBA, Java, or one of the emerging families of "Internet Middleware" based on eXtensible Markup Language (XML), such as Simple Object Access Protocol / Web Services Description Language (SOAP/WSDL).

During the mid-1990s OMG had also helped broker agreement within the fledgling Object-Oriented (OO) visual modelling community, creating the Unified Modelling Language (UML®), a family of 13 diagram types for the visual representation of different static and dynamic aspects of application software design. Applying UML and the Meta-Object Framework (MOF™), the standardised metadata framework on which it's based, to the problem of creating platform-independent service specifications led to the creation of MDA. Use of formal, rigorously-defined modelling languages is the key; only with a precise definition of the meaning of every construct in the language is it possible to mechanise translating the PIM into the implementation artefacts for the target platforms (such as IDL or Java interfaces), and thereby achieve the goal of platform independence.

MDA was thus first mooted as a way of creating standards. However, it was immediately obvious that the same tools and techniques could be used to build applications; transforming a precise but abstract design into the framework of an application is a very similar problem to translating into a platform-specific standard. Depending on the modelling language being used, it might not be possible to completely specify a whole application as a PIM, but at the very least a large part of the application's static structure and interface design could be captured and then translated into code or other platform-specific artefacts. Many applications use multiple platforms and programming languages simultaneously; transforming different parts of a common PIM into complementary Platform Specific Models (PSMs) for the different platforms used helps address the problem of maintaining common interface definitions across a variety of implementation technologies. By creating application outlines directly from models, and helping to automatically write the "glue code" between different platforms within one application, it was initially estimated that even the early modelling technology available at the time could be used to create 30-40% of the application code directly from an MDA PIM, yielding useful increases in software quality and productivity.

It's important to note that the PIM is one of the main products of the MDA design process, not just a transient stage in the process. If changes are later needed as the specification or application evolves, it is the PIM, not the generated artefacts, that are modified. In short, MDA treats design as a *product* not a *process*.

3 OMG Specification Developments to Support MDA

Once the MDA vision was in place, OMG began work to evolve its modelling specifications to better support it. The main results were the UML 2 revision and ongoing work on the MOF 2 metamodelling specification.

Although the basic structure and specification of UML 1 and UML 2 are much the same, the detailed design and underpinnings of UML have been shaped by MDA over the past 7 years. Even today, many engineers use UML merely as a way of sketching software designs, as an aide memoire or a way of documenting or communicating design. Since sketches are meant to be read by people, not tools, some imprecision, while undesirable, can be tolerated, or even go completely unnoticed. When UML is used for only for sketching, the appearance and readability of the diagrams matters much more than the underlying representation of the model itself within the modelling tool. Although UML 1 provided a formal, standard metamodel for each diagram type, common features of these metamodels had not been factored out, and there were also some inconsistencies between the metamodels for different diagrams.

With the advent of MDA, that began to change. OMG began a major revision of the UML specification to UML 2; one of the aims of this revision was to improve the quality of the UML metamodel to make it easier to extract information from them as part of the MDA process. At the same time, UML tool vendors started to devote more effort to producing compliant models corresponding to the diagrams that their tools were used to create. As a result both today's UML and the tools that implement it are much better suited to model-driven development techniques.

MOF, the metamodelling foundation on which all modelling languages used for MDA are based, has also evolved over the last seven years. A new version of the core MOF specification was released at the same time as UML 2, building on the experience of MOF 1 and UML 1 to make MOF into a truly versatile foundation for models and model transformation. The MOF2 Core specification contains the basic metamodelling framework, and has two compliance points: EMOF (Essential MOF) and CMOF (Complete MOF). Further specifications provide extra MOF-related features. Perhaps the most important is the MOF Query, View & Transformation (QVT) specification, which provides standardised mechanisms for making model-to-model transformations. Such transformations, for example from PIM to PSM, lie at the heart of MDA, and providing a standard language for executing them allows libraries of standard transformations to be created. Other MOF-related standards include Versioning and Lifecycle, which provides standard ways to support version control of MOF models. The work on MOF standardisation continues within OMG, building on this core set of MOF standards, and providing the essential tools for the metadata manipulation that underpins MDA.

