Model-Driven Architecture: Vision, Standards And Emerging Technologies

Position Paper Submitted to ECOOP 2001

Workshop on Metamodeling and Adaptive Object Models

John D. Poole
Hyperion Solutions Corporation

April 2001

john_poole@hyperion.com

1. Introduction

Recently, the Object Management Group introduced the *Model-Driven Architecture* (MDA) initiative as an approach to system-specification and interoperability based on the use of formal models [MDA, MDA2, DSouza]. In MDA, *platform-independent models* (PIMs) are initially expressed in a platform-independent modeling language, such as UML. The platform-independent model is subsequently translated to a platform-specific model (PSM) by mapping the PIM to some implementation language or platform (e.g., Java) using formal rules.

At the core of the MDA concept are a number of important OMG standards: The Unified Modeling Language (UML), Meta Object Facility (MOF), XML Metadata Interchange (XMI), and the Common Warehouse Metamodel (CWM). These standards define the core infrastructure of the MDA, and have greatly contributed to the current state-of-the-art of systems modeling [MDA2].

As an OMG process, the MDA represents a major evolutionary step in the way the OMG defines interoperability standards. For a very long time, interoperability had been based largely on CORBA standards and services. Heterogeneous software systems inter-operate at the level of standard component interfaces. The MDA process, on the other hand, places formal system models at the core of the interoperability problem. What is most significant about this approach is the independence of the system specification from the implementation technology or platform. The system definition exists independently of any implementation model and has formal mappings to many possible platform infrastructures (e.g., Java, XML, SOAP).

The MDA has significant implications for the disciplines of Metamodeling and Adaptive Object Models (AOMs). Metamodeling is the primary activity in the specification, or modeling, of metadata. Interoperability in heterogeneous environments is ultimately achieved via shared metadata and the overall strategy for sharing and understanding metadata consists of the automated development, publishing, management, and interpretation of *models*.[1] AOM technology provides dynamic system behavior based on run-time interpretation of such models. Architectures based on AOMs are highly interoperable, easily extended at run-time, and completely dynamic in terms of their overall behavioral specifications (i.e., their range of behavior is not bound by hard-coded logic).

The core standards of the MDA (UML, MOF, XMI, CWM) form the basis for building coherent schemes for authoring, publishing, and managing models within a model-driven architecture. There is also a highly complementary trend currently building within the industry toward the realization of these MDA standards in the Java platform (i.e., standard mappings of platform-independent models to platform-dependent models, where the platform-dependent model is the Java platform). This is a sensible implementation strategy, since development and integration is greatly facilitated through common

[1] In this context, the terms *model* and *metadata* can be used interchangeably, although *model* would seem to have a more general connotation.

platform services and programming models (interfaces or APIs), provided as part of the Java platform. Java 2 Platform, Enterprise Edition (J2EE), has become a leading industry standard for implementing and deploying component-based, distributed applications in multi-tier, Web-centric environments. Current efforts within the Java Community Process to develop pure Java programming models realizing OMG standards in the form of J2EE standard APIs (i.e., JMI, JOLAP and JDM) further enhance the metadata-based interoperability of distributed applications.

This paper surveys the core OMG MDA standards (i.e., UML, MOF, XMI and CWM) and discusses the current attempts at mapping these standards to J2EE, as examples of PIM-to-PSM translations that are currently under development. These forthcoming APIs will provide the initial building blocks for a new generation of systems based on the model-driven architecture concept. The progression of these initial MDA realizations to AOMs is the next logical step in this evolution.

2. The Vision

This paper's proposed vision for the evolution of the MDA is twofold, consisting of both a near term vision and a long term vision of the future. The near term vision (i.e., nearly seamless interoperability, based on formal PIM-PSM translations and shared metadata) is achievable right now. The supporting technologies are largely specified and implementations are currently being built by a number of organizations. The long-term vision (i.e., the wide spread deployment of AOMs, as an evolution of the MDA), on the other hand, is still being conceptualized.

Overview of the Near Term Vision

The proposed near term vision is that of an environment in which efficient and nearly seamless interoperability between diverse applications, tools and databases is achieved through the interchange of shared models. Components participating in this environment leverage standard services provided by implementations of MDA standards that enable them to expose and interchange their metadata as instances of well-defined models. These platform services have standard definitions that are expressed via standard programming models (APIs), which are automatically generated from platform-independent models.

