Implementing Model Driven Architecture using

Enterprise Architect

Mapping MDA Concepts to EA Features

By Frank Truyen

frank.truyen@cephas.cc

for

Sparx Systems Pty Ltd

Version 1.1

Table of Contents

Model Driven Architecture and Enterprise Architect

Abstract

The goal of this document is to define Model Driven Architecture in terms of:

its major concepts,

the promises and goals which drive its adoption,

its abstract development process,

its concrete implementation.

Next under review are the minimal as well as optional requirements which a tool should satisfy in order to qualify as being "MDA compliant".

***Enterprise Architect* from Sparx Systems is then measured against this set of requirements and its specific MDA implementation features are analyzed.**

A separate paper (MDA in Practice) uses a running example to illustrate a set of MDA transformations starting from a simple platform independent model.

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.

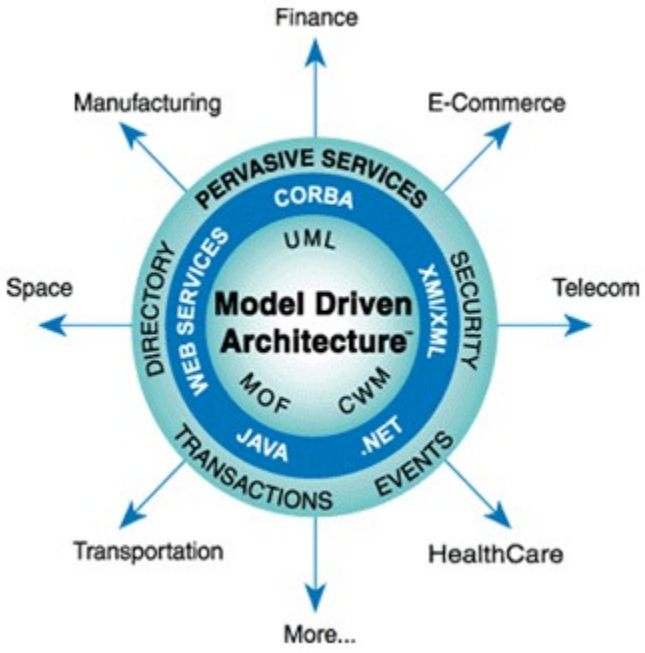
MDA in context

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/or 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].

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.

Computation independent

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

Platform independent

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 which can be abstracted out of that platform.

Platform specific

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 which 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 which 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).

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

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 which 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 which 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.
- **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.

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

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

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.

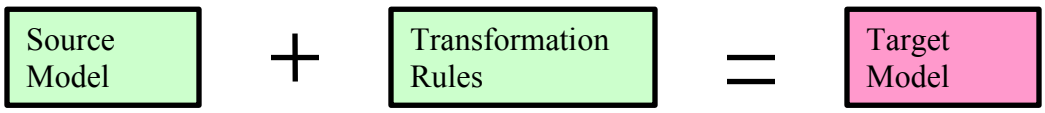
Applying a consistent architectural style across viewpoints of the system is one illustration of such a horizontal transformation. Other examples include:

- 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 will 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

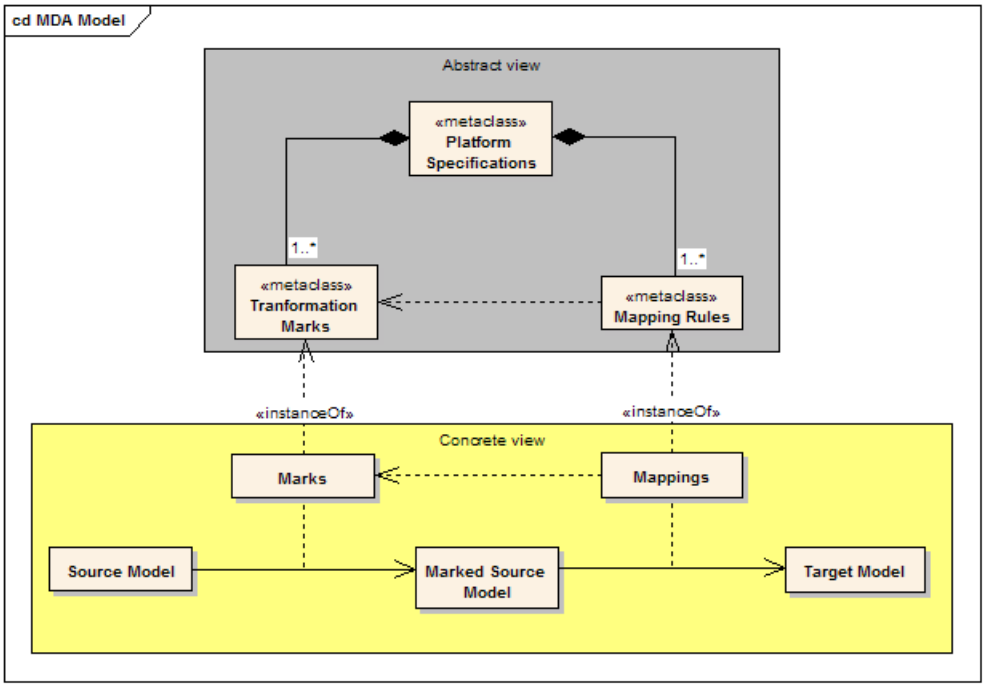
A model transformation mapping must be specified using some language, be it a natural language, an action language, or a dedicated mapping language. The OMG is currently in the process of adopting the MOF Query/View/Transformation (QVT) specification as a portable mapping language.

