11 Model-Driven Architecture

Liming Zhu

11.1 Model-Driven Development for ICDE

One problem lurking at the back of the ICDE development team’s mind is related to capacity planning for new ICDE installations. When an ICDE installation supports multiple users, the request load will become high, and the hardware that the platform runs on needs to be powerful enough to support this request load. If the hardware becomes saturated, it will not be able to process all user generated events, and important data may be lost. The situation is exacerbated by the following issues:

- Different application domains and different individual installations within each domain will use ICDE in different ways, and hence generate different request loads per user.
- Different installations will deploy ICDE on different hardware platforms, each capable of supporting a different number of users.
- The ICDE platform will be ported to different J2EE application servers, and each of these has different performance characteristics.

All of these issues relate to the software engineering activity of capacity planning. Capacity planning is concerned with how large, in terms of hardware and software resources, an installation must be to support its expected request load. Mathematical modeling techniques can sometimes be used to predict a platform’s capacity for standardized components and networks50. But more typically, benchmark tests are executed on a prototype or complete application to test and measure how the combined hardware/software deployment performs.

50 For example, Microsoft’s Capacity Manager and its support for Exchange deployments.
The only realistic way the ICDE team could anticipate to carry out capacity planning was to execute a test load on specific deployment platforms. For each installation, the team would need to:

- Install ICDE on the target hardware platform, or one that is as close as possible in specification to the expected deployment platform.
- Develop sample test requests generated by software robots to generate a load on the platform, and measure how it responds. The test requests should reflect the expected usage profile of the users operating on that ICDE installation.

So, for each installation, a set of tests must be developed, each of which will execute a series of requests on the ICDE platform and measure the response time and throughput. This is shown in Fig. 72.

---

**Fig. 72.** Capacity planning for ICDE installations

Not surprisingly, the ICDE team were extremely interested in making this whole capacity planning exercise as efficient and painless as possible. This would mean minimizing the amount of site-specific development. So for example, instead of writing a test robot specific for every installation, they would like to define the test load and test data externally to the code, and somehow input this into the robot to interpret. They would also like the performance results from test runs to be produced and collated automatically as graphs for easy analysis.
To achieve this, the team decided to exploit model-driven architecture methods and supporting development technologies. Model-driven approaches encourage the components of a software system to be described in UML models. These models are then input into code generators that automatically produce executable code corresponding to the model. The team hoped they could develop a single model of an ICDE test robot. Then, by simply changing parameters in the model, they could generate an installation-specific load test at the press of a button.

This chapter describes the essential elements of model-driven architecture approaches. It then shows how the ICDE team could use model-driven techniques to automate the development, deployment and results gathering of an ICDE installation for efficient capacity planning purposes.

11.2 What is MDA

One recurring theme in the evolution of software engineering is the ongoing use of more abstract formal languages for modelling solutions. In much mainstream software development, abstract descriptions, for example in Java or C#, are transformed by tools into executable forms. Developing solutions in abstract notations increases productivity and reduces errors because the translation from abstract to executable forms is automated by translation tools like compilers.

Of course, few people believe the nirvana of abstract programming languages is Java, C# or any of their modern contemporaries. In fact, the history of programming languages research is strewn with many proposals for new development languages, some general-purpose, some restricted to narrow application domains. A small minority ever see the light of day in “developerland”. This doesn’t stop the search from continuing however.

Model-driven architecture (MDA) is a recent technology that leads the pack in terms of more abstract specification and development tools (and use of new acronyms) aimed at the IT market. MDA is defined by the OMG\(^51\) as “an approach to IT system specification that separates the specification of functionality from the specification of the implementation”.

As the name suggests, an “application model” is the driving force behind MDA. A model in MDA is a formal specification of the function, structure and/or behaviour of an application or system. In the MDA approach, an IT system is first analysed and specified as a “Computation Independent Model” (CIM), also known as a domain model. The CIM fo-

---

\(^51\) Object Management Group: www.omg.org
Model-Driven Architecture focuses on the environment and requirements of the system. The computational and implementation details of the system are hidden at this level of description, or are yet to be determined.

