Roles in the MDA Process

MDA will make developers more productive, not redundant

Stephen J. Mellor
steve@projtech.com
www.projtech.com

Andrew Watson
andrew@.omg.or
www.omg.org

Uptake of the Object Management group’s Model-Driven Architecture (MDA) initiative is dependent not only on its technical capabilities, but also the acceptance of the approach in the developer community. This paper examines the roles in today’s software development processes and show how those roles change and improve as MDA is applied.

The MDA Process

At the heart of MDA is an approach to design based on treating it as a product, rather than a process. Non-MDA approaches treat design as a process in which developers apply their expertise to handcraft elegant solutions from often ill-defined problems. The result is running code. The design is not captured explicitly, but instead smeared across the code as a set of results of a skilled thinking process.

In an MDA approach, promulgated by OMG and based on the work of Shlaer and Mellor, and others, the design is itself a product that is expressed as a set of formal mapping rules. Examples of such rules are "Turn an application class into a class in the implementation with a container for its instances" and "Turn each transition on a statechart diagram into a tuple in a linked list defining the transition and the procedure to call to execute actions". These rules are the design, and once they are executed against an application model that captures a problem, the result is (more or less) running code.

In MDA the application model is generalized as a platform-independent model\(^1\), or PIM, usually visualized using standard UML, and the framework for formal transformation rules is currently being standardized as the QVT (Query, Views, and Transforms) specification. The result of the transformation can either be another, platform-specific, model (PSM), or components of the target

---

\(^1\) Platform-independence is a relative, rather than absolute, concept. CORBA was originally conceived fifteen years ago as a platform-independent system because it was independent of the dominant programming languages, operating systems, and database technologies of the day. In this paper, however, we view CORBA as a platform, and platform-independent means independent of the dominant middleware, programming languages, operating systems, database technologies, and data serialization technologies (such as XML) of today.
application, such as database schemas, deployment scripts, test scripts, or indeed executable code. In the latter case, the PIM must be behaviorally complete, defining business logic in terms of abstract actions.

Since a single rule, such as “Turn an application class into a class in the implementation with a container for its instances,” is not always appropriate, there is then the need to determine which rule to apply in a given circumstance dictated by performance, sizes of data sets, or other application-specific criteria. The result of making this decision is captured as a mark on, but separate from, the PIM.

How then do today’s developers fit into the MDA approach?

Today’s Roles in the MDA Process

Requirements Analysts

Capturing the results of the requirements analysis in machine-readable UML models, which can be used to test and verify models built later in the development process, reduces the chances that the results of the requirements analysis will be misinterpreted. The task of gathering and resolving requirements does not change; it’s the same people-driven task. UML models such as activity diagrams and use case diagrams can certainly help in communicating and visualizing the requirements, but their creation cannot be automated.

Analyst/Designers

The formal expression of those requirements in terms of class diagrams, state chart diagrams and method actions, however, is a more formal affair. Because these models are to be transformed into other models or code, a premium is placed on the accuracy of those input models so as to reduce downstream modification. This is nothing to be afraid of; only the business logic needs to be expressed. Decisions about data structure, distribution, concurrency, and the like are not of concern at this level.

If the analyst/designer’s tool uses an executable UML, there is the opportunity to demonstrate the behavior of his models to the customer, and hence get faster feedback and better model accuracy.

Architects

Architects, who decide on the overall structure of the system, will continue to do so. In an MDA process, this work will involve selecting the models and mappings between them. As the market matures, architects will find existing models and mappings for reuse. Architects will then apply their expertise in the selection process and in tuning mappings to improve system performance.

Analyst/programmers

Skilled analyst/programmers also continue to perform in much the same manner as before, but with two differences. First the language in which they express their abstractions will be that defined by QVT, rather than Java C++, or Smalltalk. Second, and crucially, the abstractions these developers create have higher leverage. Instead of creating (say) a “list for a set of accounts”, they would create the rule for building a “list for any class that has the same pattern
of access as an account”. That is, their job becomes creating and tuning the code generation patterns used by the MDA tool. In this way the analyst/programmer’s vital platform expertise is used to much better effect than simply writing code by hand: the MDA toolset acts as a skills amplifier—the mapping rules propagate the design across the entire system, increasing the leverage of the architect from point design to system-wide design.

Today few MDA projects claim to generate 100% of an application’s executable code. Many tools generate stereotyped glue code that binds different parts of the application together, leaving some of the more interesting algorithms and business logic to be written by hand. Programmers working with MDA tools find that they can write this code as part of a code generation template, giving them more time to work on the more interesting individual pieces of logic that the code generator can’t handle. As a consequence, the less advanced MDA tools can generate at least 50% of the entire system, and the more sophisticated tools can do much better, up to 100% of the modeled system in certain circumstances.

Testers

Testers become more productive by using tools to generate test scripts from models, which can then test the hand-coded and machine-generated parts of the production system directly against those models. Writing test scripts thus becomes much less tedious and more effective. In order to exercise all the execution paths through an application, a test script often combines a few different sets of simple input parameters in a large number of combinations, and then compares the output from the application with the expected one. By working with a test-generation tool that generates test code from a model, a tester can rapidly produce the huge number of individual tests needed to comprehensively test a large application, but without the tedium of writing and re-writing slightly-different code fragments over and over again.

Maintainers

Maintenance costs make up more than half of the total cost of ownership of any long-lived application. This is partly because of the detective work maintainers must do to understand the behavior of an application when the design documentation is out-of-date or missing. Because a machine-readable, accurate design used to build the application is one of the deliverables from the MDA process, maintaining an MDA application is a less frustrating and more productive activity than maintaining hand-written code. Maintainers can maintain the models and the design rules, not the code. When the application behavior needs modification the PIM is modified. If the target platform is changed, the mappings are modified or replaced with mappings to a wholly different platform. The result is potentially a longer life for the application and a less frustrating job for the maintainers.

Customers

And of course the customer gets better systems.

Evolution of Roles in MDA

The knock-on advantage of MDA is the ability to reuse models and design rules captured in previous projects and available commercially. As a consequence, the role of the architect will
evolve to include more strongly a search for the best available models, platforms expressed as models and mappings between them.

In turn, this will require expertise in characterizing the system properties of models, and its obverse, characterizing the required system properties. For example you may find a user interface model that has precisely the functionality you require, but the manner in which it integrates into the remainder of the system makes it unusable. Understanding the system properties of the user interface early can avoid error. Obversely, characterizing non-behavioral requirements early will facilitate finding the right model.

And, in turn, this will encourage a separation between requirements-gathering-from-people and requirements-gathering-from-other-applications. Getting the requirements right for a banking application is an example of the former; getting the requirements right for a user interface in the context of a banking application is an example of the latter.

As executable models become more popular, abstraction capability will become an identified skill. Sometimes the person best able to talk to a customer or characterize an application is not the best person to abstract good solutions to these problems.

A key element of MDA is the use of marks to performance-tune results of the translation. An application engineer can develop the required marks and can carry out the experimentation required to find best performance. This role requires knowledge of the patterns of use of the application as well as knowledge of the performance properties of the target implementation. We expect to see some analyst/programmers migrate in this direction.

Conclusion

Fear of the unknown is a powerful inhibitor of adoption of a new technology. However, moving to MDA offers architects, developers, testers, maintainers and others involved in creating large applications a chance to use their skills in a more productive way and eliminates some of the more frustrating aspects of their jobs. As you can see, the today’s developers can easily take on the tasks required in MDA, and as the technology matures and experience grows, we would expect to see an evolution in the specialization of skills: a sure sign of a maturing engineering discipline.