4 Which Modelling Language?

At the time MDA was first mooted, and even more so today, most software modelling uses UML. By 2004 it was estimated that more than 2/3 of all industrial applications

used at least some UML during their specification phase, with 82% of developers saying that they planned to use UML in future [4]. Because of its ubiquity, it was clear from the start that UML would be the language predominately used for MDA. However, to help apply UML and MDA to the widest-possible range of application areas, OMG is also publishing a rapidly-expanding family of UML profiles which extend and adapt UML to allow it to represent concepts in specific application domains. Examples include:

- MARTE – A UML Profile for Modelling and Analysis of Real-Time and Embedded systems.
- SysML – This extends UML to support modelling of complex systems with human and hardware as well as software components.
- EAI – A UML profile for Enterprise Application Integration.
- Testing – A UML profile defining a language for designing, visualizing, specifying, analyzing, constructing and documenting software test systems.
- Voice – A UML profile for modelling voice dialogs in telecom applications.

In effect, each UML profile creates a customised Domain-Specific Language (DSL) for modelling concepts in that domain. However, because each language is strongly tied to the well-understood UML syntax and semantics, and defined using UML's standard extension mechanism, it's easier to learn than a language designed from scratch, and can be used with existing and well-supported UML and MDA tools.

Although UML and its profiles are the most widely-used language for MDA, using UML is not actually an MDA requirement; completely un-UML-like MOF-based modelling languages can be defined and used with MDA. One recent example is Semantics of Business Vocabulary and Business Rules (SBVR), a text-based language for representing business rules. Work is also underway to provide a MOF foundation for Business Process Modelling Notation (BPMN™), a popular flowchart-like syntax for creating business process diagrams that represent the activities of a business process and the flow of control that defines the order in which they are performed.

5 Specifications Which Have Been Developed Using MDA

Over the last seven years OMG has created numerous specifications in a number of vertical domains using the MDA approach. A few representative examples will give the flavour of the variety of problems addressed:

- *Microarray and Gene Experiment Object Model*. A Platform-Independent Model for the representation of life-science gene expression data and relevant annotations, along with a standard mapping onto an XML Document Type Definition (DTD) for representing and exchanging this data using XML [5].
- *Product Lifecycle Management (PLM) Services*. This specification defines a PIM for standardised services for use in managing and representing the different configurations and versions a product may be sold under over its lifetime. The specification includes a PIM and a PSM for WSDL/SOAP [6].
- *PIM and PSM for Software Radio Components*. "Software Radio" is a generic term for radio receivers and transmitters where some or all of the signal processing is performed by software running on general-purpose processors, specialised Digital Signal Processors (DSP), or exotic devices like Field Programmable Gate Arrays (FPGAs). This five-volume specification provides both Domain-Specific Languages (defined as UML profiles) for designing Software Radios, and PIMs standardising parts of software radio designs. Mappings of these PIMs onto the PSMs for the industry-standard CORBA-based Software Communication Architecture (SCA) are also provided [7].
- *Application Management and System Monitoring for Combat Management Systems*. This specification addresses the problem of centralised management of CMS applications running on the wide variety of hardware and software platforms found on modern warships. It includes a PIM and PSMs for several widely-used platforms including CORBA, DMTF CIM Managed Object Format and Data Distribution Service (DDS™) middleware, as well as defining a PSM for exchange of management data using XML [8].

OMG's web site has a full list of specifications developed using MDA [9].

6 MDA and Software Development

One important difference between creating interoperability specifications and designing software is that the latter almost always involves modifying and interfacing with existing application code. Although this is often dismissively termed "legacy integration", as though it involved working with a few quaint leftovers from a former age, studies over the last 20 years have repeatedly shown that IT users spend far more effort on modifying existing software than on deploying "new" applications [10] [11] [12]. The original cost of acquiring an application (whether purchased or developed in-house) is often only 10-20% of its Total Cost of Ownership (TCO) when software lifetimes are measured in decades.

If using MDA for software development is going to help achieve a meaningful reduction in TCO, it clearly has to address the issue of maintaining and updating existing software, since this can account for up to 90% of software's true cost.