![Diagram](attachment:image.png)

**Fig. 73.** Model transformation in MDA

As Fig. 73 shows, the CIM is transformed into a “Platform Independent Model” (PIM) which contains computational information for the application, but no information specific to the underlying platform technology that will be used to eventually implement the PIM. Finally, a PIM is transformed into a “Platform Specific Model” (PSM), which includes detailed descriptions and elements specific to the targeted implementation platform.

A “platform” in MDA is defined as any set of subsystems and technologies that provide a coherent set of functionalities through interfaces and specified usage patterns. An MDA platform is therefore a very broad concept. Platforms often refer to technology specific sets of subsystems which are defined by a standard, such as CORBA or J2EE. Platforms can also refer to a vendor specific platform which is an implementation of a standard, like BEA’s WebLogic J2EE platform, or a proprietary technology like the Microsoft .NET platform.

MDA is supported by a series of OMG standards, including the UML, MOF (Meta-Object Facility), XMI (XML Metadata Interchange), and CWM (Common Warehouse Metamodel). MDA also includes guidelines and evolving supporting standards on model transformation and pervasive services. The standards in MDA collectively define how a system can be developed following a model driven approach and using MDA compatible tools. Each MDA standard has its unique role in the overall MDA picture.

In MDA, models need to be specified by a modelling language. This can range from generic modelling languages applicable to multiple domains (e.g. UML) to a domain specific modelling language. The MOF provides facilities to specify any modelling language using MOF’s metamodeling facilities, as depicted in Fig. 74.

The MOF also provides mechanisms to determine how any model defined in a modelling language can be serialized into XML documents or be represented by programmable interfaces. Any existing modelling language can be made MDA compatible by creating a MOF representation of the language. There are many debates on MOF’s expressiveness and complex-
11.2 What is MDA

The OMG has recognized these limitations and is actively working to address them in future versions.

Fig. 74. The role of MOF in MDA

The UML and CWM are two relatively generic MOF-defined modelling languages and are included in the MDA standards package. UML focuses on object modelling and CWM focuses on data modelling.

The XMI standard in MDA is a mapping which can be used to define how an XML schema and related XML serialization facilities can be derived from a modelling language metamodel specified using the MOF. For example, the OMG has applied XMI to the UML metamodel to come up with an XML schema for representing UML models. Consequently, the XML schema for UML models can be used by UML modelling tool vendors to interchange UML models.

So, from business domain models, to analysis models, to design models and finally code models, MDA principles cover every phase of the software development process, artefacts and tooling. In the next sections, we will discuss the overall benefits of MDA and give some examples.
11.3 Why MDA?

Models play the central role in MDA. But why exactly do we need models? Here’s the answer.

Models provide abstractions of a system that allow various stakeholders to reason about the system from different viewpoints and abstraction levels. Models can be used in many ways, for example, to predict the qualities (e.g. performance) of a system, validate designs against requirements, and to communicate system characteristics to business analysts, architects and software engineers. And importantly in the MDA world, they can be used as the blueprint for system implementation.

The three primary goals of MDA are portability, interoperability and reusability, achieved through architectural separation of concerns. Critical design issues concerning the CIM, PIM and PSM are very different in nature and can evolve independently of each other. Multiple CIMs, PIMs and PSMs can exist for one application, reflecting different refinement levels and viewpoints. Let’s see how these primary goals are achieved in MDA.

![Fig. 75. MDA model mappings](image-url)
11.3 Why MDA?

11.3.1 Portability

Portability is achieved by model separation and transformation. High level models do not contain low level platform and technical details. As Fig. 75 illustrates, when the underlying platforms change or evolve, the upper level models can be transformed to a new platform directly, without any remodelling.

Portability is also achieved by making models moveable across different tool environments. The MOF and XMI standards allow a UML model to be serialized into XML documents that can be imported into a new tool for different modelling and analysis purposes.

11.3.2 Interoperability

There is rarely an application which does not communicate with other applications. Enterprise level applications particularly need to communicate across internal and external organizational boundaries in a heterogenous and distributed manner. Most of the time, you have limited control over the other systems you need to interoperate with.