The Importance of Shared Metadata

Metadata is critical to all aspects of interoperability within any heterogeneous environment. In fact, metadata *is* the primary means by which interoperability is achieved (interoperability is largely facilitated by standard APIs, but ultimately requires shared metadata as the definitions of system semantics and capabilities). Any MDA-based system must have the ability to store, manage and publish both application- and system-level metadata (including descriptions of the environment itself). Applications, tools, databases, and other components plug into the environment and discover metadata descriptions pertaining to the environment. Similarly, a component or product introduced into the environment can also publish its own metadata to the rest of the environment. This scenario is illustrated in Fig. 1.

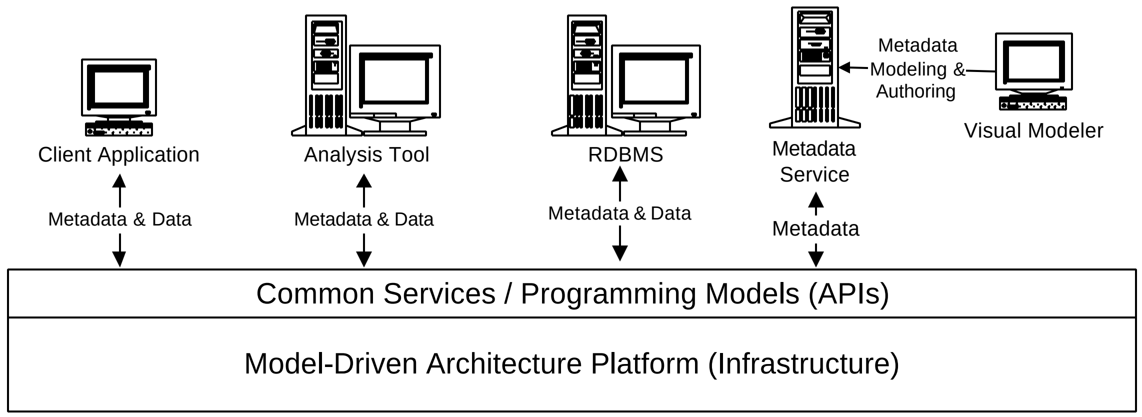
Client Application
Metadata & Data
Analysis Tool
Metadata & Data
RDBMS
Metadata & Data
Metadata Service
Metadata
Metadata Modeling & Authoring
Visual Modeler
Common Services / Programming Models (APIs)
Model-Driven Architecture Platform (Infrastructure)

Figure 1: An Example of a Realization of Model-Driven Architecture

Having an abundance of shared, descriptive metadata (*ubiquitous metadata*) facilitates software interoperability between platform components in very specific ways, including:

- Data interchange, transformation, and type mapping between dissimilar data resources can be driven by formal, product-independent metadata descriptions of the transformations, data types, and type-system mappings.

- Schema generation can be based on shared metadata descriptions of common schema elements. For example, both a relational database and an OLAP server can build their own internal representations of a dimensional model, according to a standard, metadata-based definition of "dimension" published to the environment. Having common metadata definitions facilitates data interchange between subsystems, because there is a common understanding of what the data means.

- Business intelligence and visualization functions can utilize metadata in the processing and formatting of data for analysis and display. Metadata descriptions confer the "higher level of meaning" on data items that analysts and reporting users need in order to make sense of data points and results (e.g., definitions of business terms, glossaries, taxonomies, nomenclatures, etc., are a part of the shared metadata).

- Software components with no prior knowledge of each other's capabilities, interfaces, and data representations can interoperate once they've established a metadata "handshake", in which each exposes its features and assesses those of the other. Note that this exchange of knowledge does not always need to be complete, but to the extent that components can make sense of each other's capabilities, they can interact with one another. In the absence of specific knowledge, components might rely on standard defaults, or may be able to refer to some other source of information to fill the knowledge gaps.

An MDA-based system does not require that *internal representations* of metadata within applications, tools, and databases be modified to correspond to the shared definitions. Product-specific internals and programming models remain as they are. Shared metadata consists of *externalized* definitions that are interchanged between participating components. These external definitions are readily understood by components that agree on the metamodel describing the metadata (e.g., CWM). External definitions are highly generic, but also possess sufficient *semantic completeness* (with respect to the problem domains that components need to address), and are, therefore, understood by a wide range of participants. Highly product-specific metadata that does not fit the generic model is handled through the use of extension mechanisms that are pre-defined as part of the generic models (e.g., the use of UML extension mechanisms, such as tagged values, stereotypes, and constraints).

