Model Driven Architecture: A Research Review for Information Systems Educators Teaching Software Development

Samuel S. Conn
sconn@regis.edu
Regis University
Denver, Colorado 80021

Lynne Forrester
lynne_forrester@msn.com
University of Denver
Denver, Colorado 80023

Abstract

The increasing complexity of business systems, the accelerating pace of technological change, and the highly competitive business environment are overwhelming software development methodologies that have stayed essentially the same for the last fifty years. Model Driven Architecture (MDA) is a current initiative by the Object Management Group that represents a major evolution in the way software is developed. There is growing consensus by the information systems community on the fundamental principals of MDA, but some critical elements are missing in the areas of transformation and system-behavior modeling. Agreement on standards and approaches in these areas will take some time, and substantial work remains before MDA can replace traditional, long-practiced methods and be considered a routine approach to software development. When this happens it has the potential to significantly improve the integration of customers into the software development lifecycle. Traditional development methodologies and the newer agile methods strive to overcome quality and delivery problems by emphasizing customer involvement and by attempting to move system validation activities (e.g. those concerned with confirming that the system will meet the customer's needs) earlier in the development lifecycle. The adoption of an MDA approach will not change the role of customers in the development process or the nature of their activities; it will not change what customers do. However, it can significantly change when customers validate a system's functionality. This paper will investigate these issues through a qualitative study using an interpretivist epistemology, and will form generalized conclusions about MDA.

Keywords: Model driven architecture, MDA, Customer Integration, PIM, PSM, Life Cycle Development

1. INTRODUCTION

The increasing complexity of business systems, the accelerating pace of technological change, and the highly competitive business environment are overwhelming software development methodologies that have stayed essentially the same for the last fifty years: programs are hand-coded with 3rd generation languages such as C++ using informal models and designs (Frankel, 2003; Selic, 2003). Responding to changes and to increasing complexity has become very expensive. Often applications must be extensively re-coded for new technology, and the original knowledge used to develop the application is embedded inside the source code and not readily available or even lost.

Model Driven Architecture (MDA) is a current initiative by the Object Management Group that represents a fundamental evolution in the way software is developed. With MDA, models (which until now have been informal artifacts of design that become obsolete shortly after creation) will be the primary focus of development instead of application code; and according to Frankel (2003) they will become persistent artifacts of development. MDA uses a collection of models, standardized modeling languages, and specialized tools to define system requirements in precise and formal ways so that application code can be automatically generated rather than hand-coded. What characterizes these models and makes them so different is their platform-independence. These models can become persistent organizational assets that capture domain-specific knowledge from both the organizational community and the technical community without reflecting any specific technologies needed to implement their functionality. Through complex and specialized transformation and through code generation, these models can be automatically transformed and merged to create technology-specific models as well as complete applications. As the technology changes, new applications can be regenerated for the new platforms without re-engineering them. As the business requirements change, the models can be changed with full knowledge of the implications, and the applications can be regenerated automatically.

2. INTEGRATING THE CUSTOMER INTO THE DEVELOPMENT LIFE CYCLE

Traditional software development methodologies and the newer Agile Methods strive to overcome quality and delivery problems by emphasizing customer involvement and by attempting to move system validation activities (e.g. those concerned with confirming that the system will meet the customer's needs) earlier in the development lifecycle. Agile methods, such as Extreme Programming (XP), strive to make customers a team member throughout the development process. Some methods even adopt a test-first approach to design. Traditional methods promote spending adequate time early in the development cycle to ensure detailed specifications accurately reflect the user requirements. Sommerville (2004) notes that they will often use prototypes to validate specifications before programming begins. Both methods believe that waiting until after an application has been developed to validate a system's requirements is inefficient, causes significant delays, reduces flexibility, and results in ineffective applications and potentially failed projects.