One way that MDA reduces TCO is by creating new application code with fewer bugs. Higher-quality code results in less effort later being spent on problem diagnosis and remedies, making it easier to adapt the code to new business needs, and lowering the costs to train and support users of the system. An informal side-by-side study in 2006 comparing MDA with traditional hand-coding for a new commercial billing application showed that MDA techniques produced almost three times as many lines of code per dollar spent, but with less than one third the defect rate discovered during testing (1.1 defects per thousand lines of code

for MDA, 4.1 for traditional coding) [13]. Although the lowered coding costs are impressive, the higher code quality will have a much greater impact on the system's TCO over its life.

Having models as a product of the development process also helps lower the costs of making subsequent modifications to the system. Maintainers working on existing applications typically spend more than half of their time simply trying to understand how the code works before they can begin to modify it [14]. With MDA, the design models are one of the products of application development, along with the code itself, so maintenance involves modifying the models and regenerating the corresponding parts of the code. The savings in time spent understanding and then modifying the code can be substantial. In early 2003 a side-by-side laboratory study of maintenance of MDA-based and non-MDA-based J2EE applications found that MDA increased maintainers' productivity by 37% compared to traditional code-based maintenance [15].

MDA's architects have also recognised that not all applications will have been developed this way, so there will be times when a team equipped with MDA tools and skills is faced with modifying an application for which no MDA model exists. It is therefore essential that there's some way of recovering design information from existing software, even where original designs have been lost (or even never existed in the first place). Acknowledging this, OMG started work on "Architecture Driven Modernisation" (ADM) standards in 2003, two years after the MDA initiative began.

The first product of the ADM effort is the Knowledge Discovery Metamodel (KDM) specification, which provides standard metamodels for documenting and formally representing existing software assets and their operational environments with MOF. KDM defines a common vocabulary of knowledge related to software engineering artefacts, regardless of the implementation programming language and runtime platform (a checklist of items that a software mining tool should discover and a software analysis tool can use. KDM's common MOF models and interchange format provide an integration layer between the syntax-specific parsers used to extract information from raw source code and the analysis and transformation tools which process abstract program structure information, thus allowing export and import of data currently contained within individual software modernisation tools). In this way KDM can provide both a common platform to help integrate diverse software modernisation tools, and also provide the basis for bringing knowledge about existing software assets into the MDA software creation process [16].

Independently of MDA, the KDM specification is also finding application in the Software Assurance field, to help analyse existing software to detect security vulnerabilities and other ways in which it might behave outside its required specification. As with software modernisation, many tools are likely to be involved, each producing a portion of the required knowledge about the software assets. KDM is also being used in the Software Assurance field as a standard way of representing knowledge about software collected via a variety of different tools.

Creating tools for recovering design knowledge from existing software is a challenging problem, and OMG's work on standards in this area will continue for some time; for instance, a forthcoming specification will standardise a metamodel for Abstract Syntax Trees, to facilitate the analysis, visualization and refactoring of application code below the procedural level supported by KDM. However, the work in this area is already yielding benefits in helping modify and update existing application code, rather than merely encapsulating it unchanged, and have new applications communicate with it at arms' length, as too often happens today.

7 The Future

As business users become increasingly dependent on Information Technology to deliver products and services, so problems caused by the inflexibility of IT systems becomes ever more pressing. Rather than adapting to the changing needs of the businesses they supposedly serve, IT systems' capabilities increasingly dictate business policy. Traditional software engineering techniques demand stable, well-defined requirements and long timescales for creating systems tailored to users' needs, yet the business environment is changing with increasing speed. All the while, an ever-growing deployed software base is accumulating post-design modifications that distort its structure to the point of "software death", where any further modification (whether to fix a bug or introduce a new feature) in turn introduces a new bug.

The convergence of MDA with Business Process Management (BPM) and Service Oriented Architecture (SOA) offers a road-map for organisations seeking to escape the straightjacket of software inflexibility.