To ensure that shared metadata is readily understood by all participating components, an MDA-based system requires its components to standardize on each of the following:

- A formal language (syntax and semantics) for representing metadata.
- An interchange format for exchanging and publishing metadata.
- A programming model for metadata access and discovery. This must include generic programming capabilities for dealing with metadata of an unknown nature.
- Mechanisms for extending each of the above.
- An optional metadata *service* of some form, where published metadata resides. Note that there is no strict requirement that such a service be provided; i.e., components are always free to interchange metadata in a point-to-point fashion, without the use of an intervening service).

Common Services and Programming Models

In addition to shared metadata, another key building block of interoperable systems is the standardization of common system- and application-level services, along with the application programming interfaces (APIs) used to access these services. A standard API defines a *standard programming model* of the services it represents. This form of standardization simplifies clients and facilitates the integration of new components into the MDA-based environment. Clients using common services have a smaller footprint and are less complex, because they only need to be coded to one interface, regardless of how services are actually implemented in a particular deployment. Conversely, service providers implementing standard APIs are readily available for use by a large number of clients.

Platform Specification

The final building block of the near term system vision is the *platform specification*. This is basically the complete definition of the metadata interoperability and interchange strategies, common services, and standard APIs that any instance of an MDA-based system is capable of supporting. Each MDA-based system instance includes a *descriptor* that specifies those features actually supported by that particular deployment. Software tools for specifying and integrating system components generate the descriptor, and tools for configuring, installing, and bootstrapping an instance of the system are driven by the descriptor.

Overview of the Long Term Vision

The long term vision for MDA-oriented system architectures includes software capable of automatic discovery of properties of its environment and adaptation to that environment by various means, including dynamic modification of its own behavior. This is an ambitious vision that builds significantly on experiences and insights gained from implementing the near term vision. It represents a migration of the near term vision (i.e., metadata-based interoperability) to that of systems whose behavior is largely determined at run-time by AOMs. The following points summarize the main characteristics of the long term vision:

Knowledge-Based Orientation

System functionality will gradually become more knowledge-based and capable of automatically discovering common properties of dissimilar domains, making intelligent decisions based on those discoveries, and drawing and storing resulting inferences. In general, "knowledge" is supported by an advanced and highly evolved concept of ubiquitous metadata, in which the ability to act upon, as well as revise, knowledge at run time is provided through AOMs.[2]

Our ability to engineer such systems will come about largely as the result of our extensive experiences with the use of metamodels and ontologies in influencing system behavior and decision making. We will eventually learn how to build systems in which a considerable amount of domain knowledge is pushed up into higher abstraction levels. Systems will understand how to efficiently extract and act on that information.

Another factor contributing to the development of knowledge-based systems will be the future availability of far more effective reflective capabilities, as provided by programming language implementations, as well as repository and middleware services, and most importantly, generalized metadata management, authoring and publishing

[2] It is difficult to say how crisp a distinction will remain between shared metadata and AOMs in these future systems. It is quite possible that the concept of metadata as a relatively static specification of a system may be rendered completely obsolete in the future.

services. The current state-of-the-art of reflection generally allows for static program introspection. Future reflective capabilities will efficiently support not only introspection, but also dynamic modification of both structure and behavior [Chiba, Franz, MOF].

Dynamic Architecture

Experiences gained in the development and deployment of metadata-driven systems based on extensible object models will ultimately result in the development of systems that can directly interpret models and modify their own behavior accordingly, rather than explicitly mapping external views of shared metadata to implementation models.

In the future, this mapping process will be discarded and models will be interpreted directly by software. This is the promise of the areas of dynamic objects and AOMs [Franz, Kiczales, AOM, Poole]. In this paradigm, changing the model directly changes software behavior, resulting in complete run-time extensibility. For example, publishing a new version of a model in a repository causes all software systems in the environment to automatically adjust their own behaviors to reflect the changes. Note that a highly evolved concept of metadata is critical to this sort of architecture. Metadata (i.e., the Adaptive Object Model) is updated while the system is executing, and the resulting changes to system behavior and structure takes effect as soon as the running system is ready to expose those changes.