Because MDA development tools are focused on models, MDA has the potential to bring software design and development processes closer to the realm of the customer. Platform-independent models are easier for customers to understand than are traditional design specifications. Even though they are developed in precise and formal ways, they are written in the language of the customer's domain (and not the programmer's). These models abstract away the extraneous details related to technical implementation so only the essentials are described. MDA may make it possible to integrate customers more fully into agile development cycles and provide ways to move system validation activities to an earlier phase in traditional development cycles. This research explores that potential. After distilling the current literature on MDA, a general overview of the MDA models and processes will be presented. The potential for MDA to improve customer integration into the software development lifecycle is discussed. The final section summarizes the conclusions.

3. REVIEW OF THE LITERATURE

As a new initiative, MDA has generated, somewhat paradoxically, too little and too much published material. Because MDA is comparatively new there are few works that describe the overall vision of MDA with any depth, and because there are still significant disagreements and missing pieces, there are dozens of works addressing specific issues. A few that describe the overall potential and promise of MDA are: a) shorter works by key players in the MDA initiative such as high-level summaries published in more popular literature, on the Internet, or as marketing pieces (e.g., Borland, 2004; Stephenson, 2003; Object Management Group, 2004; Klasse Objecten, 2004a, 2004b, 2004c; Frankel, 2004), b) those that cover the topic only in the first chapters as an introduction to a more specific discussion (e.g., Raistrick, Francis, & Wright, 2004; Warmer & Kleppe, 2003; Arlow & Neustadt, 2004), or c) those that describe the overall process in almost skeletal fashion such as Mellor, Scott, Uhl, and Weise (2004). Only Kleppe, Warmer, and Bast (2003) describe the full vision of MDA with any depth.

Because MDA is relatively new and because definitive works with both breadth and depth are few, there appears to be several different perspectives and disagreements in the MDA community, especially on how it should be approached and developed. However, there are three areas of general consensus in the literature. The first is that MDA methods should be based on underlying consistent standards and syntaxes (Brown, 2004). By definition MDA is based on models; what is important is that there is across-the-board acceptance that these must be based on standards. None of the literature reviewed argued for using modeling languages that were created from scratch, no matter how formal and complete. All seemed to agree that the foundational language should be based whenever possible on the OMG's Unified Modeling Language (UML) and Meta Object Facility (MOF). This does not appear to be because MDA is an OMG initiative, but rather because UML is becoming the modeling language of choice for both software development and system engineering in general (Oliver, 1997).

The second area of agreement is that MDA should enable the separation of concerns. This commonly means business concerns should be modeled and designed separately from technical concerns. According to Sommerville (2004) this has been a grounding principle in software engineering for some time. In MDA, this is envisioned as separate platform-independent models (PIMs) and platform-specific models (PSMs). This approach enables organizational and technical experience as well as successfully used software engineering patterns to be preserved while limiting the influence of implementation methods and technologies (Booch, 2004). There are differences in what platform-independence means, and Raistrick et al. (2004) call for issues to be captured in narrowly focused domains; Frankel (2003) considers independence to be relative to a specific model; and Frankel, Harmon, Mukerji, Odell, Owen, Rivitt, et al. (2003) identify a model called the computation independent model (CIM) that is even more abstract than a PIM. The authors consistently highlight separation as important and argue it should be a fundamental part of MDA. Hubert (2001) describes the overall importance of platform-independence as the separation of two lifecycles that do not belong together. Business-relevant architecture models and implementation technologies change and move with very different forces and timescales. They are obviously related, but coupling them in the wrong way can cause problems. Mellor et al. (2004) describe MDA as a set of methods, standards, and tools that strive to separate an application's functionality from the influences of specific technologies, to decouple these so that last minute implementation decisions can be made.

The third area of agreement is that computer automation to manipulate and transform models and to generate code should be a fundamental part of an MDA process. Rapid changes in technology are overwhelming software development methodologies that have stayed essentially the same for the last fifty years. Both Frankel (2003) and Selic (2003) say that even though some new programming concepts such as structured programming and object orientation have been adopted, developers are still writing the same code by hand (i.e. an IF statement

in C++ is the same as an IF statement in FORTRAN) and that the current practice of millions of developers needing to learn completely new technology every two to three years is not scalable. All the authors cover MDA processes in some depth (e.g., Mellor et al., 2004; Kleppe et al., 2003; Raistrick et al., 2004; Frankel, 2003; Brown, 2004) discuss the increasing level of abstraction in which software is developed, starting with the early invention of machine-code compilers and writing in assembly code and ending with the transition to the more abstract 3rd generation languages used today. They see MDA as the next evolutionary step in the advancement of programming languages (i.e. as the next step up in the level of abstraction).

