Modelware for Middleware

Keith Duddy, Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel

{dud,agerber,lawley,kerry,steel}@dstc.edu.au

CRC for Enterprise Distributed Systems (DSTC)[*]

April 16, 2003

Abstract

The OMG's Model Driven Architecture is a reference architecture for model driven development of computing systems. Implementations of primitive forms of this style of development are everywhere, but a reference implementation architecture is still some way off. This paper explains our set of requirements for the main components required for MDA: A single Meta-Modeling Language, Platform Independent Modelling Languages, Platform Specific Modelling Languages, and a Transformation Language. It also sketches the solutions that are emerging from various standardisation efforts in which the authors are participating in particular the EDOC ECA language and the MOF 2.0 Query, View and Transformation language.

1 Introduction

In this position paper we will demonstrate the benefits of the MDA™ approach, as it is becoming defined at OMG [4] and elsewhere.

The MDA approach can be exemplified by considering the design of an enterprise application creating a Platform Independent Model (PIM) of an application using a middleware-inspired architectural modelling language such as EDOC ECA [6, 1]. A platform for implementation and deployment of this application is then selected, such as Sun's J2EE. The transformation of the PIM into a Platform Specific Model (PSM) is achieved via a transformation description (or mapping) from the PIM modelling language to the PSM modelling language.

The benefits of this approach are manifold. Firstly there are benefits to using a high-level architectural modeling language for designing the application PIM. These have to do with the abstraction of irrelevant detail, and the constraints of such a langauge which capture best practice for design.

Secondly the transformation of the PIM to a PSM for a particular platform allows the encapsulation of further design best practice, using a paradigm of pattern matching in the PIM to fill in pattern templates in the PSM. As the modelling languages we use are MOF-based, and therefore object-oriented in nature, certain transformation rules can be applied to very general concepts, and then extended or superseded based on sub-types of these concepts, or on predicates which identify subsets of object instances.

Thirdly the transformation framework allows additional information (captured as other models) that reflects the capabilities of the platform to be used. This information relates the objects in the PIM to platform concepts, allowing the annotation of the PIM to indicate which application artifacts should be implemented in what manner. Often this will address considerations including non-functional issues (the "-ilities" such as scalability, reliability, etc) and deployment and integration issues.

[*]The work reported in this paper has been funded in part by the Co-operative Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science and Training).

2 Technologies and Terminologies

2.1 MDA

The OMG's Model Driven Architecture™ (MDA) [4] defines an approach to enterprise distributed system development that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. The MDA approach envisions transformations (or mappings) from Platform Independent Models (PIMs) to one or more Platform Specific Models (PSMs).

The main requirements we have of an MDA environment are:

- High quality (meta)models of architectural approaches are available
- High quality (meta)models of platforms are available
- Multiple levels of mappings between models of different degrees of abstraction are possible
- Mappings can expressed declaratively
- Executions of mappings are traceable and repeatable
- The results of mappings can be augmented without loss when altering the models and regenerating the results
- Different aspects or viewpoints of a system can be captured by separate but related models
- Mappings can use several models as input, and produce several models as output (preserving viewpoint separation or aspects at different levels of abstraction)

2.2 Platform Independent Models

A Platform Independent Model is an expression of an abstract system design which can be implemented for, or executed on a range of similar platforms. Platforms can be any set of services provided by a computing and/or network environment on which some kind of higher-level application can execute.

An example where the platform is any one of several application servers is the OMG's EDOC Enterprise Collaboration Architecture (ECA) [6]. It is a structured framework for recursive definitions of computational objects and their interactions. It can represent designs for B2B interactions, container managed entities and components, synchronous and asynchronous messaging, and workflow-style processes. A language like ECA has many benefits:

- Such langauges may be constrained to use best-practice patterns and structures distilled from the experience of architects.
- The model of the application can be expressed in terms closer to the business domain of the application, without mixing in concerns for optimisation of the implementation. This can be considered an aspect-oriented approach [3].
- Being based on MOF it may have several representations for different purposes:
  - *graphical* – for ease of comprehension and communication, especially to non-technical domain experts and clients
  - *programmatic* – APIs may be generated for access to the model by browsers, graphical tools, and transformation engines
  - *textual* – XML forms can be generated for tool interchange. More human usable forms such as the recently adopted OMG Human Usable Textual Notation (HUTN) standard [5] can be used for display to, or input by, programmers.

