Introduction to Model Driven Architecture

Mike Rosen
CTO, M²VP
Mrosen@m2vp.com
Model Driven Architecture

- A set of standards defining the scope, content, creation and usage of models
- An architecture-based process for integrating models into the development process
- Formally separates business and technology concerns
Courtesy Reminder

- Cell phone and pagers to silent, Thanks
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA standards
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
Today’s IT Challenges

- Alignment of business requirements with IT systems intended to meet those requirements
- The technology that supports those systems is constantly evolving
- IT must support rapid revenue growth and acquisition strategies
- And move beyond the limitations of stovepipe and point-to-point systems
- While leveraging existing applications
- Within time-to-market and budget constraints
Technology Independence

- **Requirement**: Preserve application investment
  - As platforms proliferate
  - As platforms themselves change
    - MTS \(\rightarrow\) COM+ \(\rightarrow\) .NET
    - EJB 1.1 \(\rightarrow\) EJB 2.0 \(\rightarrow\) EJB 2.1
    - XML DTD \(\rightarrow\) XML Schema
    - CORBA 2.X \(\rightarrow\) CORBA 3.0

- **Solution**: Isolate information and processing logic from technology specifics
  - Build platform-independent models
  - Map these models to specific platforms
  - Maintain the separation at the implementation level
Raising the Level of Abstraction

- A logical extension of proven techniques
  - Programming Languages
    - Bits, assembler, 3GL, OO, 4GL
    - Operating Systems, Middleware...
  - Already well-established for front and back ends
    - WYSIWYG GUI modeling
    - Data modeling
    - Hand coding no longer predominates
- Working at higher levels of abstraction increases our value
  - Higher productivity
  - More time focused on solving business problems, rather than technology details
MDA Complements Existing Approaches

- Technology churn
- Integration
- App Development
- Skills
- New application areas
- Legacy Systems
- IT Complexity
- Architectures, Architectural Styles
- Product Line* approach
- CBD
- Traceability Business->Code
- EAI
- MDA
Agenda

- Why MDA?
- A few words about architecture
  - MDA concepts
  - MDA and standards
  - Models and transformations
  - MDA development process
  - MDA tools
- Conclusion
Architecture Principles

- Separation of Concerns (viewpoints) - The architecture should separate the requirements and concerns of different constituents.
  - Business
  - Technical
  - Physical
  - Implementation
  - There must also be relationships and traceability between different viewpoints

- Enable consistent and effective applications development
Architecture Principles

- Accommodate the future
  - Future versions
  - Planned enhancement
  - New technologies

- Rigorous

- Well Documented - Expressed in industry standard UML notation

- Phased Implementation
Enterprise Architecture:
Describes concerns and guidelines for integration of process and data across the entire enterprise

Application Architecture:
Describes techniques, frameworks, guidelines within an application domain

Application:
Implementation of a specific application
Example: 4+1 Architectural Views

Logical View
Object Model
• Class Diagrams
• Collaboration Diagrams
• Sequence Diagrams

Component View
Files
Dependencies
• Component Diagrams

Process View
Processes
Threads
• Deployment Diagrams

Physical View
Network Topology
Example 2: Architecture Driven Design

**Business Domain** → **Business Model**

**Program Requirements** → **Program Plan**

**Technical Requirements** → **Application Architecture**

**Operational Requirements** → **Operational Architecture**

**IT Analysis Model (High Level)**

**Platform Independent Design Model (Detailed)**

**Platform Specific Model (Implementation)** → **Deployed Component**

**Product Requirements** → **Technology Requirements** → **Implementation Architecture**
Example 3: RM-ODP

- Reference Model for Open Distributed Processing – ISO Standard
- Transparencies
  - Access, Failure, Location, Migration, Persistence, Relocation, Replication, Transaction
- Viewpoints
  - RM-ODP prescribes five viewpoints as necessary and sufficient to describe the model of a system
    - Enterprise Viewpoint
    - Information Viewpoint
    - Computational Viewpoint
    - Engineering Viewpoint
    - Technical Viewpoint
Architecture Example