The agreement on these areas is not surprising as they are fundamental tenets of MDA; researchers and organizations involved with developing MDA approaches and tools have accepted them. At a high level there is consensus. However there is disagreement in the detail. The main issue under study is how to transform systems developed with abstract models into working applications. What is apparent in the literature is that tools for generating code from PSMs are readily available and have been used for years, especially in real-time system development and with earlier CASE technologies. This part of the process is not an issue. What does concern these researchers and organizations is how to transform PIMs into PSMs.

The majority of the missing pieces, key differences, research and development efforts, and scholarly discussions revolve around this issue. Most of the literature on MDA, while generally discussing MDA in the first few chapters, is focused on discussing the specifics of transformation (e.g. Starr, 2002; Mellor & Balcer, 2002; Raistrick et al., 2004; Warmer et al., 2003; Frankel, 2003). The OMG website lists other literature that was not reviewed but which does specifically focus on Executable UML and Object Constraint Language (OCL) as potential meta-models/modeling languages for models and transformations. Much of the current scholarly research is focused on this issue as well. There are multiple sources including discussions on specific meta-models (e.g., Haustein & Pleumann, 2004; Sunyé, Pennaneac'h, Ho, Le Geunnec, & Jézéquel, 2001; Akehurst, Linington, & Patrascoiu, 2003; Akehurst, 2004; Cariou, Marvie, Seinturier, & Duchien, 2004; Sendall & Kozaczynski, 2003; Baresi, Heckel, Thöne, & Varró, 2003), evaluations of potential transformation methods (e.g., Bettin, 2003; Czarnecki & Helsen, 2003; Küster, Sendall, & Wahler, 2004), and evaluations of proposals for a transformation standard that the OMG has received (e.g., Gardner, Griffin, Koehler, & Hauser, 2003).

There is also disagreement on the general approaches to transformation. Czarnecki et al. (2003) have evaluated seven of them. However, as Welsh (2004) describes the situation, it has come down to two approaches that are fundamentally different. Consider those who agree with Raistrick et al., (2004) and Mellor et al., (2002), as well as those with Kennedy Carter (n.d.), as translationists who create PIMs so detailed they can be executed without first being translated into PSMs or code. Others such as Kleppe and Warmer (2000), Warmer et al. (2003), and Frankel (2003) are elaborationists who start with PIMs and progressively add refinements to produce PSMs.

4. MDA MODELS AND PROCESSES

MDA uses three sets of models: platform-independent models (PIMs), platform-specific models (PSMs), and transformation models (TMs). Platform-independent models (PIMs) capture domain-specific knowledge from both organizational environments as well as technical environments. These models are independent of the actual technologies needed to implement their functionality. For example, within the organizational domain the entities, relationships and processes required for managing the withdrawal and transfer of funds in a banking application can be described without any consideration that account access might be remote or that CORBA and Oracle may be the middleware and data management technologies. Only elements such as customers, accounts, balances, and activities are described. This does not mean, however, that platform-independent models are not technical in na-

ture. Technical architectures can also be described without consideration of the actual technology needed to implement them.

Platform-specific models (PSMs) provide descriptions of structure and functionality implemented by a specific technology. For the example above, the model of the n-tier architecture becomes platform-specific when it is restructured to show its elements, components, and functionality formulated with the constructs required by Enterprise Java Beans (EJB) technology. Platform-independence and platform-specific are relative concepts. Both can be defined at different layers of abstraction in a hierarchical way. In some respects, PIMs and PSMs fall along a continuous scale. As Frankel (2003) observes, it is important to specifically identify from what a model is consider to be independent.