BPM is an umbrella term for the techniques of identifying, documenting and managing the complete end-to-end processes an organisation uses to perform an individual task, especially where this involves the cooperation of multiple individuals, departments or separate organisations. The dozens or hundreds of processes within an organisation typically have both human and IT components, and many of the individual activities within any one process are also used in conjunction with other activities in other processes. Just as different organisations have different levels of maturity in their software production processes, so business organisations have different levels of business process maturity. OMG's Business Process Maturity Model (BPMM) [17] helps organisations discover and improve the level of precision with which they understand their own processes. Given well-understood processes, precise yet easily-learned visual notations like SBVR and BPMN can be used to document and communicate them between business stakeholders, process participants and the engineers building software to support them.

Service-Oriented Architecture is one of the foundations

on which OMG's technical architecture was built almost 20 years ago, but it has now gained new prominence as a method of designing, deploying and managing the individual activities that make up a business process. At the software level, use of SOA entails building components with well-defined meta-data that defines both the information they require from each other and the services that they provide, allowing tools to orchestrate the late binding of SOA services into new processes as the needs of the organisation change. MDA provides the vital bridge between BPM design and SOA infrastructure, allowing process models captured via MOF-based languages like SBVR, BPMN or UML activity diagrams to be translated into that orchestration code.

The convergence of model-driven software development, service orientation and better techniques for documenting and improving business processes holds out the promise of rapid, accurate development of software that serves, rather than dictates, software users' goals.

8 Conclusion

Put in a historical context, MDA can be seen as the most recent step in the progressive development of better and more powerful tools for writing software over the 60 year history of electronic data processing. The first programmers write their applications directly in machine code, entering the bit patterns for instructions from memory and calculating branch offsets and index register settings by hand. Assemblers moved programmers one level of abstraction away from the raw machine, then 3rd Generation Languages (3GLs) and 4th Generation Languages (4GLs) each added another level of tooling between the user and the raw machine, providing abstractions that are progressively closer to the concepts used by the ultimate user of the IT system. MDA continues this trend to better tools and increasing abstraction. There has always been a fierce rearguard action (clinging to the efficiency argument) to any increase in the level of abstraction, but the flexibility, reduced complexity and increased productivity of the abstractions have always won through in the end [18]. MDA is proving to be no exception.

References

[1] Gartner Group. "Middleware: what end users are buying and why", February 1999.

[2] Sun Microsystems. "Java Transaction Service 1.0 Specification", <http://java.sun.com/products/jts/>, 1999.

[3] OMG. "Object Transaction Service 1.2.1", <http://doc.omg.org/formal/01-11-03>, 2001.

[4] A. Zeichick. " UML Adoption Making Strong Progress", SD Times, 15th August 2004.

[5] OMG. " Gene Expression Specification 1.1", <http://doc.omg.org/formal/03-10-01>, October 2003.

[6] OMG. "Product Lifecycle Management Services 1.0.1", <http://doc.omg.org/formal/06-04-03>, April 2006.

[7] OMG. "PIM and PSM for Software Radio Components Specification 1.0", <http://doc.omg.org/formal/07-03-01>, March 2007.

[8] OMG. "Application Management and System Monitoring for CMS Systems Beta 2 specification", <http://doc.omg.org/dtc/07-05-02>, May 2007.

[9] OMG. Specification Catalogue, <http://www.omg.org/technology/documents/spec_catalog.htm>.

[10] L. Erlikh. "Leveraging Legacy System dollars for E-business", IEEE IT Pro, May/June 2000.

[11] A. Eastwood. "Firm Fires Shots at Legacy Systems", The Standish Group, 1993.

[12] J. Moad. "Maintaining the competitive edge", Datamation 61-62, 64, 66., 1990.

[13] Steve Hudson. Private communication, 2006.

[14] B.P. Lientz, E. Swanson. "Problems in application software maintenance", Communications of the ACM 24 (11), 763-769, 1981.

[15] The Middleware Company. "Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach - Maintainability Analysis", January 2004.

[16] OMG. " Architecture-Driven Modernization: Knowledge Discovery Meta-Model 1.0 beta3", <http://doc.omg.org/ptc/2007-03-15>, March 2007.

[17] OMG. "Business Process Maturity Model 1.0 beta1", <http://doc.omg.org/dtc/2007-07-02>, July 2007.

[18] D. Otway. "Abstract & Automate", Architecture Projects Management Ltd, <http://www.ansa.co.uk/ANSATech/94/Primary/102001.pdf>, May 1994.