Summary

- 4+1 Views
  - Separation of concerns
  - Central, unifying viewpoint with traceability

- Architecture Driven Design
  - Architecture process spans application development
  - Separation of concerns addressed by specialized models
  - Business/Domain models drive implementation models
  - Platform concerns addressed independently from business solution

- RM-ODP
  - Formal definition of system viewpoints
Model Driven Architecture

- A standardization of best practices within enterprise architecture and development
- A process for the creation and usage of models as first class development artifacts
- Defines process for model development and refinement within context of enterprise architecture
- A roadmap for consistently applying modeling to enterprise solutions
- Formalizes separation of concerns into specific models
  - Expresses business concepts in formal business model
  - Expresses business services in platform independent IT model
Agenda

- Why MDA?
- A few words about architecture
  - MDA concepts
- MDA and standards
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
Basic Concepts

- **Model** - a representation of the system
  - A model describes part of the function, structure and/or behavior of a system.

- **Formal Model** – a representation of the system conforming to rigorous rules
  - A model is said to be formal when it is based on a language that has a well defined form ("syntax"), meaning ("semantics"), and possibly rules of analysis, inference, or proof for its constructs. The syntax may be graphical or textual. An MDA model must be paired unambiguously with a definition of the modeling language syntax and semantics, as provided by the MOF.

**All MDA models are formal.**
Basic Concepts 2

- **Abstraction** - the suppression of irrelevant detail
  - It is useful to characterize models in terms of the abstraction criteria that were used to determine what is included in the model. A model that is based on specific abstraction criteria is often referred to as a *model from the viewpoint defined by those criteria*.

- **Refinement** - the addition of specific detail (usually associated with reduction of scope)
  - Some pairs of models are in a *refinement* relationship in which one – the *abstraction* - is more abstract than the other - the *realization*.

- **ViewPoint** – the presentation of a system for a specific audience
  - Each viewpoint, focusing on one aspect of a system, abstracts out details that are not of primary concern to the given viewpoint.
Basic Concepts 3

- **Platform** – the technology implementation of a running system, e.g. J2EE, .NET
  - Refers to technological and engineering details that are irrelevant to the business functionality of a software component.

- **PIM** - *Platform-Independent Model*
  - A formal specification of the structure and function of a system that abstracts away technical detail.

- **PSM** – *Platform-Specific Model*
  - A refinement of a PIM which expresses the system in terms of the *specification model* of the target platform.
Basic Concepts 4

- **CIM** — *Computationally-Independent Model*
  - A higher level abstraction of a PIM which contains only business concerns.

- **Mapping / Transformation** — controls the conversion of models
  - A mapping defines a formal algorithm and method for converting one type of model to another — a transformation is the process of applying the mapping.

- **Traceability** — the ability to relate an element in one model to another model
  - Ties an element in one model to it’s transformation in another across different levels of abstraction.
MDA Mappings

- **Computation Independent Business Model**
  - PIM → PIM Mapping

- **Platform Independent Analysis Model**
  - PIM → PSM Mapping

- **Platform Specific Design Model**
  - PSM → PSM Mapping

- **Code**
  - PSM → Code Mapping
Consistent Model Separation

Service Provisioning
- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

Billing
- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

CIM
PIM
PSM
Consistent Relationships

Service Provisioning

- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

Billing

- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

CIM
PIM
PSM
Consistent / Shared Mappings

Service Provisioning

- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

Billing

- Computation Independent Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model

Platform

CIM
PIM
PSM

Copyright © M²VP Inc, 2003. All rights reserved
Mapping Example
PIM / PSM Advantages

- Easier to validate correctness of Model
- Easier to produce implementations on multiple platforms
- Integration / interoperability across platforms better defined
- Generic mappings / patterns can be shared by many designs
MDA Process
Generation Capabilities