Transformation is the process of transforming PIMs and weaving them together to create PSMs that are precise enough, formal enough, and detailed enough to automatically generate code. Transformation models (TMs) are used to guide the transformation process; they describe by using mapping functions and marks how a PIM can be transformed into another PIM or into a PSM. Mapping functions are "a collection of rules or algorithms that can be used to convert one or more input or source models into an output or target model. Marks are used by the mapping functions to provide specific details and options for choosing between rules. The transformation itself involves selecting the elements in the source models to manipulate, determining which rules in the transformation models apply to which element, and then applying those rules to generate a resulting target model. A model compiler then weaves these resulting models together and generates program code that can use traditional code compilers (Mellor et al., 2004).

5. MODELING LANGUAGES AND META-MODELS

The MDA modeling environment has four levels. At the lowest level, called M0, are object instances. These are the actual objects with attribute values behaving or being manipulated within a system. In the level above, called M1, reside the models of this system. These are the static class models that describe attributes, methods, and associations. These are the use case diagrams, action diagrams, and sequence diagrams that describe the behaviors of a system. These are the PIMs and PSMs of the MDA process.

All of the modeling constructs used in these M1 models, for example a rectangle that represents an object class or an open triangle at the end of an association line that represents an "IS-A" relationship, have very specific semantics. These semantics are defined by the UML standard. UML is a meta-model; it describes models much like meta-data describes data. It is a modeling language (as its name reflects) used to build models. The UML standard is the third level of the model hierarchy (M2). What is important is that UML itself is a model based on an even more abstract meta-model. This more abstract meta-model is called the MOF; it is a meta-meta-model. It defines the meaning of meta-model constructs. It defines how to build a meta-model, how to build a modeling language that is consistent with standards. The MOF is the top layer of the hierarchy (M3). It is part of MDA because UML is not only not required for MDA it might also be inappropriate for a particular situation. It might be necessary to build a custom modeling language. However, this language must still be precise and formal so a machine can interpret it. It must still be based on standards so the models can be exchanged between and interpreted by different modeling, transformation, and code-generation tools. The MOF provides the guidelines for creating a properly defined meta-model.

To a lesser extent, a subset of UML elements can be customized or extended to create a profile that provides more specific constructs, constructs with very specialized semantics (Mellor et al., 2004). Profiles can reflect the requirements of a specific technology, for example CORBA, which can be used to portray CORBA-specific PSMs. Profiles can also reflect the unique requirements of a particular industry, for example aircraft manufacturing, which can be used to build domain-specific PIMs. The OMG's Common Warehouse Meta-model (CWM) is a standard UML profile. It is a meta-model used specifi-

cally for the design and development of data warehouse models. There are many other specialized profiles being defined and standardized by OMG domain task forces as well as other organizations. Some are targeting technologies such as CORBA and EJB; others are more general such as the Business Semantics of Business Rules (Object Management Group, 2003).

Unlike the meta-models used for building static models, those needed to build transformation models and to model system behavior are currently too limited, too informal, too detailed, not standardized, or nonexistent. There are no standard meta-models that can be used to build complete TMs. There are several proposals. The OMG received eight submissions in 2003 to its proposed MOF 2.0 Query, View, and Transformation (QVT) standard (subsequently narrowed to 5). However Gardner et al. (2003) reviewed these proposals from the perspective of potential tool users and compared them to a set of criteria they had developed. While some proposals fit some of their criteria, none provided a complete solution. All were missing a key piece, and some were severely lacking in places. This proposed QVT standard is still being debated, and it does not appear to be close to finalization. Another potential tool for transformation modeling is the Object Constraint Language (OCL), which is a small but vital part of the UML standard. It has evolved beyond its original intent to note constraints or restrictions on UML models and is now proposed as a fundamental part of any eventual QVT standard. It can be used, for example, to develop a transformation rule that says use the string-to-string conversion process on an attribute in the source model to generate an attribute in the target model. However, this is a declarative language that can indicate when to use the string-to-string transformation process, but it does not define the rule itself. It can indicate that a specific process should occur based on a certain condition or must have a certain condition when finished, but it does not define the behavior itself (Warmer et al., 2003; Frankel, 2003).

