

Model Driven Architecture with Enterprise Architect

Measuring EA Features to MDA Concepts

By **Frank Truyen**
frank.truyen@cephas.cc





Model Driven Architecture with Enterprise Architect

Measuring MDA Concepts to EA Features

Cephas Consulting Corp

January 2010

Table of Contents

Overview	4
Trademarks	4
How to Use This Guide	4
Companion Paper1, Fast Guide	4
Companion Paper 2, Practice	4
MDA guidelines	6
Building computationally complete models	6
Adopt Design by Contract	6
Specify formal UML Constraints in behavior diagrams	6
PIM Completeness Guidelines	7
Applying MDA across viewpoints and tiers	8
Step 1 – Generating the first layer of abstraction	9
Step 2 – Generating the second layer of abstraction	9
Step 3 – Generating the implementation artifacts	9
Minimum tool requirements for MDA support	10
The Enterprise Architect solution for applying MDA	12
Product history	12
Commitment to MDA and OMG standards in general	12
Testing our MDA compliance criteria	12
Conclusion	14
References	15
About SPARX Systems	16
About Cephias Consulting Corp.	17

Overview

You are a lead engineer or an IT manager involved in establishing a model driven development environment and you are tasked with selecting a UML modeling tool that offers the right features to support the Model Driven Architecture approach. Where do you start? What makes a UML tool an MDA/UML tool? What are the core MDA capabilities you should be looking for? What should be your criteria to match such features to your enterprise requirements? If you are grappling with these or similar questions, this guide is written for you.

Support for MDA is rampant amongst UML tool vendors, and today many vendors claim “MDA support” or “MDA compliance.” This is a characterization that is challenging to validate for various reasons that we will not go through in this paper. As part of a series of white papers addressing the pragmatic aspects of MDA, this is a practical guide for organizations evaluating tools for their model driven environment. It includes general guidelines, tool selection, required MDA tool features, and the application of MDA using a specific tool.

This paper offers specific criteria you can use in modeling-tool selection and help you sort through the hype and promises of tool vendors. It also provides the set of minimal as well as optional requirements, which a modeling tool should satisfy in order to qualify as being “MDA compliant” in a practicable manner. Our criteria and compliance requirements do NOT come from some industry poll or study. Unanointed by vendor’s claims, the criteria come by way of real projects experience and are based on an experiential point of view.

Trademarks


OMG™, Object Management Group™, UML™, Unified Modeling Language™, MOF™, Model Driven Architecture™, MDA™, OMG Model Driven Architecture™, OMG MDA™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

How to Use This Guide


This guide covers general MDA guidelines and best practices, and it is specific to the Enterprise Architect (EA) UML tool. It provides the vocabulary and necessary concepts for applying MDA. In this guide, the modeling tool’s features are measured against our set of requirements, and the tool’s specific MDA implementation features are analyzed in terms of compliance and practical usage.

FIRST IN THE
SERIES
**ENTERPRISE
ARCHITECT**
FROM SPARX
SYSTEMS

Companion Paper1, Fast Guide

For an introduction to general MDA concepts, please see the companion paper  *The Fast Guide to Model Driven Architecture*, a quick reference for MDA starters.

Companion Paper 2, Practice

A separate paper,  *MDA in Practice*, is a tutorial which 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.

MDA guidelines

Building computationally complete models

How does one go about producing models which are semantically rich and precise enough to drive model transformations, and ultimately complete implementation code generation?

Declarative specifications, be it in the form of structural or behavioral UML models, are typically insufficient to produce code which encompasses all of the logic and rules of the system. For that purpose UML has defined an abstract Action Semantics Language (ASL) through which behavior can be expressed generically, at a level of abstraction beyond 3d GLs. However there is no adopted standard for a concrete syntax of this language, and there are no current standard mappings defined to common programming languages such as Java or C#. In addition, tool vendors which do provide such a generic language are typically not compliant with the specification, which means that any logic expressed through it can not be exchanged with other UML tools.

Well defined declarative specifications can however be sufficient to drive the generation of intermediate models, as well as much of the infrastructure artifacts which underlie modern software systems, in the form of various descriptors, XML schemas, test cases, database schemas, presentation tier components etc.

Here are some general guidelines for creating precise UML models which can be used as input to MDA transformation and generation processes:

Adopt Design by Contract

Adopt the Design by Contract (DBC) techniques of specifying constraints (where applicable) on:

- Operations, using **pre- and post-conditions**.
- On classes, attributes and other UML elements via **invariants**.