- Tools are standards based, not proprietary
- Resulting code base doesn’t require a specific runtime infrastructure
- 70-80% of the structural code can be generated
- Test Cases can be generated from OCL

**Business Model**

**Platform Independent Model**

**Platform Specific Model**

**Tools are standards based, not proprietary**

**Resulting code base doesn’t require a specific runtime infrastructure**

**70-80% of the structural code can be generated**

**Test Cases can be generated from OCL**
Technology Independence

- Applications are “Future-Proof” against technology churn
- When technology evolves, a new PSM can be generated rather than rewriting it
Multi-Platform Artifacts

- Artifacts can be generated for multiple platforms from the same design

Business Model

Platform Independent Model

EJB Design Model

.NET Design Model
Integrate Existing Assets

- MDA accelerates the integration of new services based on existing applications
- Business logic can be precisely expressed in business models
- New business objects can bridge to existing applications

**Business Model**

**Platform Independent Model**

**Platform Specific Model**

**Code**
MDA Under the Hood

MDA explicitly supports Enterprise Architecture

- Business Model
- Platform Independent Analysis Model
- Platform Specific Design Model
- Code

Enterprise QoS and non-functional requirements implemented in transformations

Architectural Standards and Guidelines Enforced in Model Profiles
APSL Process

1. Define the approach
   - Integrate enterprise architecture into the development process.
   - Create meta-models and profiles

2. Define the problem
   - Create Business Models (Domain, CIM, System)

3. Define the solution
   - Refine into PIMs and PSMs

4. Leverage the solution
   - Integrate assets into a reuse repository
   - Architecture and design accommodates: reuse, customization, enhancements, versioning…
MDA Benefits

- Better alignment of business requirements and IT implementations
- Unambiguous description (and validation) of business services
- Clear process and guidelines for producing and consuming models
- Integration of architectural standards
- “Future proofing” against technology changes
- Reduction of low value tasks
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA core technologies
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
MDA Core Technologies

- UML – Unified Modeling Language
  - OCL – Object Constraint Language
- MOF – MetaObject Facility
- CWM – Common Warehouse Metamodel
- XMI – XML Metadata Interchange
UML

- UML models are declarative, with the following advantages:
  - UML models are semantically rich
    - Invariants
    - Pre and Post conditions
    - Legal values (null?)
    - Operation side effects
    - Whether subtypes are disjoint or partition
    - Patterns of specification, design and refinement
  - UML is defined using core UML / MOF concepts
  - UML can be visual and/or textual
- UML (Unified Modeling Language) is an OMG Standard
OCL

- Object Constraint Language
- Design By Contract
  - Provides precise instruction to the programmer
  - Improves interoperability between different implementations of the same specification
  - Acts as a basis for conformance tests
  - Can provide input to code / test generation
So, What’s a Metamodel

Meta-Metamodel

Meta-class

Metamodel

Class

Association + Assn. Ends

Aggregation

Model

Account

Meta-class

Model

Account

Meta-class

Instance

MyAccount

YourAccount

Interesting for developing systems

Interesting for developing tools

MOF

M3

UML

M2

Model

M1

Model

M0
MOF – Meta Object Facility

- A consistent way to define metamodels
- A consistent way to store models
- A consistent way to access models
- Rules for defining lifecycle, composition and closure of elements
- A hierarchy of reflective interfaces
- Provides interoperability between otherwise dissimilar models and metamodels
MOF Repository

- MOF CORBA Interfaces
- MOF Java Interfaces (JMI)
- MOF XML (XMI) Import / Export

Diagram:
- UML Models
- Data Models
- Process Models
- CCM CORBA Interfaces
- B2B Choreography Descriptions

Courtesy David Frankel
MOF Components

- Discover & Manipulate metadata
- Find and Manage Metadata Repositories
- MOF Model
- MOF Facility
- MOF 1.3 Reflective

