WHITEPAPER

The Fast Guide to Model Driven Architecture

The Basics of Model Driven Architecture

Cephas Consulting Corp
Architecture Oriented Services

By Frank Truyen
frank.truyen@cephas.cc

The Fast Guide to Model Driven Architecture

The Basics of Model Driven Architecture (MDA)

Cephas Consulting Corp
January 2006

Table of Contents

Summary

This white paper is a first in a series of papers which provide a foundational and practical guide for software developers required to work within a model driven environment as prescribed by the OMG's Model Driven Architecture® (MDA®). It explains the MDA approach in terms of the major concepts, the premises, and the goals which drive its adoption.

This helps the software developer as well as the IT manager navigate through the concepts of MDA in order to understand its fundamental departure from traditional software development. The paper helps the reader discover the minimal set of requirements for a development environment to be "MDA" in a practicable manner.

How to Use This Guide

This guide is intended to serve as a practical reference to the ***companion tool-specific MDA white papers***. It provides the vocabulary and necessary concepts for applying MDA, and selecting the right modeling tool for your organization's development environment.

FIRST IN THE SERIES

ENTERPRISE ARCHITECT

FROM SPARX SYSTEM

Companion Paper 2, Features

In a separate series of tool-specific papers, *Model Driven Architecture with …*, the modeling tool's features are measured against our set of requirements, and the tools specific MDA implementation features are analyzed –how compliant and how pragmatic.

Companion Paper 3, Practice

A separate paper, ***MDA in Practice***, is a tutorial that uses a running example to illustrate the real application of MDA transformations. The tutorial is tool-specific and uses screen shots to show all the steps required to define and apply MDA transformations.

Model Driven Architecture

An Object Management Group (OMG) standard

The Object Management Group™ (OMG™) was formed as a standards organization to help reduce complexity, lower costs, and hasten the introduction of new software applications. One of the major initiatives through which the OMG is accomplishing this goal is by the promotion of Model Driven Architecture® (MDA®) as an architectural framework for software development. This framework is built around a number of detailed OMG specifications, which are widely used by the development community.

A brief history

In 2001 the OMG adopted the Model Driven Architecture as an approach for using models in software development. Its three primary goals are portability, interoperability and reusability through architectural separation of concerns.

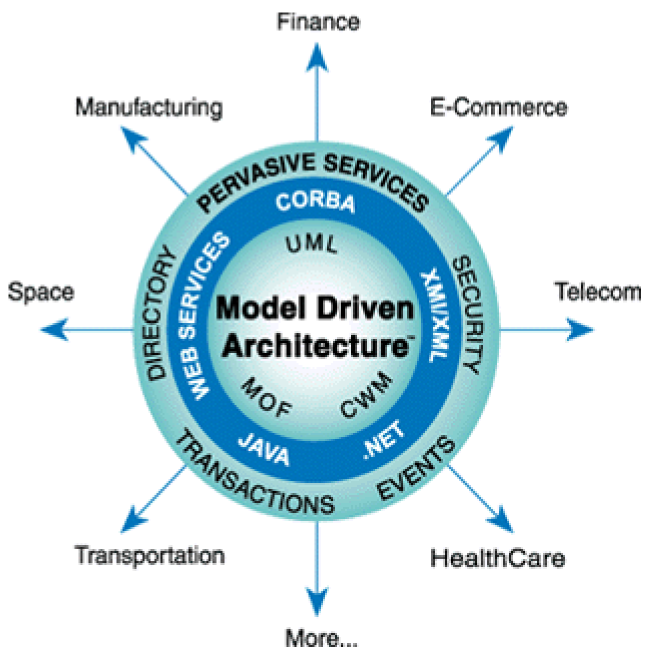
MDA in context

One fundamental aspect of MDA is its ability to address the complete development lifecycle, covering analysis and design, programming, testing, component assembly as well as deployment and maintenance.

MDA itself is not a new OMG specification but rather an approach to software development which is enabled by existing OMG specifications such as the Unified Modeling Language™ (UML®), the Meta Object Facility (MOF™) and the Common Warehouse Metamodel (CWM™). Other adopted technologies which are of interest are the UML Profile for Enterprise Distributed Object Computing (EDOC), including its mapping to EJB, and the CORBA® Component Model (CCM).

With new platforms and technologies constantly emerging, MDA enables the rapid development of new specifications that leverage them, and streamlines the process of their integration. In this way MDA provides a comprehensive, structured solution for application interoperability and portability into the future. Precise modeling of the solution domain in UML provides the added advantage of capturing its inherent intellectual property in a technology neutral way.