Recording the transformations

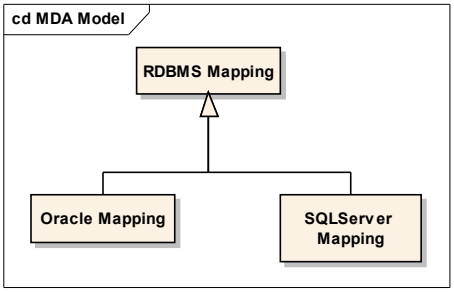
A record of the transformation should include a chart indicating which PIM elements were mapped to which PSM elements and include the mapping rule/s used for each part of the transformation.

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 will vary 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.

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, nor are there currently 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 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.

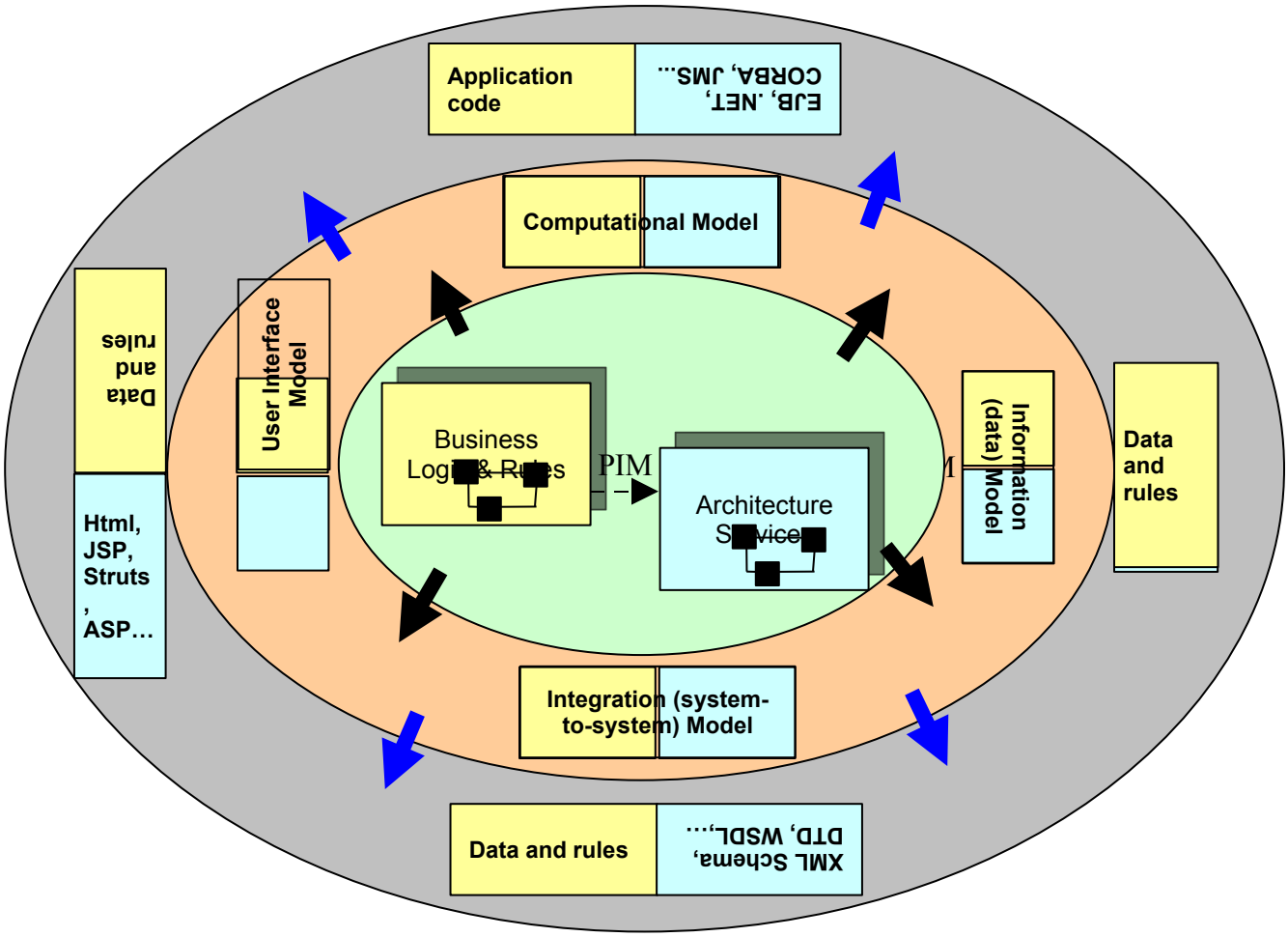
  DBC constraints typically map to exceptions, however defined in a specific programming language. They are also crucial for the manual or automated development of test harnesses.
- Adorn UML State machines describing the internal behavior of an element, for example a class, with constraints in the form 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.
- With regards to PIM class diagrams:
    - 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.
- Properly 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 between classes or components where an implementation level dependency exists which is not otherwise expressed in the model. 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 reside a set of Platform Independent Models capturing:

- The business logic and rules.
- 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 business components and applications. Examples include an Event Service, a Persistence Service, a Transaction Service, a Security Service, a Notification Service and a Directory Service. 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, not that such a model is not important but 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 can occur targeting 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 (holding 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.
- 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"?

The OMG is in the early process of defining a tool certification process, but no official document exists on the topic at the time of this writing. Luckily Michael Guttman, director of the OMG's MDA FastStart[4] program, has addressed this very question in a recent article[5]. 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.
- The commitment to support the forthcoming Query/View/Transformation (QVT) OMG standard which intends to stipulate not only how all 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).
- 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.

[4] http://www.omg.org/faststart/index.htm
[5] http://www.softwaremag.com/L.cfm?Doc=2005-04/2005-04mdatools

- 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 Systems[6]. 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 been improved over the course of seven years, with a five year commercial history and 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 6.0* 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 6.0.

MDA Feature	√	Comments
MOF-XML mapping (XMI)	√	
MOF-Java mapping (JMI)	-	
MOF-CORBA mapping	-	
Internal use of MOF/XMI	√	Entire models, package hierarchies and 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 ore more PSMs as needed, maintaining a tight coupling between

[6] http://www.sparxsystems.com/

		the two.
Commitment to implement QVT	√	Waiting for the final release of QVT as an approved OMG standard.
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 of:		
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 technologies in use.
Support for OCL:		
Syntax checking	√	
Model validation	√	Partial as of this release
Automated generation of code	-	Planned for a future release
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	-	
Mapped to a 4GL	-	
Create and apply UML patterns	√	
Ratio of implemented features	15/23	

Conclusion

The conclusion is obvious : *Enterprise Architect* satisfies all of 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 three 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. In the interim, support for specifying business logic in the two most commonly used programming languages: Java and C#.

Version 6.0 of *Enterprise Architect* offers a sufficient range of tools 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 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 seven years' investment in the development of Enterprise Architect, the company's motivated team of employees 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:

General Enquiries: sparks@sparxsystems.com.au

Support Enquiries: support@sparxsystems.com.au

Sales and Purchase Enquiries: **sales@sparxsystems.com.au**

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 techniques, using *Enterprise Architect* as the primary tool for capturing both business and system domain information.

The team of software engineers and architects at Cephas combines many years of experience, allowing it to offer a one-stop solution addressing all aspects of managing the enterprise meta-data using *Enterprise Architect*:

- Training & mentoring from beginner to expert level.
- Migrating meta-data out of legacy tools.
- Establishing onsite guardianship of the tool, including configuration and replication management.
- Customizing the 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

Cephas Consulting can be contacted at the following email addresses:

General enquiries: cephas.contact@cephas.cc

Author enquiries: frank.truyen@cephas.cc

