Model Driven Architecture An Overview

Brent Seidel
Arizona State University

Abstract—It has long been the dream of managers to eliminate computer programmers. Model Driven Architecture is the latest in a series of attempts to accomplish this. We review flowcharting, mainly for historical interest, UML, and BPMN for use in Model Driven Architecture. Based on the pros and cons of these approaches, we make a cautious recommendation for the use of Model Driven Architecture in developing new enterprise applications.

Index Terms—Model Driven Architecture, Object Management Group, Uniform Modeling Language, Business Process Modeling Notation

1 Introduction

Software development is hard work. In the early days, major advances were made with the introduction of high level programming languages and structured programming. Flowcharting was adopted for software and used for a while, but has largely been abandoned. Still, the lure of a visual representation of software has remained strong. This is in spite of Fred Brook's observation that "...software is very difficult to visualize."[1]. More recently, the idea has been promoted in IEE 598, *Introduction to Systems Engineering*[2].

This is not to say that diagrams and models aren't useful and that progress hasn't been made. There have been advances in programming languages and methodologies. Some of these have proved to be more successful than others. Object oriented programming is now common. Unified Modeling Language (UML), and other modeling notations, are useful for analysis, design, and documentation of systems. On the interoperability front, the Object Management Group (OMG) was formed to develop and promote standards for enterprise integration[3].

There is tension between the developers of platforms and the developers of application software. The platform developers, such as Microsoft and Apple, want to encourage the application developers to focus solely on their platform. The application developers on the other hand want their software to work on as many platforms as possible. Java is perhaps the best known attempt to enable platform independent software development.

One of the early interoperability standards developed by the OMG was the Common Object Request Broker: Architecture and Specifications (CORBA)[4]. The first version was released in October of 1991 and the standard has been developed since then.

The OMG views its Model Driven Architecture (MDA) initiative as a major evolutionary step in the definition of interoperability standards[5]. It moves the focus from CORBA based component interfaces to formal systems models.

Agrawal, et al[6] point out that the this next step is to move from just capturing requirements to including the design and implementation in the model. The progression has been from using the model purely as documentation to generating software skeletons that need to be filled in by hand to generating the complete application from the model. At this point, the model becomes the source code for the application.

The goal of MDA is to be able to draw the model for an application and either have it directly execute or generate the complete software for the application. This has been achieved for some problem domains[7].

With advancing development technologies and methodologies, the job of the computer programmer has moved to higher and higher levels of abstractions. There is no evidence to suggest that this trend will stop at MDA. It is also not always obvious which ideas will prove effective and which will catch on.

2 Examples

In this section, flowcharts, UML, and BPMN are covered. Flowcharting is primarily of historical interested as an early attempt to graphically model software. UML takes an objected oriented approach to software modeling. BPMN takes a process oriented approach to software modeling.

Tools do exist for methods covered. These tools range from flowchart templates used for pencil and paper to automated design tools that ensure that various design rules are met.

2.1 Flowcharts

PROGRAMMERS of a certain age will remember flowcharts[1]. These were always suppose to be drawn before writing code. Brooks[1] states that they were essentially useless as a design tool, typically being drawn after coding, not before.

It is perhaps telling that a quick search turned up FLOWTRACE[8], a program from 1966 that draws flowcharts for programs in "almost any" programming language. This suggests that in practice flowcharting was used more for documentation than design purposes.

A flowchart model consists of several elements. The most common are: Ovals indicating the start and end points of control flow, Boxes containing sequences of actions, and Diamonds indicating decisions. There are some less used symbols that are used to indicate subroutine calls, access to mass storage, printouts, and others. All of these elements are connected by lines indicating the flow of control.

One of the drawbacks of flowcharting is that it was developed before structured programming became commonplace. Thus, one could often get a visual representation of "spaghetti code". While flowcharting is not particularly common now, its ideas live on in UML Activity Diagrams and BPMN[9].

2.2 UML

UML has been extensively covered in CSE 598, *Software Analysis and Design* so a detailed introduction will be omitted.

UML is a complex system consisting of a number of different diagrams[10]. Some of these diagrams are more important for modeling the actual system. These would include Class, Object, and Activity diagrams. Other diagrams are more useful for understanding how the system works and interact with its environment.

Stephen J. Mellor identifies four ways that UML (and other modeling notations) can be used in software development[7]:

1) UML can be used as a communication tool when discussing the design abstractions.
2) It can be used as a blueprint to specify the software.
3) Code can be added to the model to make the model directly executable.
4) Models can contain an "action language". This is similar to the directly executable models, but the model can also be translated to any supported implementation.

The difference between the first two and the second two ways is that in the first two ways, UML is being used to communicate ideas with other people. In the second two ways, UML is being used to communicate with computers. When communicating with people, often details are omitted when they obscure the central idea. When communicating with a computer, especially when trying to create an executable application, the details are essential. Basically what has happened is that the modeling language has turned into a graphical programming language and has to be used with the same rigor as other programming languages.