To be machine readable these constraints should be specified using the Object Constraint Language (OCL), in addition to a textual version intended for the casual reader. A model which does not express the constraints underlying its behavior can not be considered semantically complete.

DBC constraints typically map to exceptions, however defined in a specific programming language. They are also crucial for the manual or automated development of test harnesses.

Specify formal UML Constraints in behavior diagrams

- Adorn UML State machines describing the internal behavior of an element, for example a class, with constraints in the form of guard conditions.
- Attach constraints to Activity diagrams (which can be used at many levels of abstraction) through various condition and guard expressions.
- Leverage the additions of UML 2.0 to the semantics of Interaction diagrams, which now permit the modeling of conditional execution, loops, and time & duration constraints, all of which contribute to greater precision.

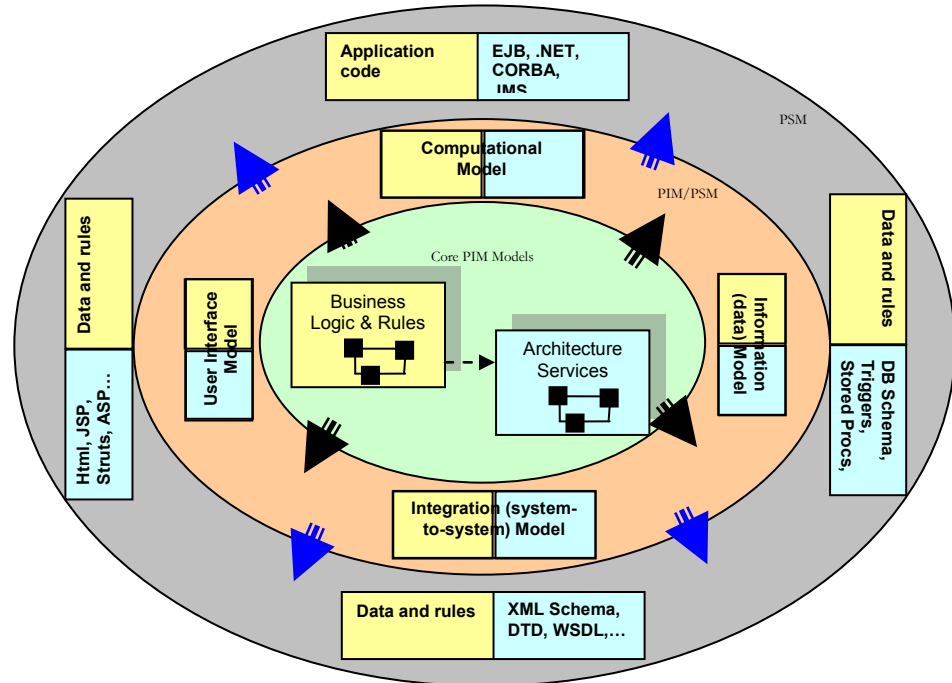
PIM Completeness Guidelines

For PIM class diagrams to be complete, all of the following should be considered:

- Do not define accessor and mutator operations for attributes. Instead let the transformation rules create the operations as appropriate for the target platform.
- Consistently set the navigability of each association end. Transformers and code generators rely on this to make appropriate mapping decisions.
- Just like for attributes, do not define accessor and mutator operations for navigable association ends.
- When appropriate, mark class properties (attributes and association ends) with the *read-only* constraint (UML property *isReadOnly*) to prevent the generation of mutator operations.
- For multi-valued unbound properties (i.e. with a multiplicity > 1 and no upper limit defined) state if the resulting collection is ordered or not (UML property *isOrdered*) and whether duplicate items may exist or not (UML property *isUnique*). Transformers make use of these constraints to select different language constructs to implement the collection (e.g. Java *List* versus *Map*).
- Define the multiplicity of **both** ends of an association, regardless of whether an end is navigable.
- Define unique names for all attributes of a class, as well as unique role names for all its navigable association ends.
- Optionally name the associations themselves (some MDA tools will automatically generate internal association names for transformation purposes).
- Draw UML dependency relationships to represent implementation-level dependencies between classes or components. Transformers can use this information, for example to generate *include* statements in code.
- Modeling package and component dependencies can also be important for generating correct build scripts.
- Use composition relationships when the intent is to tie the lifecycle of the *parts* to the lifecycle of the *whole*. Appropriate *destructor* logic can be generated from this notation.
- Mark classes that are not intended to be instantiated as *abstract* to avoid the creation of unnecessary factory operations.
- Specify the data type of all attributes.