<<OMG Metamodel>>

Model using UML Class Diagrams precisely

<<Interfaces>>
Common Warehouse Metamodel

- Standard for data warehousing
  - Business and technical metadata for business analysis and data warehousing

- Covers full data lifecycle
  - Design
  - Build
  - Manage

- Improves integration between development and deployment
  - Models that span lifecycle segments
  - Profiles that map between segments

- Integration between heterogeneous stores
Common Warehouse Metamodel

Warehouse Management

Analysis

Resources

Foundation

Warehouse Process

Warehouse Operation

Transformation

OLAP

Data Mining

Information Visualization

Business Nomenclature

Object-Oriented
(_OBJECTModel)

Relational

Record-Oriented

Multi Dimensional

XML

Business Information

Data Types

Expressions

Keys

Type Mapping

Software Deployment

ObjectModel

(Core, Behavioral, Relationships, Instance)

Courtesy OMG, ©2002
CWM Package Architecture

Modular Design

– Minimum dependencies
  • Cross package services
    provided by links to UML
– Avoid subpackages
– Reduced complexity,
  improved understanding
– Use only the packages
  you need
XMI

- Standard interchange mechanism
  - Tools
  - Repository
  - Middleware
- Produce XML Schema and DTD’s
- Serialize via XML documents
- Applies to any MOF repository
Objects and Designs

Model in XMI

XMI DTD, Schema

XMI Document

Model Interchange

Instance Interchange

Public class Auto {
public color color;
public int door; }

<Auto>
  <Color>red</Color>
  <Door>2</Door>
</Auto>
XMI in MDA

MDA Model

Metamodel

XML Document

XML Schema

conform

conform

XMI

XMI

XMI

M²VP
Java and MDA

- Several J2EE standards are formal mappings of MDA into Java
  - JMI
  - JOLAP
  - JDM

- A trend toward the realization of MDA, MOF, CWM interfaces in Java
OMG’s Model Driven Architecture
Technology Relationships
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA and standards
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
Informal UML Models

- Informal modeling
- Used to sketch out basic concepts
- Advantage over typical box and line diagrams because shapes and line types have specific meanings
- Important way to use UML, but can’t drive code generators and dynamic execution engines
  - Analogously, informal text can’t be compiled and executed like 3GL text
Formal UML Models

- **Precise**
  - Precision and detail are *not* the same!

- **Complete**
  - Missing properties and unresolved references not acceptable
  - 3GL analogy…
    - an incomplete expression such as “a +” does not compile
    - An undeclared identifier does not compile
Computationally Complete Model

- A model that can be executed - via code generation or interpretation - is said to be “computationally complete”

- Requires:
  - Action Language for algorithmic logic
  - Computational structure

- The vision is to build computationally complete PIMs
  - All development at the model level
  - Execute the model to test
  - Generate code where necessary
Business Information Model
Imprecise and Incomplete

<Account>
  id : String
  balance : Float

<SavingsAccount>
  interestRate : Float

<CheckingAccount>
  minBalance : Float

<PreferredChecking>

<Customer>
  socialSecurityNum : String
  name : String
  address : String
Business Information Model
Precise and Complete

context Account inv:
--The first character of the id must be the same as the first character of the customer name
id->substring(1,1) = customer.name -> substring(1,1)

context PreferredChecking inv:
--Cannot go below the minBalance
balance >= minBalance

<<BusinessEntity>>
Account
id : String
balance : Float

<<BusinessEntity>>
SavingsAccount
interestRate : Float

<<BusinessEntity>>
CheckingAccount
minBalance : Float

<<BusinessEntity>>
PreferredChecking

<<BusinessEntity>>
Customer
socialSecurityNum : String
name : String
address : String

1..n +account

1 +customer

{disjoint}
Disjoint means no instance can be an instance of both subclasses.

context Account inv:
--The first character of the id must be the same as the first character of the customer name
id->substring(1,1) = customer.name -> substring(1,1)