Using MDA, interoperability is achieved through horizontal model mapping and interaction (see Fig. 76). Early versions of MDA guidelines refer to integration as the single biggest goal for MDA, which aims to improve interoperability in two ways:

- The interoperability problem can be seen as a problem of horizontal model mapping and interaction. For simplification, let’s suppose we have two sets of CIM/PIM/PSM for the two systems, as shown in Fig. 76. The interaction between higher level CIMs and PSMs can be first modelled and analysed. These cross model mappings and interactions then can be mapped to detailed communication protocols or shared databases supported by the underlying models. Since explicit vertical transformations exist between models in each system, the elements involved in the high level mapping can be easily traced or even automatically translated into lower level elements.

- The same problem can also be seen as a problem of refining a single high level model into multiple models operating across two or more platforms. Different parts of the higher level models are refined into models specific to different platforms. Associations in the original models are refined into communication channels or shared databases between platform specific models.
With unified metamodeling facilities and explicit model transformation tools, these two approaches become feasible in practice.

![Diagram](Diagram.png)

**Fig. 76.** Horizontal model mapping for interoperability

### 11.3.3 Reusability

Reusability is the key to improving productivity and quality. MDA encourages reuse of models and best practices in designing applications, especially in creating families of applications as in software product lines (see Chap. 9). MDA supports software product line approaches with increasing levels of automation. For example, the PIM is intended for reuse by mapping to different PSMs that a product line supports, and an MDA platform is designed for reuse as a target for multiple applications in a product line.
11.4 State-of-Art Practices and Tools

Although it is possible to practice parts of the MDA without tool support, this is only recommended for the brave and dedicated. A large portion of the standards is aimed at tooling and tool interoperation. Some standards are meant to be mainly machine readable, and not for general human consumption.

Since MDA standards, especially the guidelines, are intentionally suggestive and non-prescriptive, there has been a plethora of tools claiming to support MDA, all with very different features and capabilities. Some loosely defined parts of MDA have caused problems in terms of tool interoperability and development artefact reusability. However, the correct balance between prescriptive and non-prescriptive standards is hard to determine \textit{a priori} and requires real world inputs from industry users.

We’ll now discuss some promising tool examples from the J2EE/Java platform community because of its relatively wide adoption of MDA. The .NET platform is also moving towards model driven approaches through its own Domain Specific Language (DSL) standard. This is not compatible with MDA although third party vendors have successfully developed MDA tools for .NET Platform.

Although the tools discussed in the following have their roots in J2EE/Java technologies, all here have the capability to support other platforms. The architecture and infrastructure services of these tools all allow extensions and “cartridges” to be built to support other platforms. Some of them simply have out of the box support for J2EE related technologies.

11.4.1 AndroMDA

AndroMDA is an open source MDA framework. It has a plug-in architecture in which platforms and supporting components can be swapped in and out at any time. It heavily exploits existing open source projects for both platform specific purposes (e.g. XDoclet for EJB) and general infrastructure services (Apache Velocity for transformation templating).

In AndroMDA, developers can extend the existing modelling language through facilities known as “metafacades”. The extension is reflected as a UML profile in modelling libraries and templates in transformation tools. AndroMDA’s current focus is to generate as much code as possible from a marked PIM using UML tagged values, without having an explicit PSM file (it exists only in memory). Hence it does not provide opportunities for PSM inspection and bi-directional manipulation between PSM and PIM.
The reason for this is mainly because of the trade off between the complexity of bi-directional PIM/PSM traceability and the benefits of maintaining explicit PSMs for different platforms. At the UML stereotype level, this approach usually works well because only general platform independent semantics are involved, but for code generation, markings through tagged values usually includes platform dependent information which pollutes PIMs to a certain degree.

11.4.2 ArcStyler

ArcStyler is one of the leading commercial tools in the MDA market. It supports the J2EE, and .NET platforms out of the box. In addition to UML profiles, ArcStyler uses its own MDA “marks” as a way to introduce platform dependent information in PIMs without polluting the model with platform level details. Like AndroMDA, ArcStyler supports extensible cartridges for code generation. The cartridges themselves can also be developed within the ArcStyler environment following MDA principles. The tool also supports model to model transformation through external explicit transformation rule files.