Even though Mellor[7] has an optimistic view of what is possible, he notes that the problem is not yet solved. In particular, he says, there is a need for a common "action language". Without this, models developed using one UML tool may not execute on a different UML tool.

Aqrawal, et al.[6] lists three levels of tools needed for the model based development process:

1) Tools used to create the model.
2) Tools used to transform the model to some executable form.
3) A platform where the executable form of the model is executed.

They go on to suggest that there is still room for improvement in the model transformation tools. High quality code generators tend to be focused on a particular domain.

Should the standard UML not provide the needed semantics, there is a defined mechanism[10] for extending UML using stereotypes and tagged values.

2.3 BPMN

BUSINESS Process Modeling Notation (BPMN) was originally developed by the Business Process Management Initiative[9] and is now also managed by the OMG. While UML takes an object-oriented approach, BPMN takes a process oriented approach to modeling[11].

BPMN supports three diagrams[12]:

- The process diagrams
- The collaboration diagrams
- The conversation diagrams

The BPMN diagrams are based on flowcharting techniques[9] with boxes, diamonds, etc. There are four catagories of elements:

2.3.1 Flow Objects

In contrast with UML, BPMN has just three core elements:

- Event
- Activity
- Gateway

2.3.2 Connecting Objects

There are three connecting objects that are used to connect the flow object in a diagram:

- Sequential Flow
- Message Flow
- Association

1. I was given a flowcharting template in the early 80s. It did not see much use.

2.3.3 Swimlanes

Swimlanes are used to organize activities to illustrate different functional capabilities or responsibilities. They consist of two elements: the pool and the lane, which is a partition within the pool.

2.3.4 Artifacts

Artifacts are used for extending the basic notation. There are three basic types defined and more can be added as needed:

- Data Object
- Group
- Annotation

The BPMN standard[12] also defines what extensions to the diagrams are permitted. Basically new elements are permitted, but the existing definitions may not change.

In contrast to UML which still needs a common "action language"[7], BPMN includes Business Process Execution Language (BPEL) as part of the standard[12]. However, both Vignéras[13] and Swenson[14] argue that BPEL isn't particularly useful because of limitations in the language itself and because BPMN models can be directly executed without needing BPEL.

The language problem is that since BPMN does not enforce structured programming structures and BPEL is a structured programming language, it is possible to create BPMN models that do not easily translate to BPEL. This language problem would also apply to UML since modern implementation languages are all structured.

2.4 Pros and Cons

OUYANG, et al.[15] point out a problem with both the UML Activity Diagrams and BPMN. As mentioned earlier, flowcharting was developed before structured programming. Neither BPMN or UML restrict one from using unconstrained control transfers. This makes it difficult to map the resulting diagrams into a structured programming language.

Mellor[7] notes success in generating software for embedded systems. Specifically, after ten years of development, his company's flagship model compiler can produce code faster and smaller than code produced by hand. This is because of the refinement of rules used to generate code. Shunk[2] suggests that it might be possible to replace large numbers of programmers with a few business analysts using BPMN[2].

Seifert, et al.[16] discuss an often overlooked part of software development, that is, software maintenance. Most software will be modified over its lifetime. Sometimes these changes are due to features being added or dropped. Sometimes they are due to interfaces being added or dropped. Sometimes they are due to changes in other support software. MDA claims to address this, however it doesn't offer much support for changes in the tool chain.

Aqrawal, et al.[6] say that the weak point of MDA is still in the model transformation step. There is still a need for tools that can generate high-quality products.

UML has thirteen possible diagrams[17], while BPMN has only three[12]. It is not required that one use all of the possible diagrams. One could argue that UML is more cumbersome and BPMN is more streamlined or that UML offers richer opportunities for modeling and that BPMN is simpler. Which is preferable would depend on the specific situation.

As mentioned above, both UML and BPMN offer extension mechanisms. When one is scribbling diagrams on the back of a napkin, this isn't a big issue. As long as it is understood by your colleagues, anything goes. The problem comes when one is developing executable models. If vendor A offers a set of useful extensions, your model may not work using vendor B's tools.

BPEL[12] is better standardize than the UML action language[7]. This suggests that it may be easier to move models between tools using BPMN. However, keep in mind the previously mentioned drawbacks of BPEL.

UML takes an object-oriented approach, while BPMN takes a process oriented approach to modeling[11]. Which is better would depend on your particular situation.