As illustrated in the following diagram, the OMG envisions MDA to encompass a full range of "pervasive" services, which are commonly found in modern distributed applications.



Major MDA concepts

System

The context of MDA is the software system, either preexisting or under construction.

Model

A model is a **formal** specification of the function, structure and behavior of a system within a given context, and from a specific point of view (or reference point). A model is often represented by a combination of drawings and text, typically using a formal notation such as UML, augmented where appropriate with natural language expressions.

A specification is said to be formal when it is based on a language that has a well defined semantic meaning associated with each of its constructs, to distinguish it from a simple diagram showing boxes and lines. It is this formalism, which allows the model to be expressed in a format such as XML, in accordance with a well-defined schema (XMI).

Model driven

Describes an approach to software development whereby models are used as the primary source for documenting, analyzing, designing, constructing, deploying and maintaining a system.

Architecture

The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors[1].

[1] Shaw and Garlan, Software Architecture, Prentice Hall

Within the context of MDA these parts, connectors and rules are expressed via a set of interrelated models.

Viewpoint

A viewpoint is an abstraction technique for focusing on a particular set of concerns within a system while suppressing all irrelevant detail. A viewpoint can be represented via one or more models.

MDA viewpoints

MDA specifies three default viewpoints on a system: computation independent, platform independent and a platform specific.

The computation independent viewpoint focuses on the context and requirements of the system without consideration for its structure or processing.

The platform independent viewpoint focuses on the operational capabilities of a system outside the context of a specific platform (or set of platforms) by showing only those parts of a complete specification that can be abstracted out of that platform.

A platform specific viewpoint augments a platform independent viewpoint with details relating to the use of a specific platform.

Platform

A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and usage patterns. Clients of a platform make use of it without concern for its implementation details. Examples of platforms include operating systems, programming languages, databases, user interfaces, middleware solutions etc.

Platform independence

Platform independence is a quality that a model may exhibit when it is expressed independently of the features of another platform. *Independence* is a relative indicator in terms of measuring the degree of abstraction, which separates one platform from another (i.e. where one platform is either more or less abstract compared to the other).

Platform Model

A platform model describes a set of technical concepts representing its constituent elements and the services it provides. It also specifies constraints on the use of these elements and services by other parts of the system.

Model Transformation

Model transformation is the process of converting one model to another within the same system. The transformation combines the platform independent model with additional information to produce a platform specific model.

Implementation

An implementation is a specification that provides all the information required to construct a system and to put it into operation. It must provide all of the information needed to create an object, and to allow the object to participate in providing an appropriate set of services as part of the system.

MDA models

MDA specifies three default models of a system corresponding to the three MDA viewpoints defined above. These models can perhaps more accurately be described as layers of abstraction, since within each of these three layers a set of models can be constructed, each one corresponding to a more focused viewpoint of the system (user interface, information, engineering, architecture, etc.).

Computation Independent Model (CIM)

A CIM is also often referred to as a business or domain model because it uses a vocabulary that is familiar to the subject matter experts (SMEs). It presents exactly what the system is expected to do, but hides all information technology related specifications to remain independent of how that system will be (or currently is) implemented.

The CIM plays an important role in bridging the gap which typically exists between these domain experts and the information technologists responsible for implementing the system.

In an MDA specification the CIM requirements should be traceable to the PIM and PSM constructs that implement them (and vice-versa).

Platform Independent Model (PIM)

A PIM exhibits a sufficient degree of independence so as to enable its mapping to one or more platforms. This is commonly achieved by defining a set of services in a way that abstracts out technical details. Other models then specify a realization of these services in a platform specific manner.

Platform Specific Model (PSM)

A PSM combines the specifications in the PIM with the details required to stipulate how a system uses a particular type of platform. If the PSM does not include all of the details necessary to produce an implementation of that platform it is considered abstract (meaning that it relies on other explicit or implicit models which do contain the necessary details).

A growing rate of adoption

MDA has been profitably implemented in many small and large organizations. While some companies prefer to keep their success a secret to their competition, many have agreed to publish their accomplishments, as can be seen on the OMG website[2] as well as in various magazine articles[3].

The promise of MDA

The promise of Model Driven Architecture is to facilitate the creation of machine-readable models with a goal of long-term flexibility in terms of:

- **Technology obsolescence**: new implementation infrastructure can be more easily integrated and supported by existing designs.