Adaptive Systems

The architectures and capabilities described above will produce a general class of highly dynamic and self-organizing systems that can act directly on domain knowledge and behave intelligently without having to be told how. Such systems can readily accommodate unforeseen changes in the environment and react appropriately without the need for programmer intervention (e.g., when dynamic and largely unstructured data resources are brought into the environment). When systems do need to be modified, this is accomplished by altering the system model. This may be performed by domain experts who are not necessarily software specialists, or perhaps by the system itself, in many cases.

3. Realizing the Near Term Vision: A Survey of the Standards

Metadata Integration: CWM, UML, MOF and XMI

The key to successful integration and interoperability lies in the intelligent use and management of metadata across all applications, platforms, tools, and databases. Metadata management and integration can be accomplished through the use of the OMG's core MDA standards: CWM, MOF, UML and XMI.

CWM

The *Common Warehouse Metamodel (CWM)* defines a metamodel (a *model of the data model*) representing both the business and technical metadata that's most often found in the data warehousing and business analysis domains [CWM]. It is used as the basis for interchanging instances of metadata between heterogeneous, multi-vendor software systems (i.e., for integrating the data warehousing and business analysis information "supply chain"). Systems that understand the CWM metamodel exchange metadata in formats that are consistent with the metamodel.

CWM is actually comprised of a number of constituent metamodels representing data resources, analysis, warehouse management, and foundational components of a typical data warehousing/business intelligence environment. Data resource metamodels support the ability to model legacy and non-legacy data resources, including relational databases, record-oriented databases, and XML- and object-based data resources. An analysis layer of CWM defines metamodels for data transformations, OLAP, information visualization/reporting, business nomenclature, and data mining. A warehouse management layer consists of metamodels representing standard warehouse processes, activity tracking and scheduling (e.g., daily extracts and loads). Finally, the foundation metamodel supports the specification of various common elements and services, such as data types, type system mappings, abstract keys and indexes, expressions, business information, and component-based software deployment.

CWM represents a *model-based* approach to interchanging metadata between software systems [Tolbert]. Metadata shared between products is formulated in terms of data models that are consistent with one or more CWM metamodels. A product *exports* metadata by formulating a model of its internal metadata structures in a format prescribed by CWM. Similarly, a product *imports* metadata by consuming a CWM-compliant model and mapping it to its internal metadata.

The collection of metamodels provided by CWM is comprehensive enough to model an entire data warehouse. Using CWM-aware tools, a data warehouse instance could be generated directly from a warehouse model. Each of the various tools consume those portions of the model that they can make use of. For example, a relational database server will consume the relational portion of the model and use it to build its catalog. Similarly, an OLAP server will search the model for OLAP metadata and use it to define its multidimensional schema. An extract-transform-load (ETL) tool would most likely process a slice of the warehouse model spanning several CWM metamodels, including the relational, OLAP, transformation, data type, type mapping and expression metamodels.

CWM models are intended to be highly generic, external representations of shared metadata. Metadata that does not readily fit the CWM format (e.g., highly tool-specific metadata that must be interchanged) is handled either through standard extension mechanisms provided by CWM, through extensions to the core CWM metamodel, or through the use of product-specific defaults, user input, or some other deployment-defined logic.

UML

CWM is expressed in the *Unified Modeling Language (UML)*, an OMG standard language for modeling discrete systems [Rumbaugh]. UML is the notational basis for the definition of CWM, but CWM also extends a subset of the core UML metamodel with data warehousing and business analysis domain concepts.

When constructing data warehouse models based on CWM, the use of visual modeling tools (supporting UML or some equivalent, formal notation) is the preferred method, since visual models of complex metadata structures are more easily managed and comprehended by human beings than when represented in other formats (e.g., textual representations). On the other hand, since the UML language has a precise definition (i.e., via the UML metamodel), visual UML models are capable of automatic translation to other formal languages (visual as well as non-visual). This facilitates the interchange of CWM models in various platform- and tool-independent formats (e.g., XML), as well as the construction of tool-specific metadata from CWM models (e.g., translation of a CWM relational model into SQL DDL statements that actually build the schema).

MOF

The *Meta Object Facility (MOF)* is an OMG standard defining a common, abstract language for the specification of metamodels [MOF]. MOF is an example of a *meta-metamodel,* or *model of the metamodel* (sometimes called an *ontology*).