Applying MDA across viewpoints and tiers

Which aspects of a typical business software system are the best candidates for applying the MDA approach? The following diagram illustrates the most common viewpoints and tiers:



At the core of the system resides a set of Platform Independent Models capturing the business logic and rules, and the architectural services that the application(s) will rely upon. The MDA documentation refers to a set of *pervasive services* which typically support a wide range of business components and applications. Examples include Event, Persistence, Transaction, Security, and Directory services. These services are an essential foundation for satisfying many non-functional requirements such as security, performance and fault tolerance.

Many organizations do not make the effort to abstract out such services from the underlying platforms realizing them. Some feel that, for example, the use of CORBA as a middleware platform provides sufficient protection from a technology lock-in since many implementations of that standard are available from a variety of vendors targeting different programming languages, operating systems, network protocols, etc. For similar reasons J2EE can be considered a relatively “independent” platform.

There is certainly a trade off involved between a greater degree of platform independence on the one hand, and the cost of developing and maintaining a platform independent architectural layer on the other. Note however that MDA changes the terms of this discussion by providing the option of automatically generating the platform specific models associated with these services. Note that the

diagram does not include a representation of a Computation Independent Model (CIM) because the focus of the diagram is on transformations that are closer to the final implementation platforms.

Step 1 – Generating the first layer of abstraction

From the core, a first level of transformations target four key viewpoints of the system: the User Interface model, the Information/Data model, the Web Services model (system-to-system interfaces) and the Computational model, which holds the “middle-tier” business logic.

The key driver for mapping each of these platforms from the same core model is to ensure that all of its rules and constraints are consistently applied across the different target interfaces. If, for example, a business rule dictates that a certain property must be a numeric value between 0 and 100, then this requirement should be enforced regardless of whether it is updated via the User Interface, a database transaction, or an XML file upload.

While the target models at this level of abstraction are still technology independent they can be viewed as platform specific from the perspective of this transformation.

Each of these target transformation models may contain a mix of both business elements and architectural aspects relevant to that platform (user interface, data structures, etc.). The distinction between the two facets may not be as clean cut as implied by the diagram because architectural styles and patterns are often embedded in the transformation rules and marks of a platform. Thus, for example, a transformation based on a 3-tier architecture applied for the creation of a user interface model will yield different results than a transformation defined for a 4-tier architecture (one with an explicit *workspace* layer) applied against the same core PIM model. The differences in the resulting models will find their cause in the different mapping rules embedded in each transformation.

Step 2 – Generating the second layer of abstraction

Once the models created in step one are ready to be transformed into technology specific platforms each one then takes on the role of a PIM. The transformation process is now repeated with different rules and marks being applied to yield a new set of target models, each of which is specific to a technology platform. The same remark about the mix of business and architectural aspects applies here as well.

Step 3 – Generating the implementation artifacts

The final step, not expounded here, consists of generating the code and other artifacts necessary to deploy and execute the various components of the application(s) from the lowest levels of platform specific models.

Additional viewpoints not shown in the above diagram but which may be favorably considered as targets for MDA transformations are the Testing Model and the Deployment Model.

Minimum tool requirements for MDA support

Keeping in mind the MDA features as exposed in the previous sections, what would constitute the minimal set of requirements that a modeling tool needs to satisfy in order to qualify as being “MDA compliant”?

To date, the OMG has not defined a formal MDA certification process for tool vendors. Luckily Michael Guttman, one of the early MDA evangelists, has addressed this very question in a magazine article¹. The following represents a summary of his criteria:

- The ability to exchange models with other tools, including from different vendors, using one or more of the standardized MOF mappings: XML (XMI), Java (JMI) or CORBA.
- The use of MOF/XMI internally, within the different components of the tool (or tool suite).
- The ability to make the model-to-model transformations traceable, and thus by implication the ability to repeatedly synchronize the source and target models.
- Supporting the Query/View/Transformation (QVT) OMG standard which stipulates not only how model transformations are specified, but also how they can actually be modeled.
- Full support for all of the UML 2.0 features and OMG adopted UML Profiles.