[2] http://www.omg.org/mda/products_success.htm

[3] For example at http://www.softwaremag.com/L.cfm?Doc=2005-07/2005-07mdauml

- **Portability**: existing functionality can be more rapidly migrated into new environments and platforms as dictated by the business needs.
- **Productivity and time-to-market**: by automating many tedious development tasks architects and developers are freed up to focus their attention on the core logic of the system.
- **Quality**: the formal separation of concerns implied by this approach plus the consistency and reliability of the artifacts produced all contribute to the enhanced quality of the overall system.
- **Integration**: the production of integration bridges with legacy and/or external systems is greatly facilitated.
- **Maintenance**: the availability of the design in a machine-readable form gives analysts, developers and testers direct access to the specification of the system, simplifying their maintenance chores.
- **Testing and simulation**: models can be directly validated against requirements as well as tested against various infrastructures. They can also be used to simulate the behavior of the system under design.
- **Return on investment**: businesses are able to extract greater value out of their investments in tools.

The MDA Process

Whatever the ultimate target platform may be, the first step when constructing an MDA-based application is to create a platform-independent model expressed via UML. This general model can then be transformed into one or more specific platforms such as CCM, EJB, .NET, SOAP, etc.

A complex system may consist of many interrelated models organized along well defined layers of abstraction, with mappings defined from one set of models into another. Within this global set of models **horizontal** transformations may occur inside a single layer of abstraction, in addition to the typical **vertical** transformations across layers.

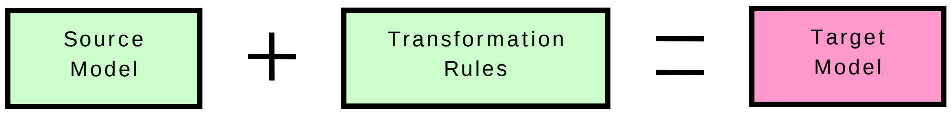
Applying a consistent architectural style across viewpoints of the system is one illustration of such a horizontal transformation. Other examples include:

- Within a CIM, create a target model containing all of the business rules defined in the core business model.
- Within a PIM, create a target model containing only the data elements define in the conceptual model.
- Within a PSM, create a target JUnit test model from a Java class model.

Note also that a PSM at one layer of abstraction may take on the role of a PIM with regards to a further transformation down into another layer.

Beyond the perhaps simplistic notion of CIM/PIM/PSM, the two key concepts of MDA are **models** and **transformations**.

The general pattern is:



This pattern can be repeatedly applied to successive models, each one playing the role of either a PIM or a PSM.

Transformation Mappings

An MDA mapping provides specifications for how to transform a PIM into a particular PSM. The target platform model determines the nature of the mapping. While part of the transformation can result from a manual exercise, the intent is clearly to automate as much of the process as allowed by the toolset in use.

Let's take the example of a mapping which defines annotations (called marks in MDA parlance) that are to be used for guiding the transformation of a UML PIM to an EJB PSM: marking a UML class with the Stereotype "Session" would result in the creation of a session bean - and other supporting classes - within the PSM.

Transformation rules between models can be expressed:

- At the type level, from types specified in the PIM language to types expressed using the PSM language. In UML, examples of such *types* referred to here include *class*, *attribute* and *operation*.

- In accordance to patterns of type usages in the PIM (e.g. a GOF *strategy* pattern within a PIM can translate into an idiomatic equivalent in the PSM).

- At the metamodel (MOF) level, for transformations that need to bridge model languages.

For the purpose of this white paper we limit our scope of discussion to mappings from one UML model to another UML model.

Marking Models

Type mappings are generally insufficient to specify a complete transformation: additional rules are required to specify that certain types in the PIM must be annotated (**marked**) a specific way in order to produce the desired output in the PSM. This extra information cannot be determined from the PIM itself.

A mark represents a concept in the PSM which is applied to an element of the PIM in order to indicate how that element is to be transformed.

Marks, being platform specific, are considered part of the PSM (in a multiple PSM scenario each PSM would use different marks against the same PIM). A PIM plus all of the platform marks constitutes the input to the transformation process resulting in a PSM.

Implicit or explicit transformation rules exist to indicate which model elements in the PIM are suitable for certain marks in order to generate the desired element in the PSM.

The set of marks can be viewed as an overlay (or transparency) placed over the PIM for the purpose of the transformation. This set can optionally be supplied as part of a UML profile.

Another option is to associate a set of marks with a template containing the rules according to which instances in a model are to be transformed. These rules can specify which values in the source model can be used to fill the parameters of the template.