11.4.3 Eclipse Modelling Framework (EMF)

The inseparable link between MDA models and the code created through code generation requires consistent management of models and code in a single IDE. EMF is the sophisticated metamodeling and modelling framework behind the Eclipse IDE. Although EMF was only released publicly as an Eclipse sub-project in 2003, it has a long heritage as a model driven metadata management engine in IBM’s Visual Age IDE.

EMF is largely MDA compatible with only minor deviations from some of the standards. For example, the base of EMF’s metamodeling language is known as Ecore, which is close but not identical to the Essential MOF (EMOF) in MOF 2.0. EMF can usually load an EMOF constructed metamodel, and mappings and transformations have been developed between EMOF and Ecore.

EMF comes with standard mechanisms for building metamodels and persisting them as programmable interfaces, code and XML (see Fig. 77). A model editor framework and code generation framework are also provided. However, EMF does not include any popular platform support out of the box, and it didn’t initially impress the MDA community as a full fledged ready-to-use MDA tool for platform-based distributed systems.
However, EMF’s tight integration with the Eclipse IDE and the capability of leveraging the Eclipse architecture and common infrastructures supports the integration of disparate metadata across multiple tools cooperating in a common Eclipse-based ecosystem. This raises the level of tool interoperability while being largely compatible with MDA practices.

This is also an example that demonstrates that model driven principles and standards go beyond the modelling of the system, and include modelling of all aspects of system construction. With little fanfare, IBM has migrated many of its development tools to Eclipse and manages their metadata via EMF. Third party vendors are also actively developing EMF based tools.

![Fig. 77. The Eclipse Modeling Framework](image)

Due to the ongoing standardization of model transformation and the significant production gains from code generation, most existing tools focus on code generation from models. The support for model to model transformation is usually lacking. This results in primitive support for bi-directional CIM-PIM-PSM transformation. Overall though, the MDA market is maturing with both industry strength commercial and open source tools emerging.

### 11.5 MDA and Software Architecture

Most of models in MDA are essentially representations of a software architecture. In a broad sense, domain models and system models are abstractions and different viewpoints of software architecture models. Generated code models possess the characteristics of the architecture models along with implementation details. The code can in fact be used in reverse engineering tools to reconstruct the application architecture.
A software architecture can be described in an architecture description language (ADL). There have been many ADLs developed in recent years, each with their expressiveness focused on different aspects of software systems and application domains. Many useful ADL features have recently been either absorbed into revisions of the UML, or specified as lightweight (through UML profiles) or heavyweight (MOF) UML extensions. Hence, the UML is used in MDA as an ADL.

Some exotic formalisms and dynamic characteristics of certain ADLs still cannot be fully expressed using UML. But the growing MDA/UML expertise pool in industry along with high-quality architecture and UML modelling tools outweigh the downside of some modelling limitations in most domains.

11.5.1 MDA and Non-Functional Requirements

Non-functional requirements (NFRs) are a major concern of software architecture. NFRs include requirements related to quality attributes like performance, modifiability, reusability, interoperability and security. Although MDA does not address each individual quality attribute directly, it promotes and helps achieving these quality attributes because:

- A certain degree of interoperability, reusability and portability is built into all models through the inherent separation of concerns. We have explained how these benefits are achieved in previous sections.
- The MOF and UML profile mechanisms allow UML to be extended for modelling requirements and design elements specifically targeting NFRs. UML profiles for expressing NFRs exist, such as the OMG’s profile for performance, scheduling and time.
- Along with NFR modelling extensions for requirements and design, explicit model mapping rules encourage addressing quality attributes during model transformation.

11.5.2 Model Transformation and Software Architecture

A large part of software architecture R&D concerns how to design and validate software architectures that fulfil their requirements and are implemented faithfully to the design. One major obstacle in architecture design is the difficulty of designing an architecture that clearly captures how the various aspects of the design satisfy the requirements. For this reason, it can be difficult to systematically validate whether the architecture models fulfil the requirements, as traceability between requirements and design
elements is not formalized. This does not help to increase confidence that the architecture is fit for purpose.