MOF is distinctly object-oriented in nature. It defines the essential elements, syntax, and structure of metamodels that are used to construct object-oriented models of discrete systems. MOF serves as the common model of both the CWM and UML metamodels. Specifically, the MOF specification provides:

- An abstract model of the generic MOF objects and their associations.
- A set of rules for mapping any MOF-based metamodel to language-independent interfaces (defined in CORBA IDL). An implementation of these interfaces for a given metamodel would be used to access and modify any model based on that metamodel.
- Rules defining the life cycle, composition, and closure semantics of elements of MOF-based metamodels.
- A hierarchy of reflective interfaces. These define generic operations for discovering and manipulating models based on MOF-compliant metamodels, but whose mapped interfaces are unknown.

The power of MOF is that it enables otherwise dissimilar metamodels (representing different domains) to be used in an interoperable manner. MOF-aware applications may not have any knowledge of the domain-specific interfaces of some model instance, but can still read and update that model using the generic operations of the reflective interfaces.

MOF semantics generally define metadata repository services that support model construction, discovery, traversal, and update, where models are understood to be instances of some particular metamodel. In particular, the MOF's support for model life cycle semantics means that a MOF implementation provides an effective *metadata authoring and publishing tool*, when combined with support for visual modeling. For example, newly developed metamodels can be persisted in the MOF repository and combined with existing metamodels according to MOF life cycle and composition semantics (inheritance, clustering, nesting, etc.). Model interfaces and default implementations can then be generated and made available to the environment. Default implementations are further enhanced with the inclusion of additional programmed logic, either written by hand or generated from tools (e.g., implementation of OCL constraints). A fully MOF-compliant repository provides a significant number of metadata services that go well beyond the construction and serving of metadata (e.g., persistence, versioning, directory services).

XMI

XML Metadata Interchange (XMI) is an OMG standard that maps the MOF to the W3C's eXtensible Markup Language (XML) [XMI]. XMI defines how XML tags are used to represent serialized MOF-compliant models in XML. MOF-based metamodels are translated to XML Document Type Definitions (DTDs) and models are translated into XML Documents that are consistent with their corresponding DTDs.

XMI solves many of the difficult problems encountered when trying to use a tag-based language to represent objects and their associations. Furthermore, the fact that XMI is based on XML means that both metadata (tags) and the instances they describe (element content) can be packaged together in the same document, enabling applications to readily understand instances via their metadata. Communication of content is both self-describing and inherently asynchronous. This is why XMI-based interchange is so important in distributed, heterogeneous environments.

Common Services and Programming Models: Java and J2EE, JMI, JOLAP, JDM

Java and Java 2 Platform, Enterprise Edition

The Java programming language [Java] has provided an infrastructure for the wide spread deployment of heterogeneous, component- and Web-based, distributed applications. Java programs are highly transportable because the Java language is interpreted. A Java program is compiled into a byte stream that is processed by a Java Virtual Machine

(JVM). If a JVM is available for a particular operating system, or is embedded in a browser, then Java programs can run in either environment.

Portability is also facilitated through the availability of a large collection of standard and optional services and APIs. Java services and APIs are developed within the Java Community Process, an open process in which participants contribute to the development of Java specifications [JCP]. Any system developer may provide an implementation of some particular Java language library. As long as the implementation conforms to the interfaces and semantics of the Java specification, portability is guaranteed.

Java 2 Platform, Enterprise Edition (J2EE), is set of Java specifications that collectively define a complete multi-tier, component-based architecture for deploying distributed applications [J2EE]. J2EE takes a container-based approach to deploying platform services, where a container is a run-time environment for a collection of related components. Web-oriented middle-tier services, for example, are organized in Web containers, and application/business logic components are generally organized in Enterprise Java Bean (EJB) containers. Access to data resources residing on the third tier is also addressed by a number of J2EE specifications, including JDBC [JDBC] and the forthcoming Java 2 Connector Architecture [J2C].

J2EE provides a number of significant advantages to application developers, including the availability of a consistent programming model across multiple tiers. J2EE allows for the rapid development and deployment of distributed applications across heterogeneous, multi-tier platforms.

Within the JCP, a number of Java specifications are currently under development that represent formal mapping of OMG MDA standards to Java technology models. These are the Java Metadata Interface (JMI), Java OLAP Interface (JOLAP), and Java Data Mining API (JDM). Each of these is described in subsequent subsections.