Mapping Languages

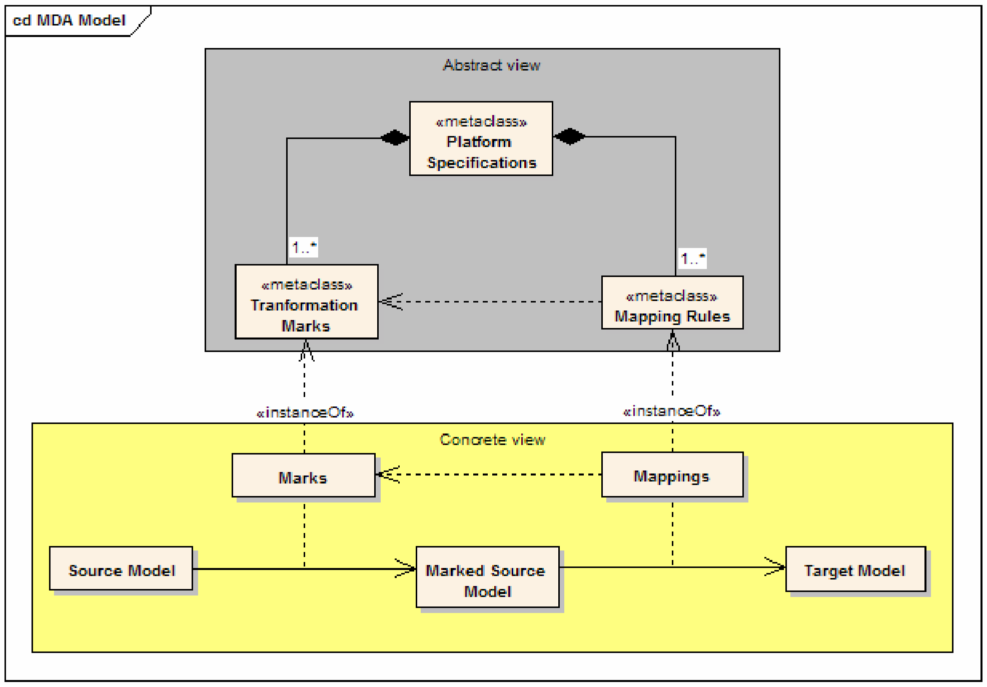
A model transformation mapping must be specified using some language, be it a natural language, an action language, or a dedicated mapping language. The OMG is currently in the process of adopting the MOF Query/View/Transformation (QVT) specification as a portable mapping language.

Recording the transformations

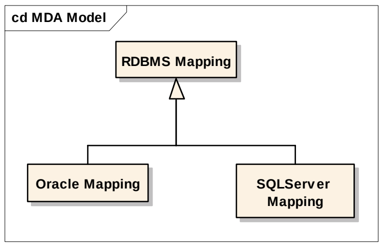
A record of the transformation should include a chart indicating which PIM elements were mapped to which PSM elements and include the mapping rule/s used for each part of the transformation.

Transformations illustrated

The following diagram uses UML notation to present the various concepts involved in an MDA transformation for a given platform (i.e. target model). It distinguishes between the abstract view of the platform's transformation directives, and a concrete implementation in the context of a specific source model to be transformed into this platform.



A further way to organize transformation mappings is via an inheritance hierarchy. For example, a mapping to create an RDBMS information model could be specialized for specific database vendors.



Generating code and other artifacts

The final step in a transformation process is to generate the implementation code as well as perhaps other kinds of supporting artifacts such configuration files, component descriptor files, deployment files, build scripts, etc. The more fully the application semantics and run-time behavior can be included in the PSM, the more complete the generated artifacts can be.

Depending on the maturity and quality of the MDA toolset, code generation varies from significant to substantial or, in some cases, even complete. Note that even minimal automation simplifies the development work and represents a significant gain because of the use of a consistent architecture for managing the platform-independent and platform-specific aspects of applications.

Next Steps

Think Transition

Moving from a traditional software development approach to one that is model driven is a full-fledged transition program that requires formal planning, orchestration and sponsorship. Transitioning an organization to the model driven approach is a major project/program that will take time and resources to plan and manage over a period of years. Defining the transition lifecycle consists of developing a strategy and a plan for each stage and monitoring the resulting activities. This requires a major on-going effort, and continuing support from IT management.