To this essential list one may add the following features:

- Forward and reverse engineering of code in at least one programming language. Similar forward and reverse engineering capabilities in these additional areas is also a great plus:
 - Database schemas.
 - XML schemas.
 - WSDL declarations.
 - Common user interface implementation modules such as Java Server Pages (JSP) and Active Server Pages (ASP).
- Support for OCL with at a minimum syntax validation. But ideally with:
 - Validation against the model (i.e. validating that the semantics of the OCL expressions are accurately stated in terms of the model elements being referenced).
 - Automated generation of constraint-checking code from the OCL, to be included in the test and/or live version of the implementation. This can cut down significantly the manual coding effort needed.
- The ability to create and persist a custom MOF-based metamodel. Additionally, the ability to create an instance model of the custom metamodel (i.e. a model which is fully compliant with all the constraints specified by that metamodel).

¹ <http://www.softwaremag.com/L.cfm?Doc=2005-04/2005-04mdatools>

- The ability to “attach” and “detach” [marks](#) from a PIM (marks typically appear in the form of stereotypes, tagged values and constraints). Since marks clearly belong to the target PSM domain, this capability allows the PIM to remain truly platform neutral, and supports the option of applying multiple PSM marks against the same PIM.
- Comprehensive model validation so that the tool can ensure that all of the guidelines set forth for [building computationally complete models](#) are followed.
- Support for a concrete implementation of the abstract Action Semantics defined in UML. A welcome extension would be the mapping of this concrete syntax to at least one common programming language (e.g. Java or C#).
- The ability to identify and create UML patterns and to apply these patterns against model elements.

The Enterprise Architect solution for applying MDA

After gaining an understanding of what it means for a tool to support MDA, we can now take a look at the hottest UML modeling product currently on the market : *Enterprise Architect* (EA) from Sparx Systems². For a complete list of all of the product's features please visit the Sparx website. Here the focus is primarily on those capabilities which are relevant to the support of MDA.

Product history

Enterprise Architect is a mature UML 2.0 based modeling tool for the Windows platform (with a version for Linux running under *Cross-Over Office*) which has a decade of commercial history behind it, and a track record of fast evolution and impressive innovation.

Commitment to MDA and OMG standards in general

Sparx Systems has at numerous times stated their strong commitment to MDA and its underlying OMG standards such as UML, MOF and XMI. Its mission statement is quoted here:

To provide an affordable, high-quality, team based modeling environment founded on the UML 2.0 specification, with comprehensive support for model to model transformations as well as model driven generation of common development artifacts such as documentation, source code, test scripts, deployment descriptors, XML schemas, database schemas, etc.

Out of the box, *Enterprise Architect* offers a number of features targeted at MDA driven development, including a model-to-model transformation engine, allowing modelers to target multiple platform specific models from a single PIM – and to synchronize PIM changes into each PSM on demand. The built-in transformation templates include mappings to C#, DDL, EJB, Java, JUnit, NUnit, WSDL and XSD (XML Schema).

Testing our MDA compliance criteria

The set of criteria for [MDA compliance](#) defined earlier can now be used to determine how well *Enterprise Architect* measures up. This comparison is based on the feature set of EA version 7.5.

MDA Feature	✓	Comments
MOF-XML mapping (XMI)	✓	
MOF-Java mapping (JMI)	-	However a COM Automation API for Java is available.
MOF-CORBA mapping	-	
Internal use of MOF/XMI	✓	Entire models, package hierarchies and

² <http://www.sparxsystems.com/>

MDA Feature	√	Comments
		individual packages can be saved as XMI files for import/export as well as for configuration management purposes.
Traceability and synchronization of model-to-model transformations	√	Changes to a PIM can be synchronized into one or more PSMs as needed, maintaining a tight coupling between the two.
Support for QVT	-	EA provides its own proprietary transformation language.
Complete UML 2.0 support	√	
Support for UML Profiles	√	Allows the user to import predefined (standard) profiles and to create her/his own.
Forward and reverse engineering		
Programming languages	√	Supports Java, C++, C#, VB, & others. Code generation templates can be customized by the user.
Database schemas	√	SQL Server, Oracle, My SQL,...
XML Schema	√	
WSDL	√	
JSP, ASP, others	-	No standard currently defined. Too many UI technologies in use...
Code generation from behavioral models	√	Within constraints, supports generation from Activity and Sequence diagrams as well as State Machines.
Support for OCL		
Syntax checking	√	
Model validation	√	Very limited!
Automated generation of code	-	
Support for MOF		
Create & persist MOF models	√	
Create MOF model instances	-	
Attach & detach marks from a PIM	-	
User extensible model validation	√	
Support for Action Semantics		
Mapped to a concrete syntax	-	Informally, this feature is available via code generation from UML behavioral models.
Mapped to a 4GL	-	Ditto.