In MDA, all modelling languages are well defined by syntax and semantics in a metamodel. The process of transforming from one model (e.g. requirements) to another model (e.g. design) is a systematic process, following explicitly defined transformation rules. This explicitness and potential automation could greatly improve the quality and efficiency of validating an architecture model.

However, at the time of writing, the transformation process is still suggestive from the general MDA guidelines. A model “Query, View and Transformation” (QVT) standard is under development and many proposals have been submitted for public review, with some of these implemented in tool environments. With the adoption of this standard in the near future, it is possible that much of the tacit knowledge, best practices and design patterns used in architecture design and evaluation will be formally codified as various forms of bi-directional transformation rules. These will create rich forms of traceability in architecture models. In fact, transformations based on patterns and best practices have already been implemented in some tools in addition to normal platform specific mappings between PIMs and PSMs.

11.5.3 SOA and MDA

A strong industry direction currently is service-oriented architectures (SOA). In SOA, enterprise solutions are viewed as federations of services which communicate using intra-service communication protocols. While the communication protocols have been undergoing standardization and evolved to support both heterogeneous platform communications (e.g. XML based SOAP) and pervasive services (WS-* standards), these standards do not address questions of semantic mappings and interactions between the multiple SOA based systems.

Both MDA and SOA try to solve the same interoperability problem but from a totally different perspective and level of abstraction. One is from the general semantic modelling perspective; the other is from the communication protocols and architecture style perspective. Following MDA, it is possible to consistently map high level semantic interactions and mappings between the two systems into lower level model elements and communication channels with necessary supporting services.

MDA can also increase productivity when the functions of a system need to be exposed as Web services, one of the most common requirements in SOAs. If the existing system is already modeled following MDA
rules, exposing its services is just a matter of applying transformation rules for the Web services platform. For example, in AndroMDA, the “webservice” cartridge provides WSDL and WSDD file generation using a simple UML profile. To expose the same business logic as Web services, users only need to change the business process PIM (the ultimate goal is to have no change) and use the “webservice” cartridge.

In summary, SOA bridges heterogenous systems through communication protocols, pervasive services and an associated service-oriented architecture style. MDA can take care of the seamless high level semantic integration between systems and transforming the system models into lower level SOA based facilities. This synergy between MDA and SOA might mean that the next generation service oriented computing world with a highly federated and flexible architecture is not too far away.

SOA and Web Services are described in the next chapter.

### 11.5.4 Analytical Models are Models too

The importance of using analytical models to examine characteristics of a system is often ignored, even in the official MDA guidelines. However, the benefits of having analytical models that are also compatible with MDA are potentially huge.

According to the MDA definition, a model is defined as a description of a system in a well-defined language. This definition can be applied to a wide range of models. For example, in performance engineering, we can choose to view a system as a queue-based model which has servers and queues. In modifiability analysis, we can choose to view a system as a dependency graph model which has nodes to represent conceptual or implementation elements and edges to represent dependency relationships among them.

Currently, these models are usually expressed in their own modelling languages. In order to build an analytical model for an existing UML model, either we have to do the modelling manually or a low level transformation must be carried out based on the UML model represented in XML. This is shown in Fig. 78, and has several limitations:

- The transformation relies solely on primitive XML transformation facilities such as XSLT. Debugging and maintenance is difficult with no clear semantic mapping between the two models.
- Without a clear semantic mapping and round trip engineering facilities, it is very hard to place the results gained from the analytical model back into the original UML model context.
• The original design model will likely be further refined and eventually implemented in code. The analytical model is essentially also a derived model from the same design model. But as the analytical model is not compatible with the MDA standard, it is even harder to cross-reference the analytical model with all the other derived models for validation, calibration and other purposes.

![Diagram showing MDA model transformation for model analysis](image)

**Fig. 78.** MDA model transformation for model analysis

### 11.6 MDA for ICDE Capacity Planning

In order to conduct capacity planning for ICDE installations, the ICDE team needed a test suite that could be quickly tailored to define a site-specific test load. It should then be simple and quick to execute the test suite on the intended deployment environment, and gather the performance statistics such as throughput and response time.