The transition is not a one-time process improvement effort, rather, it is a transition program that must be formally planned, approved and funded by management. It must be executed with all the diligence and check marks of a real project. Moving too fast, out-of-sequence, or proceeding without proper planning simply puts the whole transition process at risk. The main ingredient for ensured success is keeping the end-vision of the transition in sight.

Promote Tooling

Tooling is an essential part of the MDA environment because tool capabilities replace human skill sets. Understanding that the tools will generate most of your project artifacts, it is crucial to establish the right infrastructure for the development environment, along with a dedicated team to support it, manage it, and continually evolve it to meet the IT development requirements.

The IT organization must carefully evaluate and select the right tools with the capabilities to meet the organization's specific requirements. The companion tool-specific MDA papers offer guidance and feature checklists to consider.

Success Factors

Establish the foundation at the appropriate levels of abstraction

Establish the foundation, the equivalent to investing the time and forethought in the analysis phase of the transition project. The foundation is manifested as a set of 'blueprints' for your transition and a measurement system to track progress.

Set appropriate expectations Time, effort, and workload balance

Setting the appropriate expectations is crucial, especially when it comes to keeping early expectations in bounds with the current realities of the daily workload.

Design the development environment tooling and solution delivery

Designing the development environment properly for the model driven approach is one of the most crucial factors for success. This is where tooling takes over human grunt work and transformations replace routine skill sets.

Plan for pilot projects from pilot to prototype to real deliverable

Plan for pilot projects and don't be tempted to jump into a real project only equipped with notions of how to proceed in the new approach. Try before you buy! A successful pilot can always be turned into a real deliverable.

Plan for recurrent reviews Measure and adjust

Plan for recurrent "review-and-adjust" milestones. This is where you put measurement system to work. With no proof of progress, you cannot sustain funding.

References

MDA – A Technical Perspective by the OMG Architecture Board
ftp://ftp.omg.org/pub/docs/ab/01-02-04.pdf

MDA White Paper by Richard Soley and the OMG Staff Strategy Group
http://www.omg.org/docs/omg/00-11-05.pdf

A Proposal for an MDA Foundation Model (An ORMSC White Paper)
http://www.omg.org/docs/ormsc/05-04-01.pdf

MDA Guide Version 1.0.1
http://www.omg.org/docs/omg/03-06-01.pdf

MDA Explained – The Model Driven Architecture: Practice and Promise
Anneke Kleppe, Jos Warmer and Wim Bast – Addison-Wesley

Model Driven Architecture – Applying MDA to Enterprise Computing
David S. Frankel – OMG Press

The Object Constraint Language – Getting Your Models Ready for MDA
Jos Warmer & Anneke Kleppe – Addison-Wesley

Model Driven Architecture with Executable UML
Chris Raistrick, Paul Francis, John Wright, Colin Carter, Ian Wilkie

Business Component Factory
Peter Herzum and Oliver Sims – OMG Press

Executable UML ; A Foundation for Model Driven Architecture
Marc J. Balcer & Stephen J. Mellor – Addison-Wesley

About Cephas Consulting Corp.

COMPANY BACKGROUND

Since 2001, Cephas Consulting Corp. has been active helping its corporate clients introduce state of the art information technologies. We offer expertise in the areas of:

. Modeling business applications using object oriented techniques.

. Building distributed component infrastructures.

. Introducing formal software development processes.

. Migrating development organizations into Model Driven Architecture (MDA).

. Providing advanced UML/MDA training and mentoring.

COMPANY FOCUS

Cephas specializes in introducing UML® modeling into organizations via training and mentoring. The team of consultants and architects at Cephas draw on many years of experience to offer a one-stop solution addressing all aspects of managing the enterprise meta-data.

. Training & mentoring from beginner to expert level.

. Migrating meta-data out of legacy environments.

. Training for onsite guardianship of the development environment.

. Customizing the modeling tool in order to respond to unique client requirements.

. Providing expert level support and maintenance.

COMMITMENT TO THE OMG AND MDA

Cephas Consulting has the required expertise to lead organizations into the use of Model Driven Architecture. As early adopters we have successfully implemented MDA and are pleased to be among the first participants to the MDA FastStart program put in place by the OMG. We are also thrilled to work as OMG members on expanding the mind share of MDA in the marketplace, because we believe it is ideally suited to deal with the challenges of managing complex software development in times of rapid technology obsolescence.

Our highest commitment is in achieving success through quality, and we take pride in the accomplishments of our clients.

CONTACT DETAILS

Website : http://www.cephas.cc

General inquiries: cephas.contact@cephas.cc

Author inquiries: frank.truyen@cephas.cc