MDA Feature	√	Comments
Create and apply UML patterns	√	
Ratio of implemented features	15/24	

Conclusion

The conclusion is clear: *Enterprise Architect* satisfies the core requirements for being considered MDA compliant, as well as many of the additional features that have been identified in this document.

As a roadmap for an even more extensive compliance in a future release, the most important capabilities that are missing at this time can be identified as:

1. Full validation of OCL statements against model elements.
2. Automated generation of code from OCL constraints.
3. Support for a concrete implementation of the Action Semantics Language.
4. Support for QVT.

However *Enterprise Architect* already offers a sufficient range of features for the successful implementation of an MDA based development approach.

The model-to-model transformation engine augmented with customizable code generation templates, the ability to leverage UML profiles and patterns, as well as the integrated API for scripting and extending the tool itself all make *Enterprise Architect* a compelling MDA solution.

References

ftp://ftp.omg.org/pub/docs/ab/01-02-04.pdf	MDA – A Technical Perspective by the OMG Architecture Board
http://www.omg.org/docs/omg/00-11-05.pdf	MDA White Paper by Richard Soley and the OMG Staff Strategy Group
http://www.omg.org/docs/ormsc/05-04-01.pdf	A Proposal for an MDA Foundation Model (An ORMSC White Paper)
http://www.omg.org/docs/omg/03-06-01.pdf	MDA Guide Version 1.0.1
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

For a summary of MDA resources and MDA style transformations in EA see:

<http://sparxsystems.com.au/resources/mda/index.html>

For an overview of writing transformations using EA see:

http://sparxsystems.com.au/resources/mda/writing_transformations.html

About SPARX Systems

**COMPANY
BACKGROUND**

Established in 1996 by Geoffrey Sparks, Sparx Systems is an Australian company based at Creswick, near Ballarat, Victoria. With over a decade invested in the development of Enterprise Architect, the company's motivated team of engineers are dedicated to the ongoing development and support of software tools, object-oriented methodologies and CASE tools.

**COMPANY
VISION**

Sparx Systems aims to satisfy the growing needs of the software and business development industry by providing immediate delivery and ongoing support of affordable, productive and user-friendly business/system design software.

Sparx Systems believes that a complete modeling and design tool should be used throughout the full process/software lifecycle. Our subscription plan reflects this, and our belief that "life-cycle" software should be as dynamic and modern as the systems you design and maintain.

Sparx software is intended for use by analysts, designers, architects, developers, testers, project managers and maintenance staff - almost everyone involved in a software development project and in business analysis. It is Sparx Systems' belief that highly priced CASE tools severely limit their usefulness in a team, and ultimately to an organization, by narrowing the effective user base and restricting easy access to the model and the development tool. To this end, Sparx Systems are committed to both maintaining an accessible pricing model and to distributing a 'Read Only' (EA Lite) version of EA for use by those who only need to view modeling information.

USER BASE

Sparx software is utilized by a wide variety of companies ranging from large, well-known, multinational organizations to many smaller independent companies and consultants. The Sparx discussion forum confirms a solid and active user base.

Sparx software is used for the development of various kinds of software systems for a wide range of industries, including: aerospace, banking, web development, engineering, finance, medicine, military, research, academia, transport, retail, utilities (gas, electricity etc.), electrical engineering and many more. It is also used effectively for UML and business architecture training purposes in many prominent colleges, education facilities and universities around the world.

**CONTACT
DETAILS**

Website : <http://www.sparxsystems.com>

Sparx Systems can be contacted at the following email addresses:

Sales inquiries: sales@sparxsystems.com.au

Support inquiries: support@sparxsystems.com.au

About Cephass Consulting Corp.

Since 2001, Cephass 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.

**C O M P A N Y
B A C K G R O U N D**

Cephass specializes in introducing modeling practices into organizations via training and mentoring. The team of consultants and architects at Cephass 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.

**C O M P A N Y
F O C U S**

Cephass Consulting has the required expertise to lead organizations into the use of Model Driven Architecture. As early adopters we have successfully helped a number of clients implement MDA. 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.

**C O M M I T M E N T
T O T H E O M G
A N D M D A**

Website : <http://www.cephass.cc>

General inquiries: cephass.contact@cephass.cc

Author inquiries: frank.truyen@cephass.cc

EA license purchase inquiries: cephass.license@cephass.cc

**C O N T A C T
D E T A I L S**