OCL is not useful for describing system behavior, and the other current UML models meant to describe system behavior–use case diagrams, action diagrams and sequence diagrams–are too informal and not precise enough for automation. Several groups (e.g., Kennedy Carter, n.d.; Mellor et al., 2002; Raistrick et al., 2002) are actively promoting Action Semantics (i.e., Executable UML or xUML) as a way to model system behavior in machine-readable ways. This language uses the concept of state machines to simulate behavior and creates models that can be executed on virtual machines (Raistrick et al., 2002). This approach has been used successfully in the development of real time systems. However, Warmer et al. (2003), Frankel (2003), and Klasse Objecten (2004a) all question the viability of Action Semantics and state machines because the language requires the development of low-level, detailed models; it does not have a standard concrete syntax like OCL, and state machines that require identification of all possible system-states, while useful in narrowly defined systems like real-time applications, might not be viable for complex enterprise applications. The process also requires the development of simulation environments that may not be possible in many organizations (Frankel, 2003), though products like OptimaJ might provide some of the capability off the shelf (Stephenson, 2003). The literature bears out that there is still considerable disagreement in this area, and it will be some time before transformation-modeling standards are established and before MDA models can effectively describe system behavior.

6. CUSTOMER INVOLVEMENT POTENTIAL

Customer involvement in the software development process is limited to activities that are essentially outside of the developer's realm. While business analysts, as the primary customer representatives, can become quite technically skilled and involved in evaluating some aspects of a system's design, customers are realistically limited to providing inputs to and evaluating outputs from the development team. Even in traditional processes that use prototypes, customers are still testing results from the team; they are not directly involved with the development activities. And in agile processes, where customers are often considered

full team members, their involvement is still limited to design and testing activities. In both processes, customers rarely (if ever) get involved in the actual programming. The adoption of an MDA approach will not change these customer roles or the nature of their activities. They will continue to provide application requirements, attempt to validate some of the system design specifications, and test applications produced by the development team. MDA will not change what customers do. However, it will significantly change when customers validate a system's functionality.

MDA impacts three areas of the development process: design, development, and testing. In the design area, the creation and validation of PIMs offers some potential for improving the integration of customers. In the actual development area, with its focus on PSMs and TMs, MDA offers none. It is in the testing area, with the validation of working applications, where its potential is the most significant. With automated code generation, MDA can greatly improve the integration of customers by integrating their system validation activities directly into the daily activities of the development team. In essence, it can extend the Development Bubble to incorporate customer testing.

7. SUMMARY AND CONCLUSIONS

Model Driven Architecture represents a fundamental evolution in the way software is developed. With MDA, the focus shifts from informal modeling and manual coding to precise and formal models and automated code generation. The PIMs, PSMs, and TMs will become key organizational assets that capture domain-specific knowledge from both the organizational and technical communities. As Frankel (2003) describes, these models will become persistent artifacts of development rather than just informal artifacts of design.

The literature indicates there is growing consensus in the information systems community on these fundamental principals of MDA: it should be based on underlying, consistent standards and syntaxes (meta-models); enable the separation of business and technical concerns, and use computer automation to manipulate and transform models and to generate code. However, there are some critical pieces missing in the areas of transformation and system-behavior modeling. Agreement on standards and approaches in these areas will take some time, and substantial work remains before MDA can replace traditional, long-practiced methods and become considered a routine approach to software development.

When this happens, MDA has the potential to significantly improve the integration of customers into the development lifecycle. It will not change the roles that customers play or

the nature of their activities. They will continue to provide application requirements, attempt to manually validate static system designs, and test working applications using the same requirements, use cases, and test cases they currently use. MDA will, however, make it possible, through the use of executable models and the automatic generation of prototypes and full systems, to incorporate customer validation activities directly into the daily activities of the development team. With automatic code generation, MDA removes the delay between design and test, and customers will be able to evaluate system functionality regularly throughout the development process.