These efforts represent a natural extension and continuation of the OMG's MDA standards. *In fact, an obvious trend developing within our industry is the increasing "realization" of the generic MOF and CWM metamodels in the form of J2EE specifications*. This trend is highly significant, because it means that there is an acknowledged need to provide consistent programming models and services for metadata and platform interoperability within the open platform environment.[3]

JMI

Java Metadata Interface (JMI) provides a formal mapping of the OMG's MOF to the Java language. A JMI implementation allows for the generation of pure Java interfaces for

[3] Two other highly significant, related efforts are UML Profile for EDOC [EDOC] and UML/EJB Mapping Specification [UMLEJB]. These efforts will ultimately form the basis for integrating UML-based modeling with Java and component-based development environments. These efforts are not elaborated here, however, as this paper has focused primarily on metadata, metamodeling and AOMs.

programmatic and XMI-based access to repository-based MOF metamodels and their instances. This means that a Java implementation of any MOF-based metadata service can expose both the generic and metamodel-specific interfaces derived from the MOF's interface mapping rules. Java clients have completely portable access to metadata services via JMI.

The development of JMI is being led by Unisys and includes the participation of Sun Microsystems, Hyperion, IBM, Oracle, and a number of other industry leaders [JMI].

JOLAP

Java OLAP Interface (JOLAP) is an effort to develop a pure Java API for OLAP servers and applications deployed in the J2EE environment. The development of JOLAP is being led by Hyperion Solutions Corporation, and includes the participation of IBM, Oracle, Unisys, Sun Microsystems, and other industry leaders [JOLAP].

In the J2EE environment, JOLAP generally serves as the client API of an OLAP server or other multidimensional database system residing on the data services tier of the J2EE environment.

JOLAP uses the CWM OLAP metamodel to describe OLAP metadata, thus ensuring that JOLAP-compliant resources are capable of complete metadata interoperability and interchange via the CWM standard. JOLAP also defines query interfaces that support the formation and execution of OLAP queries, along with the management and manipulation of multidimensional result sets.

JOLAP will leverage a number of existing and forthcoming J2EE APIs. This includes: Java 2 Connector Architecture (J2C) for connection management and data-tier resource and transaction management, Java Metadata Interface (JMI) for advanced metadata functionality (including reflective capabilities), Java Naming and Directory Interface (JNDI) for directory services, and the Java Security Model to provide single sign-on, authentication and authorization.

JDM

Java Data Mining API (JDM) provides a pure Java API for business intelligence applications employing data mining techniques for knowledge discovery and analysis. JDM is similar to JOLAP in the sense that it represents the Java reification of a CWM metamodel (i.e, CWM Data Mining package) [JDM].

The development of JDM is being led by Oracle and includes the participation of Hyperion, IBM, Sun Microsystems, and others.

Relating the Various Standards

The diagram below illustrates the general relationships between the OMG MDA standards and corresponding J2EE Platform standards. Note that in the case of JOLAP and JDM, alignment between OMG and Java occurs specifically in the metadata management interfaces of the Java APIs. Both JOLAP and JDM define additional interfaces for data and query management which, of course, are not within the scope of CWM or MOF.