2.3 Platform Specific Models

A Platform Specific Model (PSM) is an expression in some language that represents some aspect of a system targeted directly to a platform. Following the PIM example, an example Application Server platform is Enterprise Java Beans (EJB).

Designs for the EJB platform can also be expressed as MOF models using the OMG's EDOC specification [6]. This avoids the difficulties of manipulating text files and other uniquely formatted artifacts, often with an implicit abstract syntax.

At some point in a process of mapping models to other models, a native representation (usually in the form of text files) is required, as the PSM produced will be

directly executable (perhaps with some additional programming effort added) on the platform. DSTC has produced technology known as Anti-Yacc [2] which describes what is essentially a model-to-text pretty print.

2.4 Model Transformations

At the time of writing the OMG has issued a Request for Proposals (RFP) for MOF 2.0 Query, View, and Transformation [7] technology. The authors are currently preparing such a proposal. The main concepts it introduces are:

- Transformation rules consist of source model and target model patterns and specifications of equivalences between the two.
- Source model patterns are used to find matching instances in a (set of) source model(s).
- Target model patterns are used to construct new instances in the target model(s) with attribute values and relationships (associations) satisfying the equivalence expressions.
- The relationships *extends* and/or *supersedes* can be specified between rules, and are similar to the object-oriented concepts of inheritance and overriding.
- An extra *configuration model* can be used to "mark" model elements in the source PIM with information that is not directly expressible in the PIM language.

3 Capturing Best Practice in the Mapping

In this section we discuss the requirements for a mapping language that allows the most flexible and expressive definitions of the mappings between models. We also position our MOF Query/View/Transformation proposal in terms of the capabilities it offers as solutions to these requirements.

3.1 Requirements for a Mapping Language

It is widely accepted that a transformation language must be able to:

- Match elements, and ad-hoc tuples of elements, by type (include instances of sub-types) and precise-type (exclude instances of sub-types).
- Filter the set of matched elements or tuples based on associations, attribute values, and other context.
- Handle recursive structure with arbitrary levels of nesting.
- Support both multiple source extents and multiple target extents.

The requirement that a transformation language must be declarative is stated in the MOF Query, View and Transformation RFP. We take this to mean that it must not describe a procedural mechanism for how a target model should be constructed from a source model, but simply assert the relationships between them.

In addition we believe that the following requirements for are important for application to a broad range of transformations and for clarity of expression.

- There should be no dependency on the application order of the rules, and all rules are applied to all source elements that match.
- Creation of target objects is implicit rather than explicit. This follows from the previous requirement; if there is no explicit rule application order, then we cannot know which rule creates an object and are relieved of the burden of having to know. Objects are simply created on demand during the execution of a transformation.
- A single target element may be defined by multiple rules. That is, different rules can provide property values for the same object.
- Named associations are established between source and target model elements. These associations can then be used for maintaining traceability information.

3.2 Aspect-driven Transformation

The last set of requirements are important so that readable transformation descriptions can be written to map many source model elements into a single target model element, or vice-versa. That is, a rule may be written to deal with each concept of interest, no matter what the granularity on either source or target side of the rule.

However, the general case of this approach, which we call aspect-driven transformations, allow rules to be structured around semantic concepts rather than objects. E.g. transforming all imperial measurements to metric ones, replacing one naming system with another, changing a security policy for a certain type of attribute.

Aspect-driven transformations are a major reason why we favour implicit (rather than explicit) creation of target objects, as aspect-driven transformation rules rarely address entire objects, and thus it is extremely difficult to determine which of several transformation rules (which may or may not apply to any given object) should then have responsibility for creating the target object. Typically the target object is only required if any one of the transformation rules can be applied, but no target object should be created if none of the rules can be applied. This is extremely difficult to express if explicit creation is used.