Invariant rules expressed in UML’s Object Constraint Language (OCL)

class Account {
    id : String; balance : Float;
}

class CheckingAccount {
    minBalance : Float;
}

class SavingsAccount {
    interestRate : Float;
}

class Customer {
    socialSecurityNum : String; name : String; address : String;
}

class PreferredChecking {
    --Cannot go below the minBalance
    balance >= minBalance
}

Multiplicity could be 1 or 0..1, must be specified

Composition of Account by Customer formally captures an important business rule: An account cannot be transferred from one customer to another.
FundsXFerFromChecking(in fromAcct : CheckingAccount, in toAcct : SavingsAccount, amount : Float, out fromAcctBal : Float, out toAcctBal : Float)

class FundsXFer (XFerFromChecking)

--Pre and post conditions

pre:
  {fromAcct.balance >= amount}
  {fromAccount.customer = toAccount.customer}

post:
  {fromAcct.balance = fromAcct.balance@pre - amount}
  {toAcct.balance = toAcct.balance@pre + amount}
  {fromAcctBal = fromAcct.balance}
  {toAcctBal = toAcct.balance}
PSM

- Expressed in UML
  - UML is platform neutral, so...
- UML Profiles extend UML using stereotypes and tagged values
  - Create a Platform Specific Template
Mapping the Business Information Model to XML

Platform-Independent Model

XMI’s UML-XML Mapping Rules

Platform-Independent Model

XML DTD (or Schema)

<!ELEMENT Bank.Customer.name (#PCDATA | XMI.reference)*>
<!ELEMENT Bank.Customer.Address (#PCDATA | XMI.reference)*>

<<BusinessEntity>>
Customer
SocialSecurityNum : String
name : String
Address : String

...
The message payload format is based on a UML-XML mapping applied to the business information model.
Modeling System Behavior

- **Functional Behavior**
  - What a component must do to satisfy its functional (business) requirements.
  - This type of behavior is the concern of the business expert, the business analyst, and the functional tester.

- **Non-functional Behavior**
  - What a component must do to satisfy its non-functional requirements.
  - This is the concern of the designer who looks for ways to re-assemble components to generally maximize cohesion, minimize coupling, and to specifically increase throughput, availability, changeability, and other “-ilities” that are required of the system.
Semantic Behavior

– What a “component” must do to interact with other components within its environment.
– A specialized form of non-functional behavior that is related to architectural style
– This is the concern of the system architect.

Idiomatic Behavior

– What a component must do to operate within the environment of its technical platform/programming language.
– This is the concern of the programmer.
MDA & Separation of Concerns

- Provides separation of concerns by defining…
  - Platform Independent Models (PIM)
    - Provides formal specifications of the structure and function of the system that abstracts away technical details.
    - Details functional, non-functional, and semantic behavior
  - Platform Specific Models (PSM)
    - Provides idiomatic behavior expressed in terms of the specification model of the target platform.
    - PSMs have to use the platform concepts of exception mechanisms, parameter types (including platform-specific rules about objects references, value types, semantics of call by value, etc.) and component model.
Architecture vs. Architectural Style

- **Model Driven Architecture**
  - We’ve talked about models.
  - So where’s the architecture?

- **Architecture**
  - The highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or **structure of significant components** interacting through interfaces, those components being composed of successively smaller components and interfaces.*
  - Architecture, then, is dependent upon the specifics of the system it describes, i.e. its major components, interfaces, and constraints.
  - Two companies in the same line of business using the same development and deployment platforms will have different architectures.

*Rational Unified Process
Architecture vs. Architectural Style

- Architectural Style
  - “A description of component types and a pattern of their runtime control and/or data transfer. A style can be thought of as a set of constraints on an architecture – constraints on the component types and their patterns of interaction – and these constraints define a set or family of architectures that satisfy them.”*
  - Architectural style, then, provides the “rules of engagement” when building an architecture. It is a set of patterns that provide guidance on the proper use of the different types of components that exist within your architecture.
  - The best way to convey an architectural style is via a formal model of it, referred to as a metamodel.