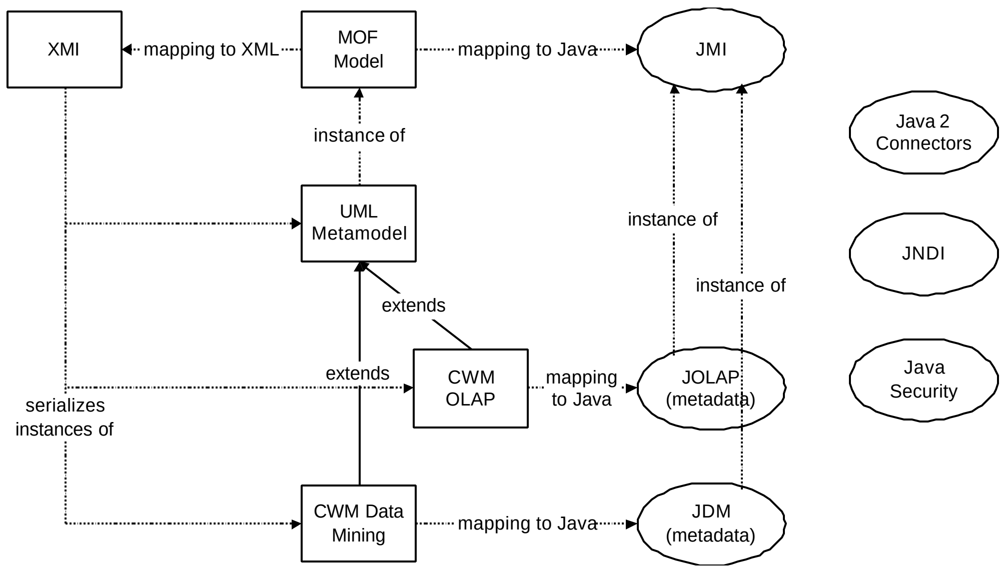

Figure 2: Relationships Between Standards

References

[AOM] ECOOP '2000 Workshop on Metadata and Active Object-Models, June, 2000, Cannes, France. *Lecture Notes in Computer Science* 1852, Springer-Verlag, Heidelberg, http://www.adaptiveobjectmodel.com/ECOOP2000/.

[Chiba] Chiba, S., "Load-Time Structural Reflection in Java", *Proceedings of ECOOP 2000*, pp.311-336, Lecture Notes in Computer Science 1850, Springer-Verlag, 2000.

[CWM] Object Management Group, The Common Warehouse Metamodel (specifications, papers, presentations, OMG press kit, etc.). http://www.cwmforum.org/, http://www.omg.org/.

[DSouza] D'Souza, D., "Model-Driven Architecture and Integration: Opportunities and Challenges", Version 1.1. http://www.catalysis.org/publications/papers/2001-mda-reqs-desmond-6.pdf

[EDOC] OMG, UML Profile for EDOC RFP Home Page: http://cgi.omg.org/techprocess/meetings/schedule/UML_Profile_for_EDOC_RFP.html

[Franz] Franz, Inc., "The Meta-Object Protocol and Knowledge-Based Systems", December, 1997. http://www.franz.com/support/tutorials/mopnkbs.php3

[J2C] J2EE Connector Architecture, JSR-16 Home Page http://java.sun.com/aboutJava/communityprocess/jsr/jsr_016_connect.html.

[J2EE] Java 2 Platform, Enterprise Edition Home Page: http://java.sun.com/j2ee/

[Java] Java Technology Home Page: http://java.sun.com/

[JCP] Java Community Process Home Page: http://java.sun.com/jcp

[JDBC] JDBC Data Access API Home Page: http://java.sun.com/products/jdbc/index.html

[JDM] Java Data Mining API, JSR-73 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_073_dmapi.html

[JMI] Java Metadata Interface, JSR-40 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_040_mof.html

[JOLAP] Java OLAP Interface, JSR-69 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_069_jolap.html

[Kiczales] Kiczales, G., des Rivieres, J., and Bobrow, D.G., *The Art of the Meta-object Protocol*, MIT Press, 1991.

[MDA] OMG Model-Driven Architecture Home Page: http://www.omg.org/mda/index.htm

[MDA2] OMG Architecture Board MDA Drafting Team, "Model-Driven Architecture: A Technical Perspective", ftp://ftp.omg.org/pub/docs/ab/01-02-01.pdf

[MOF] OMG Meta Object Facility Specification, Version 1.3, September, 1999. http://www.dstc.edu.au/Research/Projects/MOF/rtf/. http://www.omg.org/.

[Poole] Poole, J., "The Common Warehouse Metamodel as a Foundation for Active Object Models in the Data Warehousing Environment", ECOOP '2000 Workshop on Metadata and Active Object-Models, June, 2000, Cannes, France. *Lecture Notes in Computer Science* 1852, Springer-Verlag, Heidelberg. http://www.adaptiveobjectmodel.com/ECOOP2000/. http://www.cwmforum.org/CwmAOM.pdf.

[Rumbaugh] Rumbaugh, J., Jacobson, I., Booch, G., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.

[Tolbert] Tolbert, D., "CWM: A Model-Based Architecture for Data Warehouse Interchange", Workshop on Evaluating Software Architectural Solutions 2000, University of California at Irvine, May, 2000. http://www.cwmforum.org/uciwesas2000.htm

[UMLEJB] Java UML/EJB Mapping Specification, JSR-26 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_026_uml.html.

[XMI] Object Management Group, XML Metadata Interchange Specification, Version 1.1, http://www.omg.org/.