3.3 Transformation Re-use

The final consideration for writing readable and re-usable transformation definitions is the ability to re-use both patterns for matching or creating model elements, and whole rules.

The former requires a named pattern construct with parameters to allow a common structure to be matched or created in multiple rules by using an expressive name. The latter is supported by two kinds of rule re-use:

- *extension*, in which additional constraints can be added to the matching of a source model element, and additional target elements for creation can be defined.
- *supersession*, in which a rule can be overridden in certain circumstances. Superseding is not only ideal for rule optimisation and rule parameterisation, but also enhances reusability since general purpose rules can be tailored after-the-fact without having to modify them directly.

3.4 Parameterising the Mapping

One of the major stumbling blocks in the acceptance of MDA-like technologies in the past has been the one-size-fits-all nature of the mappings.

Our approach to transformation specification avoids this trap via two complementary mechanisms.

The first mechanism has been discussed in the previous sections. It is the expressive power of the extension mechanisms in the proposed MOF transformation language.

The second is the use of mapping configuration models and parameterised mappings. With this approach, a source model is effectively annotated, via a separate configuration model, with information that can be used by the mapping rules to select between a variety of possible transformations.

For example, if an operation is known to be read-only but the PIM has no place to indicate that an operation is read-only, then we can add this information to a configuration model and write the transformation rules such that they use this information if it is available.

So, while the second approach works well when the parameterisation possibilities are known in advance by the rule writer, the first approach allows new rules to be written as new knowledge comes to light, or to compensate for rules that were, perhaps, written for an older version of the PSM language. Another possible use is that transformation to a generic platform language, for example EJB, can be superseded for a language that extends this with product-specific details and features.

4 Conclusion

The benefits of an MDA approach to middleware application development are as follows.

Firstly, the application architecture can be expressed in a “pure” form, without regard for the optimisation concerns of a particular middleware platform. This results in a consistent, easy to understand, domain-focused model which can be easily explained to domain experts in diagrams.

Although platforms such as EJB attempt to offer applications a number of simplifying services to aid implementation, performance considerations often dictate that these services are worked around, or used only in certain constrained ways [8].

Thanks to the power of the proposed DSTC MOF Transformation specification the experience of multiple architects over multiple projects can be expressed as a

concrete, declarative mapping from the concepts available in the PIM language to the concepts of the implementation platform. This mapping specification can capture the best practice for implementation of domain concepts on a particular platform.

In addition to mappings based only on the PIM concepts, configuration models can be exploited to add back the details of a platform that are abstracted out in languages like ECA. This effectively allows tools to query architects about which of the available implementation styles to choose for each PIM artifact based on, among other things, knowledge of the domain, and knowledge of probable usage patterns, performance, and robustness requirements.

References

[1] A. P. Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, and A. Wood. Processes, roles, and events: Uml concepts for enterprise architecture. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*, volume 1939 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2000.

[2] D. Hearnden, K. Raymond, and J. Steel. Anti-Yacc: MOF-to-text. In *EDOC 2002, Proceedings of the Sixth IEEE International Enterprise Distributed Object Computing Conference*, Lausanne, Switzerland, Sept. 2002. IEEE, IEEE.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *11th Europeen Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.

[4] OMG. Model Driven Architecture – A Technical Perspective. OMG Document: ormsc/01-07-01, July 2001.

[5] OMG. Human-Usable Textual Notation. OMG Document: ptc/02-12-01, Dec. 2002.

[6] OMG. UML Profile for Enterprise Distributed Object Computing (EDOC). OMG Document: ptc/02-02-05, Feb. 2002.

[7] Request for Proposal: MOF 2.0 Query / Views / Transformations RFP. OMG Document: ad/02-04-10, Apr. 2002.

[8] J. Shirazi. EJB Performance Tips. `http://www.javaperformancetuning.com/tips/j2ee_ejb.shtml`, Nov. 2002.