8. REFERENCES

Akehurst, D., Linington, P., & Patrascoiu, O. (2003). *OCL 2.0: Implementing the standard technical report*. Retrieved September 22, 2004, from University of Kent Computer Laboratory web site: http://www.cs.kent.ac.uk/pubs/2003/1746/content.pdf

Akehurst, D.H. (2004). *Relations in OCL*. Paper for presentation in the workshop "OCL and Model Driven Architecture" at the 7th International UML 2004 Conference on October 12, 2004. Retrieved October 8, 2004, from http://www.cs.kent.ac.uk/projects/ocl/oclmdewsuml04/papers/9-akehurst.pdf

Arlow, J., & Neustadt, I. (2004). Enterprise patterns and MDA: building better software with archetype patterns and UML. Boston, MA: Addison-Wesley.

Baresi, L., Heckel, R., Thöne, S., & Varró, D. (2003). Modeling and analysis of architectural styles based on graph transformation. In Crnkovic, I., Schmidt, H., Stafford, J., & Wallnau, K. (eds.), *Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*. Carnegie Mellon University.

Bettin, J. (2003). *Ideas for a concrete visual syntax for model-to-model transformations*. Retrieved September 18, 2004, from http://www.softmetaware.com/oopsla2003/bettin.pdf

Booch, G. (2004, August). MDA: A motivated manifesto? *Software Development*. Retrieved September 24, 2004, from http://www.sdmagazine.com/documents/s=7206/sdm0408a/

Borland. (2004). *Keeping your business relevant with Model Driven Architecture (MDA)*. Retrieved September 22, 2004, from http://www.borland.com/products/white_papers/pdf/tgr_keeping_your_business_relevant_with_model_driven_architecture.pdf

Brown, A.W. (2004, February). An introduction to model driven architecture -- part I: MDA and today's systems. *The Rational Edge*. Retrieved September 24, 2004, from http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/feb04/3100.pdf

Cariou, E., Marvie, R., Seinturier, L., & Duchien, L. (2004). *OCL for the specification of model transformation contracts*. Paper for presentation in the workshop "OCL and Model Driven Architecture" at the 7th International UML 2004 Conference on October 12, 2004. Retrieved October 8, 2004, from http://www.cs.kent.ac.uk/projects/ocl/oclmdewsuml04/papers/2-cariou_marvie_seinturier_duchien.pdf

. Czarnecki, K., & Helsen, S. (2003). *Classification of model transformation approaches*. Presented at the OOPSLA'03 workshop "Generative Techniques in the Context of Model-Driven Architecture" on October 27, 2003. Retrieved September 24, 2004, from http://www.softmetaware.com/oopsla2003/czarnecki.pdf

Frankel, D. (2003). Model driven architecture: applying MDA to enterprise computing. Indianapolis: Wiley and Sons, Inc.

Frankel, D.S., Harmon, P., Mukerji, J., Odell, J., Owen, M., Rivitt, P., et al. (2003, September). *The Zachman Framework and the OMG's Model Driven Architecture*. Business Process Trends Whitepaper. Retrieved September 29, 2004, from http://www.omg.org/mda/mda_files/09-03-WP_Mapping_MDA_to_Zachman_Framework1.pdf

Frankel, D. (2004, March 2). *MDA Journal: The MDA Marketing Message and the MDA Reality*. Retrieved October 15, 2004, from BPTrends.com web site: http://www.bptrends.com/publicationfiles/03%2D04%20COL%20Marketing%20Message%20%2D%20Reality%20Frankel1%2Epdf

Gardner, T., Griffin, C., Koehler, J., & Hauser, R. (2003, July 21). *A review of OMG MOF 2.0 Query/Views/Transformations submissions and recommendations towards the final standard*. Retrieved October 8, 2004, from http://www.omg.org/docs/ad/03-08-02.pdf

Haustein, S., & Pleumann, J. (2004). *OCL as expression language in an action semantics surface language*. Paper for presentation in the workshop "OCL and Model Driven Architecture" at the 7th International UML 2004 Conference on October 12, 2004. Retrieved October 8, 2004, from http://www.cs.kent.ac.uk/projects/ocl/oclmdewsuml04/papers/4-haustein_pleumann.pdf

Hubert, R. (2001). Convergent architecture: building model-driven J2EE systems with UML. New York, NY: Wiley and Sons, Inc.