Metamodels

- Provide rules for how to build a correct model for a particular purpose, e.g. “business integration metamodel”

- **UML Profile**
  - Provides a targeted subset of UML
  - Standard mechanism for extending UML

- **Refinement and Constraint**
  - Metamodels refine the definition of modeling elements by placing constraints on their behavior through the use of stereotypes

- **Stereotypes**
  - Standard UML Stereotypes
    - <<boundary>>, <<control>>, <<entity>>
  - Extending the UML Stereotypes
    - Inheritance used to extend and refine the meaning of stereotypes
    - Tagged Values use to apply specific properties
Sample Web Implementation
Sample Web Architecture
Metamodel Stereotypes

Standard UML Stereotypes

Class Diagram: Metamodel / Metaclass Interactions

Usage

Scope and visibility

Location and purpose

Behavior

{state & composition control}

<<stereotype>>
boundary

<<stereotype>>
presentation client

<<stereotype>>
control

<<stereotype>>
entity

<<stereotype>>
business entity

<<stereotype>>
data

<<stereotype>>
session context

<<stereotype>>
state

{transient}
{long lived}
{persistent}

Courtesy Terry Merriman, ©2003
Web Architectural Style Metamodel
“Rules of Engagement”

**Presentation Tier**
- Presentation controller
  - Send data to user
  - Post updates
- Presentation client
  - Display information
  - Receive user's input

**Application Tier**
- Presentation controller
  - Format data for presentation
  - Control local navigation
- Application session
  - Control the flow of the user session
  - Massage data for viewing or processing
- Application service
  - Access back end services
  - Performed shared application logic
  - Expose application logic

**Enterprise Tier**
- Business service
  - Expose business functionality
  - Call domain services to do the processing
  - Aggregate results of Domain Services
- Domain service
  - Process domain specific algorithms
  - Control resources of the domain
  - Visible only within the domain
- System entity
  - Persists data beyond a session
  - Has behavior specific only to its internal state

**IT Services**
- IT service
  - Security
  - Logging
  - Profiling
  - Persistence
  - Transactions etc.

**Resource Tier**
- Technology Resource
  - RDBMS
  - Rules Engine
  - Transaction Monitor etc.

**Metaclass Interactions**
- Presentation Tier
- Application Tier
- Enterprise Tier
- IT Services
- Resource Tier

**Technology Requests**
- IT service requests
- Technology requests

**IT Service Requests**
- Technology requests
- Interaction with technology products

**System Entity**
- Persists data beyond a session
- Has behavior specific only to its internal state

**Web Architectural Style Metamodel**

**“Rules of Engagement”**

Courtesy Terry Merriman, ©2002
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA and standards
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
Build the MDA Framework

Create the metamodels

Detailed metamodels describing the types and scope of models, rules for developing the models, and the mapping between models

Enterprise Metamodels
Start with a Business Model

A Detailed Business / Domain model describing the business requirements independent of computational concerns

Start with a Computational - Independent Business Model (CIM) representing business requirements and processes independent of if / how they are automated.
Create a PIM

Create a Platform-Independent Model (PIM) by mapping the business model conformant with the enterprise metamodel. The PIM represents business functionality and behavior in terms of computational concepts, but is undistorted by specific technology details.
Generate Platform-Specific Model

MDA tool applies a standard mapping to generate *Platform-Specific Model (PSM)* from the PIM. Code is partially automatic, partially hand-written.
Multiple Deployment Technologies

MDA tool applies an standard mapping to generate *Platform-Specific Model* (PSM) from the PIM. Code is partially automatic, partially hand-written.
MDA Tool generates all or most of the implementation code for deployment technology selected by the developer.
Integrating Legacy & COTS

MDA Tools for reverse engineering automate discovery of models for re-integration on new platforms.
Automating Bridges