3 Recommendations

FOR the development of large enterprise applications (eg. a replacement for Blackboard), I would offer a qualified endorsement for the use of MDA. The following items should be considered:

- A tool does not have to be perfect in order to be useful. A tool that only generates skeletons for the classes still does some work that the programmers won't have to do.
- The use of MDA will not allow large numbers of programmers to be replaced by a few business analysts.
- Consider how applicable the specific tool is to your problem domain. A tool that is optimized for creating state machines for embedded systems will probably not do a good job for a large enterprise application, and vice versa.
- Consider how flexible the tool is. Can you generate your own rules for code generation, or are you tied to vendor specified rules?
- Consider the lifecycle of your application. It is important to remember that software development tools and methodologies themselves are continuously being developed. Will the tool that you are using continue to be supported? This may be an

2. I argued in personal communication with Shunk that if the model is executable then the business analysts are really programmers and this should be seen as improving the productivity of programmers rather than replacing them.

argument for using an open source tool[3]. On the other hand, if you are only going to use the tool for the initial program creation and then maintain the source code manually, this may not be an issue.
- Consider the portability of your model. How easy will it be move your model from one tool to another? Are you going to be tied to a specific tool and vendor? If the tool gets discontinued, you may be left scrambling for support.
- Consider the languages and frameworks that the tool supports. How easy is it to support new ones? Are they popular, widely available ones, or are they proprietary to a specific vendor?
- Consider what you and your staff have experience with. If your staff has already been using UML for software analysis and design, then a UML based MDA tool would be a natural transition. If you have just used ad hoc techniques, you may be better off training your staff in software development methodologies.
- Since the modeling tools do not enforce structured programming and most modern programming languages are structured, you will need to be careful to construct models that are structured.
- It may be worthwhile to consider trying one, or more, small pilot projects first to gain some in-house experience with MDA before attempting a large project.

If approached cautiously and with careful consideration, one should be able to successfully complete a project using MDA. The important thing to keep in mind is that software development is hard and that "There is no magic bullet"[1].

APPENDIX

Acronyms and Abbreviations

BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
CORBA	Common Object Request Broker
MDA	Model Driven Architecture
OMG	Object Management Group
UML	Unified Modeling Language

REFERENCES

[1] F. P. Brooks, *The Mythical Man-Month (anniversary ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] D. L. Shunk, "Bpmn overview," IEE 598 Introduction to Systems Engineering - Class Notes, 2008. [Online]. Available: SEModule18New-BPMNOverview.ppt

[3] (2011) About the object management group. [Online]. Available: http://www.omg.org/gettingstarted/gettingstartedindex.htm

[4] OMG. (2011) History of corba. [Online]. Available: http://www.omg.org/gettingstarted/history_of_corba.htm

[5] J. D. Poole, "Model-driven architecture: Vision, standards and emerging technologies," Workshop on Metamodeling and Adaptive Object Models, April 2001. [Online]. Available: http://www.omg.org/mda/mda_files/Model-Driven_Architecture.pdf

[6] A. Agrawal, T. Levendovszky, J. Sprinkle, F. Shi, and G. Karsai, "Generative programming via graph transformations in the model-driven architecture," Institute for Software-Integrated Systems, Tech. Rep., 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.4824&rep=rep1&type=pdf

[7] S. J. Mellor, "Embedded systems in uml," Object Management Group, Tech. Rep., 2007. [Online]. Available: http://www.omg.org/news/whitepapers/050307_Embedded_Systems_in_UML_by_S_Mellor.pdf

[8] P. M. Sherman, "Flowtrace, a computer program for flowcharting programs," *Communications of the ACM*, vol. 9, no. 12, December 1966.

[9] S. A. White, "Introduction to bpmn," IBM Software Group, Tech. Rep., 2004. [Online]. Available: http://www.zurich.ibm.com/~olz/teaching/ETH2011/White-BPMN-Intro.pdf

[10] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, ser. Object Technology, Addison-Wesley, Ed. Addison-Wesley, 1999.

[11] Faq. [Online]. Available: http://www.bpmn.org/Documents/FAQ.htm

[12] O. M. Group, *Business Process Model and Notation (BPMN)*, Object Management Group, 2010.

[13] P. Vignéras. Why bpel is not the holy grail for bpmn. [Online]. Available: http://www.infoq.com/articles/bpelbpm

[14] K. Swenson. (2011) Bpel: Who needs it anyway? [Online]. Available: http://www.bpm.com/bpel-who-needs-it.html

[15] C. Ouyang, M. Dumas, S. Bruetel, and A. H. ter Hofstede, "Translating standard process models to bpel," *Lecture Notes in Computer Science*, vol. 4001/2006, pp. 417–432, 2006.

[16] T. Seifert, G. Beneken, and N. Baehr, "Engineering long-lived applications using mda," in *Intl. Conference on Software Engineering and Applications*, November 2004, pp. 241–246.

[17] F. Calliss, "Object-oriented design analysis and design and introduction," CSE 598 Software Analysis and Design - Class Notes, February 2008.

3. I have just recently discovered an open source MDA tool called AndroMDA, http://www.andromda.org/docs/index.html. I hope to get some time to evaluate it this summer. However, this shows that such tool do exist.

Brent Seidel is an online Masters of Engineering in Systems Engineering student at Arizona State University. He as worked at Boeing and at Honeywell on a variety of aircraft systems.