Kennedy Carter (n.d.). *Supporting MDA with executable UML*. Retrieved September 24, 2004, from http://www.kc.com/MDA/xuml.html

Klasse Objecten (2004, February 27). *MDA frequently asked questions*. Retrieved September 22, 2004, from http://www.klasse.nl/english/mda/mda-faq.html Klasse Objecten (2004, February 27). *The current status of the MDA*. Retrieved September 22, 2004, from http://www.klasse.nl/english/mda/mda-status.html

Klasse Objecten (2004, February 27). *What is the Model Driven Architecture?* Retrieved September 22, 2004, from http://www.klasse.nl/english/mda/mda-introduction.html

Kleppe, A., & Warmer, J. (2000). Extending OCL to include actions. In A. Evans, S. Kent, & B. Selic (Eds.), *Proceedings of the 3rd International Conference UML 2000*, (pp. 440-450). Lecture Notes in Computer Science, volume 1939. Berlin, Springer.

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture: Practice and promise.* Boston, MA: Addison-Wesley.

Küster, J.M., Sendall, S., & Wahler, M. (2004). *Comparing two model transformation approaches*. Paper for presentation in the workshop "OCL and Model Driven Architecture" at the 7th International UML 2004 Conference on October 12, 2004. Retrieved October 8, 2004, from http://www.cs.kent.ac.uk/projects/ocl/oclmdewsuml04/papers/6-kuster_sendall_wahler.pdf

Mellor, S.J., & Balcer, M.J. (2002). *Executable UML: a foundation for model-driven architecture*. Boston, MA: Addison-Wesley.

Mellor, S.J., Scott, K., Uhl, A., & Weise, D. (2004). *MDA distilled: Principles of model-driven architecture*. Boston, MA: Addison-Wesley.

Object Management Group (2003). *Business semantics of business rules: Request for proposal*. Retrieved October 10, 2004, from the OMG web site: http://www.omg.org/docs/br/03-06-03.pdf

Object Management Group (2004). *Executive Overview*. Retrieved September 24, 2004, from OMG web site: http://www.omg.org/mda/executive_overview.htm

Oliver, D.W. (1997). *Engineering complex systems with models and objects*. New York, NY: McGraw-Hill.

Raistrick, C., Francis, P., & Wright, J. (2004). *Model Driven Architecture with Executable UML*. Cambridge, UK: Cambridge University Press.

Selic, B. (2003, September/October). The pragmatics of model-driven development [Electronic version]. *IEEE Software, 20*(5), 19-25.

Sendall, S., & Kozaczynski, W. (2003). Model transformation: the heart and soul of model-driven software development [Electronic version]. *IEEE Software, 20*(5), 42-45.

Sommerville, I. (2004). *Software Engineering* (7th ed.). Boston, MA: Addison-Wesley.

Starr, L. (2002). *Executable UML: how to build class models*. Upper Saddle River, NJ: Prentice Hall.

Stephenson, J. (2003, August). *Model Driven Architecture and governance*. Retrieved September 22, 2004, from ITPapers.com web site: http://www.compuware.com/dl/cbdimda.pdf

Sunyé, G., Pennaneac'h, F., Ho, W., Le Geunnec, A., & Jézéquel, J. (2001). Using UML action semantics for executable modeling and beyond. In K.R. Dittrich, A. Geppert, & M.C. Norrie (Eds.), *Advanced information systems engineering: 13th international conference, CAiSE 2001 proceedings* (pp. 433-447). Lecture Notes in Computer Science, volume 2068. Berlin, Springer.

Warmer, J., & Kleppe, A. (2003). *The object constraint language: getting your models ready for MDA*. Boston, MA: Addison-Wesley.

Welsh, T. (2004, September/October). *MDA at the tipping point*. Application Development Advisor. Retrieved October 13, 2004, from http://www.appdevadvisor.co.uk/features/