Bridge generation is simplified by common application models, simplifying creation of integrated applications both within and across enterprises.
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA and standards
- Models and transformations
- MDA development process
- MDA tools
- Conclusion
MDA Process Review

- Business Model
- Platform Independent Model
- Platform Specific Model
- Code, Test, Doc Generation
- Meta & Modeling
- MDA Architect
- Model Transformation

- Business Analyst
- Application Architect
- Developer / Tester
- Code

Copyright © M²VP Inc, 2003, All rights reserved
Tools in the MDA Process

1. Architecture Modeling Tool
2. Business Model
3. Modeling, Validation and Simulation Tool
4. Transformation Tool
5. Code and Test Generation Tool
6. Code
7. Platform Specific Model
8. Integrated Development Env.
9. Asset Repository
10. Project Management

Business Analyst
Application Architect
Developer / Tester
MDA Architect
Questions for Your Enterprise

- What current tools and processes does MDA have to integrate with or support for development, test, reuse, documentation, etc.
- What will the MDA development lifecycle be in the organization?
- Who will perform the business modeling?
- What is the enterprise and application architecture?
- Can these be supported by standard profiles or will the organization be creating custom profiles and metamodels?
- What technology platforms will need to be supported?
Vendors

- www.omg.org/mda/committed-products
- Adaptive: Adaptive Framework
  www.adaptive.com
- Codagen Tech.: Gen-it Architect
  www.codagen.com
- Ebuilt/Codigo Solutions: CodigoXpress
  www.codigoXpress.com
- Headway Software: Headway ReView
  www.headwaysoftware.com
More Vendors

- Interactive Objects Software GmbH: ArcStyler (Being incorporated in Borland’s Enterprise Studio 2)
  [www.io-software.com](http://www.io-software.com)
- Kabira Tech: ObjectSwitch, Kabira Business Accelerator
  [www.kabira.com](http://www.kabira.com)
- Kennedy Carter Ltd: iUML, iCCG
  [www.kc.com](http://www.kc.com)
Even More Vendors

- Metamatrix: Modeler, MetaBase, Server
  www.metamatrix.com
- Metanololgy: Meta Development Environment
  www.metanology.com
- ONTOS: ObjectSpeak
  www.ontos.com
- Project Tech.: BridgePoint, DesignPoint
  www.projtech.com
- Secant Technologies: ModelMethods
  www.secant.com
Agenda

- Why MDA?
- A few words about architecture
- MDA concepts
- MDA and standards
- Models and transformations
- MDA development process
- MDA tools

Conclusion
Technical Summary

- Business semantics, technical infrastructure, etc. need to be modeled formally and rigorously.
- Technical infrastructure requirements need to be mapped to various middleware, databases, and legacies.
- Business semantics should be separated from technology.
- Models should be expressed in an industry standard, neutral format designed for models.
- Metamodels define the content and scope of models.
Benefits

- Even without automation, PIMs:
  - Improve tracing between business and technology models
  - Separates business and technical modeling roles
  - Promotes abstraction leading to better, more comprehensible designs
  - Improve portability to other platforms
  - Ease modifications to either the business or technology
OMG MDA Adoption Status

- Major direction agreed March ’01; overall architecture adopted September ’01.
- UML 1.4 complete; 2.0 in process.
- Mappings ("profiles") underway:
  - EDOC (adopted)
  - CORBA (adopted)
  - EAI (in FTF)
  - EJB (adopted by JCP)
  - SOAP/XML (in process)
  - .Net (RFP issued)
Standardization

- UML standardized modeling notation 5 years ago. Today we have:
  - Lot’s of tools
  - Supporting methodologies
  - Interoperability of models

- MDA will standardized architecture like UML standardized modeling:
  - Tools, services, methods.
The Bottom Line

- For MDA to deliver value
  - It must make models *first class development artifacts*
  - Tools must support all aspects
    - IDE for modeling
References and Bibliography

More References

Questions

To learn more: www.omg.org/fast-start