After a close look at their performance testing requirements, the ICDE team found that their needs for rapid development across different J2EE platforms were amenable to applying MDA principles, leveraging its support for portability, interoperability and reusability. The reasons are as follows:
- For different J2EE application servers, only the platform related plumbing code and deployment details differ. Using MDA, a generic application model could be used, and platform specific code and plumbing generated from the model. This leverages the portability inherent in MDA.
- The generation of repetitive plumbing code and deployment configuration is supported for many J2EE application servers code by a number of open source MDA projects. These code generation cartridges are usually maintained by a large active user community, and are of high quality. Thus the ability to reuse these cartridges in MDA tools was very attractive.
- The ICDE team has extensive experience in performance and load testing. By refactoring their existing libraries into a reusable framework, much of this can be easily reused across J2EE platforms. However, each site-specific test will require custom code to be created to capture client requirements. Using MDA, these site-specific features can be represented using UML stereotypes and tagged values, as a combination of modelling details and configuration information. From this design description, the MDA code generation cartridge can produce the site-specific features and hook these in with the team’s reusable framework components.

Fig. 79. Overview of ICDE’s MDA-based performance test generator
So, the ICDE team designed a UML profile and a tool to automate the generation of complete ICDE performance test suites from a design description. The input is a UML-based set of design diagrams for the benchmark application, along with a load testing client modeled in a performance tailored version of the UML 2.0 Testing Profile. The output is a deployable complete benchmark suite including monitoring/profile reporting utilities. Executing the generated benchmark application produces performance data in analysis friendly formats, along with automatically generated performance graphs. The tool is built on top of an open source extensible framework – AndroMDA. The overall structure of the benchmark generation and related process workflow is presented in the boxed area in Fig. 79.

A snippet of the model is represented in Fig. 80. The load testing entry point is the \texttt{ICDEAPIService}. It is the front end component of the system.

\footnote{http://www.omg.org/technology/documents/formal/test_profile.htm}
under test, which is marked with the <<SUT>> stereotype. ICDEAPIClient is the <<TestContext>> which consists of a number of test cases. Only the default loadTestAll() test case is included with its default generated implementation.

All the test data to be used for calling ICDE APIs is modeled in the TrxnData class. The TranDeck class contains values that configure the transaction mix for a test using tagged values, shown in Fig. 80. For example, calls to the ICDE API queryData represents 25% of all transactions and writeData represents 55% for the test defined in this model. This data is used to randomly generate the test data which simulates the real work load of the ICDE installation under test.

In Fig. 81, example test outputs are depicted for the response time distribution for two different application servers under a workload of 150 concurrent clients.

![Fig. 81. Example response time results](image)

The amount of time saved using MDA can be considerable. Community-maintained technology cartridges automatically generate repetitive and error prone plumbing code, and the best practices inherited through using the cartridges improve the quality of the performance testing software. Above all, MDA principles raise the abstraction level of the test suite development, making it easy and cheap to modify and extend.

For more information on this work, please refer to the MDABench reference at the end of the chapter.

### 11.7 Summary and Further Reading

MDA, as the industry wide standardization of model driven software development, is proving successful and is continuing to evolve. MDA impacts on software architecture practices, as it requires the architecture team
to create formal models of their application using rigorously defined modeling languages and supporting tools. This essentially represents raising the level of abstraction for architecture models. The software industry has been raising abstraction levels in software development (e.g. from machine code to assembly language to 3GLs to object-oriented languages and now to models) for the best part of five decades. MDA is the latest step in this direction, and if it achieves its goals the industry could attain new levels of development productivity only dreamt of today.

Still, MDA draws criticism from many sides concerning its limitations, some of which are arguably intrinsic and hard to improve without a major revision. Microsoft has chosen not to comply with MDA standards and follow its own path, defining and using its own DSL as the modelling language in its Visual Studio IDE. While this may splinter the development community and create incompatible models and tools, both the OMG’s and Microsoft’s promotion of general model-driven development principles is likely to have positive outcomes for the software community in the years to come.

The best reference for all MDA-related information is the OMG’s website:


Two good books on MDA from prominent authors are:


For some further details on the MDA-based performance and capacity planning tools, see: