Elemental and Pegamento: The Final Cut
Applying the MDA Pattern

K. Duddy, M. Lawley and Z. Milosevic.

Distributed Systems Technology Centre,
University of Queensland,
Brisbane, QLD 4072, Australia.
{dud, lawley, zoran}@dstc.edu.au

Abstract

We provide an investigation of the applicability of the Model Driven Architecture ^{TM} *to the development of a technical architecture in a specific domain, viz: contract monitoring. We define MDA in terms of a single basic pattern, which is then composed in several ways to represent the behaviour of a large range of MDA tools.*

The paper introduces both the work of the Elemental project to provide the domain example, and the Pegamento project to explain the metamodels and tools in the MDA framework which might be applied to the domain.

Elemental has developed an architecture and a language for supporting Enterprise Contract Management (ECM) as part of an extended enterprise model. Pegamento has specified and built MDA prototype tools using several modelling languages, meta-languages and platforms. As the work of both these projects reaches maturity, "the Final Cut"[1] *is our proposed application of this toolset to facilitate building a model-based implementation of the relevant parts of Elemental's ECM system.*

We discover several generic MDA pattern matches in this case study, and several specific to the ECM. We also identify and discuss pattern mismatches.

Keywords: *Model Driven Architecture, Contract Architecture, Patterns*

[1] This paper is dedicated to several young researchers involved in these two projects since 1999, in particular James Cole, Anna Gerber, Simon Gibson, Sachin Kulkarni and Jim Steel.

1. Introduction

This paper is an attempt to investigate the extent to which the OMG's Model Driven Architecture^{TM} [1][2][3] can be applied to the development of a technical architecture in a specific domain, viz: automated contract monitoring. This technical architecture belongs to a class of architectural designs that could be characterised by the existence of a generic enterprise architecture pattern that can be tailored to a specific deployment environment through model-based configuration of the enterprise pattern.

This paper has been in the making over the last year or so, as the culmination of many years of collaborative efforts between two DSTC projects, namely Elemental and Pegamento. While both projects were established in 1999 and worked jointly until 2001 on a response to the OMG's "UML profile for EDOC" RFP [10], our subsequent work has diverged. Pegamento has continued its focus on establishing modelling and model transformation as the driver for middleware application development in the OMG – now branded as MDA^{TM} . Elemental has focused on inter-organisational issues such as the specification of enterprise policies [8], communities and business contracts [18][25][28], and have contributed to the OASIS legalXML e-contracts TC [33]. The key outcomes of the Elemental ECM are the Business Contract Architecture (BCA) (primarily based on [21], and the Business Contract Language (BCL) [29].

In developing the BCA and the language for expressing its main concepts: the Business Contract Language (BCL), the Elemental project initially adopted an XP-based approach to develop a proof-of-concept demonstrator and to experiment with emerging Web Services technologies (in 2001-2002) [9]. This has allowed us to gain initial experience in building a cross-organisational contract management system. The latest versions of the BCL abstract syntax are expressed as metamodels, with a view to automatically generating a significant part of the engine which executes expressions in the language. This paper investigates the applicability of the Pegamento toolset to this task.

We begin by restating MDA in terms of a single basic pattern that is then composed in several ways to represent the behaviour of a large range of MDA approaches and tools. Some discussion of the relationships between models and platforms arises here. Then we describe the BCA and BCL. The subsequent section presents the MDA toolset used and developed by the Pegamento project. This is followed by our pattern-matching analysis of how the MDA and Pegamento models and tools could be applied to building an engine for executing contract monitoring as specified by the BCL. The paper concludes with a summary.

2. The MDA Pattern

Most of the literature on MDA describes the mapping of Platform Independent Models (PIMs) to Platform Specific Models (PSMs) as the most general case of the MDA approach, and then goes on to specialise this "pattern" to include mappings of other kinds, such as translations between models at the same level of abstraction, mappings with multiple input or output models, or an additional level of mappings where the resulting PSM plays the role of a PIM in a second mapping step.

We generalise from the descriptions of an MDA pattern given in [1], [2] and [3] to a single simple case of two metamodels with a directional transformation definition between them Figure 1. The astute reader will notice that we use the words "metamodel" and "transformation" where others use "model" and "mapping". This is not because we disagree with the other authors about what actually takes place in an MDA tool, but because in their attempts to make MDA accessible to wide audience, they find it useful to blur the distinction between models and metamodels, at least initially, to allow them to phrase things in terms of a single application of the MDA pattern.

In our view, and we believe the authors mentioned concur with us, any model-based development which does operations on a single, unique instance of a model–usually a design of a specific software system–is using a pre-MDA approach. MDA is characterised by the ability to define or implement a mechanism that relates all valid models conforming to some metamodel, into a class of valid models conforming to another metamodel. Validity of models may be defined in various ways, but is usually based on conformance to some set of constraints or preconditions.

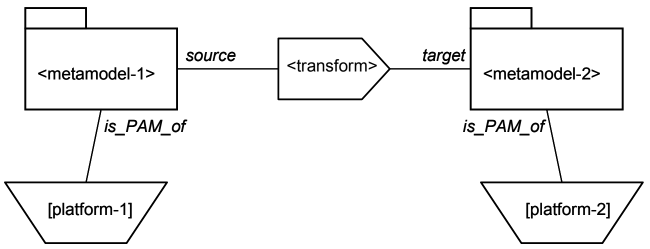


Figure 1: The Base MDA Pattern

3. What is a PAM?

Much discussion has been had in OMG and in other fora on the topics "What is a model?", "What is a platform?" and "What is a platform model?". We do not intend to attempt a full exposition of these topics here. However, it is possible to state that it is widely understood that the "PIM/PSM" pairing requires a statement of what platform these models are relative to. Conversely, given a platform, and some PSM language that allows models for that platform to be created, we can state that the PSM metamodel is a model of some aspect of that platform: a Platform Aspect Model (PAM).

As an example let us consider a mapping from classes in UML 1.4 to interfaces in CORBA 3.0. Let us instantiate the pattern in Figure 1. (A complete description of the pattern language follows in Section 3.1). First, we provide "UML 1.4 Metamodel" as the argument to fill parameter <metamodel-1>. Then we fill <metamodel-2> with "CORBA 3.0 IDL Metamodel", and [platform-2] with "CORBA 3.0". Finally, <transform> is replaced by "UML1.4 to CORBA 3.0 IDL". We ignore the optional parameter [platform-1] because characterising the kind of platforms for which UML designs are suitable would require "system" or something equally meaningless.

The invariant relationships "*source*" and "*target*" have an obvious meaning in relation to the directional transformation "UML1.4 to CORBA 3.0 IDL". And the relationship "*is_PAM_of*" explains that the metamodel for CORBA 3.0 IDL is also a model of only the interface definition *aspect* of the CORBA 3.0 platform, and ignores the protocol and language mapping aspects of CORBA (among others).

3.1 The Pattern Language

The pattern abstraction we have chosen is informally based on the work in [11]. In the notation, boxes of any shape represent (parameterised) named things and lines represent named relationships between them. Italicised text names are invariant relationships in the pattern. We use the UML style of writing the relationship name at the end far from the subject and close to the object of the relationship. Angle brackets denote a mandatory parameter, and square brackets an optional parameter.

In addition to the diagrams, the pattern language supports:

Parameter Renaming:	oldname/newname
Parameter Instantiation:	<param>/actual
Parameter Unification:	param1 = param2
Optional Parameter Mandating:	<param>
Optional Parameter Deleting:	X[param]
Renaming and Mandating	[name]/<manname>

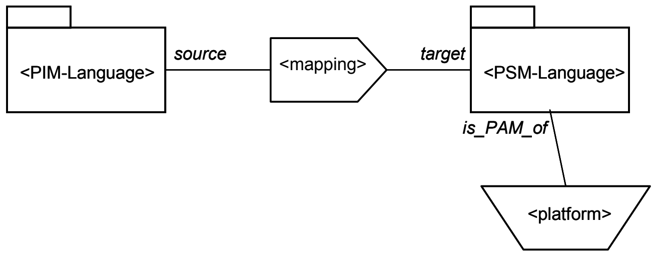


Figure 2: The PIM-PSM MDA Pattern

For example, the following instantiation of the Base MDA pattern in Figure 1 shows the PIM-PSM based pattern given in most MDA literature (Figure 2).

metamodel-1/PIM

metamoldel-2/PSM

X[platform-1]

platform-2/platform

Another example restatement is refactoring (Figure 3):

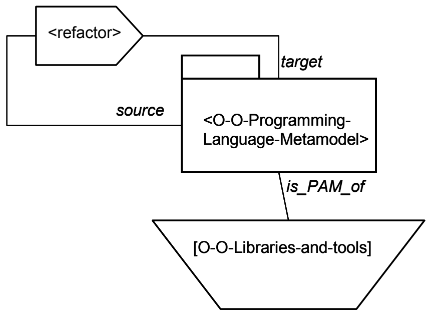
metamodel-1 = metamodel-2

platform-1 = platform-2

metamoldel-1/O-O-Programming-Language-MM

platform-1/O-O-Libraries-and-tools

transform/refactor



<refactor>
target
source
<O-O-Programming-Language-Metamodel>
is_PAM_of
[O-O-Libraries-and-tools]

Figure 3: The Refactoring Pattern

Patterns may, of course, also be instantiated, and we use the following instantiation as an example of a specific type of refactoring, the encapsulation of public properties by accessor methods (4):

<refactor>/Encapsulate-properties

<O-O-Programming-Language-Metamodel>/Java-Metamodel

<O-O-Libraries-and-tools>/JDK-1.2

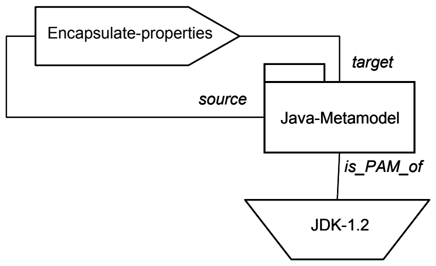


Figure 4: The Encapsulate Properties Pattern

3.2 Pattern Composition

The composition of two basic MDA patterns can also show the recursive nature of the approach. The Enterprise Collaboration Architecture (ECA) metamodel [12] is the key model in the so-called UML Profile for EDOC standard [31]. It models component encapsulation and interaction via synchronous operation invocation, choreographed asynchronous messaging, publish/subscribe, and process orchestration. In other words, it is an abstraction of the facilities of popular application server platforms such as J2EE, CORBA Components and Services, and .NET. Its sister standard, the Enterprise Application Integration (EAI) metamodel allows specification of lower-level messaging interactions to be executed in a Message-Oriented Middleware (MOM) platform such as Tibco, Websphere-MQ, and JMS. This is how the PIM-PSM MDA pattern can be applied to generation of a MOM application from an ECA model.

First we restate (and partially instantiate, delete and mandate) the PIM-PSM MDA pattern in terms of the ECA metamodel resulting in Figure 5:

<PIM-Language>/ECA-Metamodel
transform/ECA-to-App-Server
PSM-Language/App-Server-PAM
[platform]/<App-Server>

Then we restate the PIM-PSM MDA pattern in terms of mapping EAI to some MOM Application Server platform resulting in Figure 6:

<PIM-Language>/EAI-Metamodel
transform/EAI-to-MOM
PSM-Language/MOM-App-Server-PAM
platform/<MOM-App-Server>

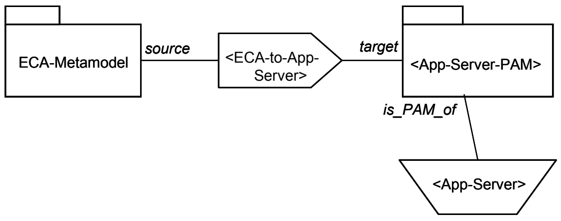


Figure 5: The ECA to App Server MDA Pattern

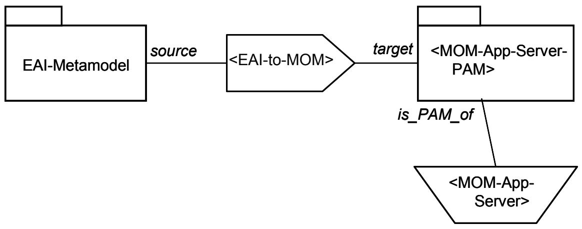


Figure 6: The EAI to MOM App Server Pattern

Now we can compose these patterns with a couple of parameter equalities, and instantiate the <ECA-to-AppServer> transformation resulting in Figure 6. Note that unification of a formal parameter with an instantiated actual parameter results in the actual parameter being used in the composed pattern.

<App-Server-PAM> = EAI-Metamodel
<App-Server> = MOM-App-Servers
<ECA-to-App-Server>/ECA-to-EAI

Note that when we use an intermediate model, the platform associated with it via *is_PAM_of* is more general than the platform associated with the final PSM. In this case EAI is a PAM of all MOM application servers, while the final PSM is a PAM of a specific MOM application server.

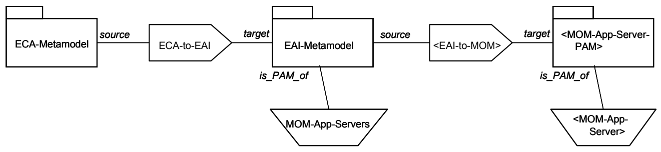


Figure 7: The ECA via EAI to MOM App Server MDA Pattern

4. Elemental Contract Management System

The Elemental project has focused on developing enterprise modelling concepts for intra- and inter-organisational structures and behaviour (Figure 8: below). An initial and joint activity with the Pegamento project was work on the UML profile for EDOC, mostly covering intra-enterprise aspects (1999-2001). Our subsequent work (2001-2003) addressed inter-organisational problems. As part of this we developed an architecture for supporting business contracts, primarily based on the Business Contract Architecture initially proposed in [21], and a Business Contract Language (BCL) for the specification of contract conditions for monitoring purposes.

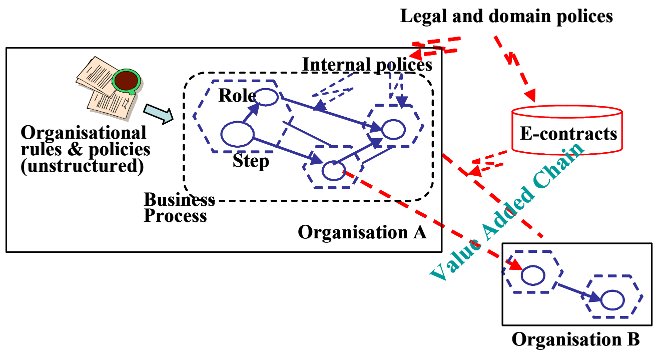


Figure 8: The Elemental Vision in 1999

4.1 Business Contract Architecture

The Business Contract Architecture supports the full contract life cycle and consists of the following components (see Figure 8, [25]).

- A Contract Repository, for storing standard contract templates, and optionally standard contract clauses as building blocks for drafting new contract templates;
- A Notary that stores evidence of agreed contracts (and their relationships) created during the negotiation process – to prevent any of the parties repudiating it;
- An Interceptor, providing non-intrusive interception of business messages exchanged between trading partners for further contract monitoring;
- A Business Activity Monitoring (BAM) component that processes events received from the interceptors, manages internal states related to the contract and provides access to various enterprise data sources needed by the Contract Monitor for policy evaluation;
- A Contract Monitor that evaluates contract policies to determine whether the signatories have fulfilled their obligations or whether there are violations to the contract; this component uses the BAM component for event pattern and state processing; it then sends appropriate messages to the Notifier component;
- A Notifier, for sending human readable notification messages to contract managers.

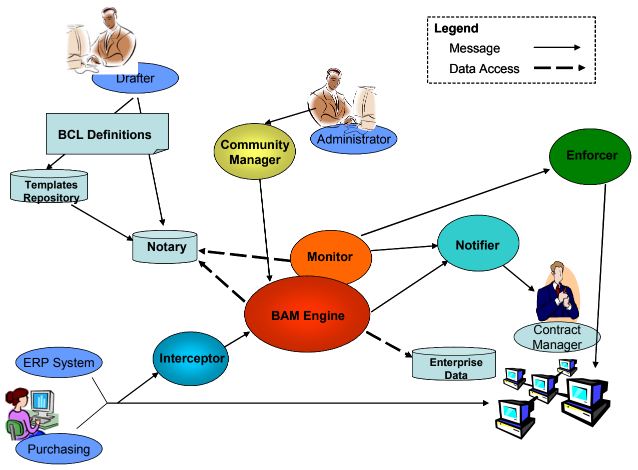


Figure 9: Business Contract Architecture

The components above constitute the core functionality needed for most contract management processes. Additional capabilities may be required for specific contract management systems. Examples of other possible components are:

- A Contract Enforcer, for implementing corrective measures if some violation has been detected;
- Contract Mediator and Arbitrator that can be used for discretionary contract enforcement;

A Community Manager, which allows the contract administrator to make dynamic updates of roles, policies and other community model elements; The BCA is easily configurable so that other components can be added as necessary.

4.2 Business Contract Language

This section describes Business Contract Language (BCL) for the specification of contract monitoring conditions. BCL can be used for the tailoring of BCA for the specific contract management environment.

4.2.1. Main characteristics

Domain Specific – BCL is a domain-specific language that introduces modelling abstractions which correspond directly to terms used in the contract management domain. It allows the unstructured text of contracts, stated in natural language to be re-expressed in a structured form, amenable to automated processing (Figure 10).

Declarative – BCL is primarily a declarative language whose notation allows the expression of contract domain concepts in a manner close to the way domain experts think. This allows the user to explicitly express their intention, the *what* of the problem, while the BAM engine takes care of the *how*

Event-driven – most of the execution in BCA is triggered by events. For example, states are updated in response to events, policy checking is triggered by events, and generation of internal events is driven by other events.

Model-based - this principle was adopted to ensure rapid and predictable development and deployment for specific contracting environments. This entails the use of:

- models to describe rules, structures and constraints of a specific contracting environment; the models expressed in BCL are used to parameterise the contract framework described below
- templates to represent *patterns* of structure and behaviour.

Figure 10 shows how the BCL configuration models parameterise the framework, producing a specific contract management system.

4.2.2. BCL modelling concepts

BCL language concepts can be grouped into three categories:

1. Community and Policies - these BCL concepts are introduced to define organizational, basic behavioural, and modal constraints associated with contracts. They directly map onto the terms expressed in natural language statement of contracts.

Organizational constraints can be expressed using a *community model* [25] that specifies the *roles* involved in a contract and their relationships. The roles can represent organizations participating in an overarching community, or they can be within organizations. A *community template* and associated *instantiation rules* specify conditions for the creation of a *community*. This is mirrored by the notion of a *contract template* as a basis for the creation of the corresponding *contract* instances.

Basic behavioural interactions between roles in a contract express the ordering of actions or steps in a business process carried out by the signatories to a contract. In BCL, most basic behaviour constraints are expressed using event patterns.

Similarly, *policies* apply to the roles involved. Most are modal constraints such as obligations, rights, permissions, prohibitions, and authorizations. Policy conditions are also generally expressed in terms of event patterns.

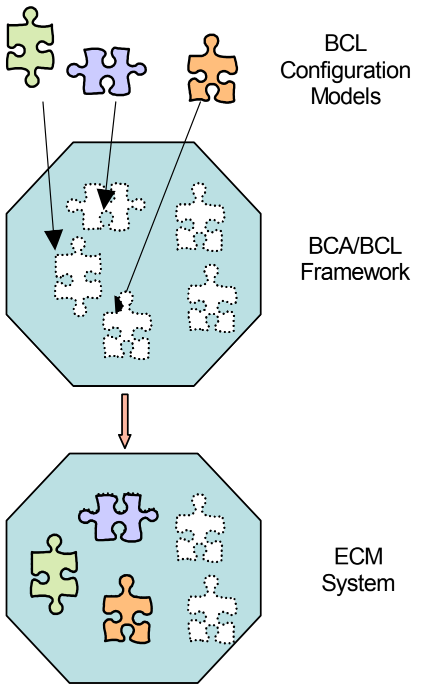


Figure 10: Model based solution

2. Events and States – these BCL concepts are the building blocks used to describe community models and contract monitoring conditions. They include:

- *event types* – to be created when certain conditions have been matched, e.g. creation of contract violation or contract fulfilment events.
- *event patterns* – for detecting specific contract-related occurrences, either as a single event or as multiple events related to each other;
- *internal states* and their changes in response to the events;

The purpose of event and state related concepts is to support real-time evaluation of the execution of basic behaviour and policies as stated in the contract with the aim of detecting contract violations or contract fulfilments.

BCL provides a rich set of options for expressing relationships between events, however their full description is beyond the scope of this paper. Representative examples of such expressions are [29]:

- Sequence of events - the event pattern is satisfied when all the events have occurred in the order specified in the sequence;
- Disjunction of events - the event pattern is satisfied when any of the events have occurred;
- Conjunction of Events - this pattern is satisfied when all the events have occurred;
- Quorum – this pattern is satisfied when a specified number from the set of all events have occurred;
- Event Causality - the event pattern is satisfied when the currently matched event has as its causal parent some previously recognised event.

The event pattern mechanism in BCL has many similarities to the specification of complex event processing, as described in [19].

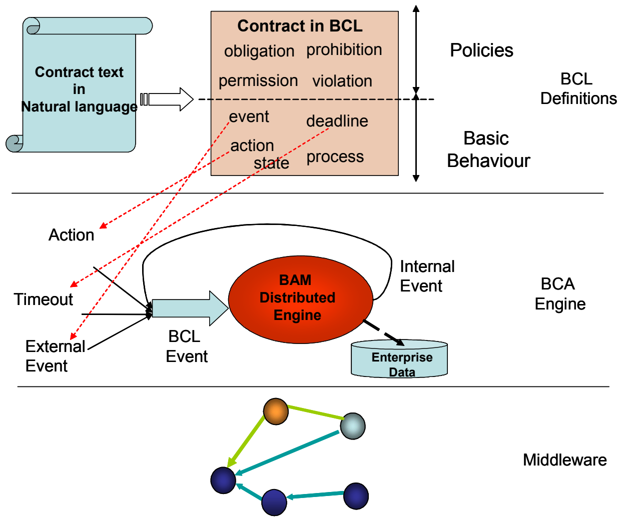


Figure 11: Elemental reality in 2004: BCL Models and BCA Engine

3. General language concepts - while the Communities and Policies, and Events and States aspects of BCL are used to express key concepts of the contracting domain we needed additional language constructs similar to typical programming languages.

4.3 Executing BCL

The BCL definitions for specific contract models will closely follow the expression of contract conditions stated in natural language contract text (see Figure 11).

The semantic model for the execution of these behaviour constraints is realised as part of the Business Activity Monitoring (BAM) component, which can be distributed, if necessary.

Once the BCL descriptions are submitted to the BAM engine it will respond to events as they occur. As Figure 11 shows, there are different types of events, such as external events resulting from the actions of people or systems, temporal events such as timeouts or events generated internally by the BAM engine. Often, as part of a condition evaluation, the BAM engine needs to access data from various enterprise repositories.

This monitoring design is quite generic and the BAM engine can be used to monitor execution of any business activity, whether directly related to a legally binding contract, or as part of internal business processes.

5. The MDA tools developed and used by Pegamento

The approach to software engineering of model mapping and code generation from models has been articulated by Czarnecki [17] among others before the coining of MDA as a term/brand. The DSTC Pegamento project proposed such an approach at the beginning of 1999, captured graphically in Figure 12.

5.1 MOF

The initial MDA-like approach was influenced by our involvement in the MOF standard [16] of OMG, which we quickly began using as the basis for language design for type management, for object-oriented databases, and for software design in cases where UML had known problems (for example software component and workflow design). Our prototype MOF repository tools were used to bootstrap the engineering for the standards-compliant dMOF product [32].

5.2 Human Usable Textual Notation (HUTN)

The HUTN (pronounced hootin') was a DSTC initiative in the OMG to allow a human-friendly grammar (compared to the XML syntax), and parsers and pretty printers, to be generated from any metamodel. These are used for the creation and browsing of instance data conforming to that metamodel.

The HUTN format is structurally related to XMI, but has a concise Java-like syntax [6].

5.3 Anti-Yacc

Although generating standard grammars is useful for new MOF-based languages, many metamodels used in MDA software development represent languages which already have grammars defined: CORBA IDL, Java, XML, C# etc. For these languages we must be able to output models as text that can be parsed by compilers and other tools.

Anti-Yacc [5] is essentially a pretty-printer for MOF models that uses EBNF grammars with embedded navigation through the MOF metamodel representing the language. It is an essential component for integration between modelling- and text-based software engineering tools. It enables us to use MOF models to represent everything in our software environment – including code.

5.4 EDOC/ECA Metamodel

In March 1999 the UML Profile for Enterprise Distributed Object Computing (EDOC) RFP called for a modelling abstraction of all the concepts embodied by modern application server platforms: encapsulated components whose threading, transactional and persistence properties were managed by the platform, choreographies of these components, publish/ subscribe and other kinds of messaging, business process definitions. The stated intention was that the models created in such a language could be automatically mapped to several application server platforms, and that this should be demonstrated in responses to the RFP.

5.4.1. Model Apocalypse

In late years of last century UML ran aground upon a paradigm shift in software development. Objects were out, components were in; methods were out, loosely coordinated asynchronous messaging was in; sequences of invocations were out, business process coordination was in. In the modelling of structure UML had no good way of representing components of the kind that COM, CORBA and, to a lesser extent, EJB were implementing in their platforms. Even the name "Component" in the UML metamodel was used to define physical configuration.

In short, UML was not prepared for the advent of MDA, because the metamodel, the extension mechanisms, and the graphical tools all assumed that a human was driving the development process. This meant that well-formedness wasn't important – the human who drew the picture knew what the diagram meant, and she could derive the right component interface or coordination description for the platform. She could even write plugins to the tools that made this semi-automatic.

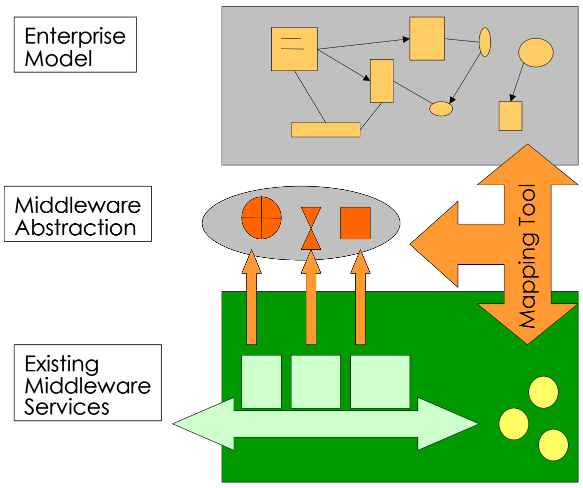


Figure 12: The Pegamento vision in 1999

5.4.2. Business Object Facility (BOF)

Leading up the issuance of the EDOC RFP, the OMG had taken the rare step of blocking the adoption of a specification known as the BOF. This was a language, expressed as a layer above CORBA, which allowed the capture of recursively structured "components" which represented business concepts.

5.4.3. A New Metamodel (and a Graphical Approximation)

It soon became obvious that the semantics of UML were impossible to reuse for the specification of EDOC without breaking all the principles of object-orientation. Therefore a new metamodel was defined which gave the correct semantics. However, the submitters found that the Collaboration metamodel of UML was very permissively interconnected, which allowed recursive composition and component port structure to be simulated. The ability to attach state machines to any UML model element made it possible to define protocols for these ports. In short, it was possible to reuse the structure of UML, while ignoring much of its semantics, thereby facilitating the use of UML graphical tools.

The adopted submission to the EDOC RFP actually consisted of six metamodels and UML profiles. The Enterprise Collaboration Architecture (ECA) is the key metamodel (also expressed as UML profile).

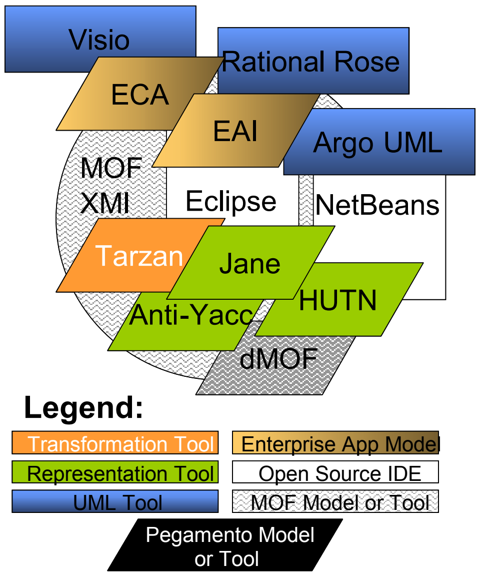


Figure 13: Pegamento reality in 2004

5.5 EAI Metamodel

Simultaneously with the EDOC RFP, submissions were sought in OMG for modelling of messaging-based enterprise application integration solutions. The metamodel subtypes and extends the FCM model from EDOC, and defines message typing, and various approaches to the transmission, reception, interception, queuing, aggregation, filtering, routing and transformation of messages in an application integration scenario. Complex message manipulation structures can be composed by the modeller from simpler operators defined in the metamodel.

5.6 Tarzan

Tarzan is the working name for a transformation engine which implements the transformation language specified in the DSTC revised submission to the OMG's MOF 2.0 QVT RFP.

The Pegamento team has prototyped three generations of transformation engines for MOF models. The first of these approaches was known as generator-generators (gen-gens for short). Let us chose a simple case where a metamodel, MM1, had a transformation description relating it to another metamodel, MM2. The description of the transformation was read by the gen-gen, which generated a specific model generator for MM2 models, which needed an instance of an MM1 model as input. The gen-gen was coded in Java, accessed the CORBA-based dMOF product, and generated a Java tree-walker.

The next approach we attempted swung the pendulum too far in the other direction on the compiler/interpreter axis: we used F-Logic to code transformations directly as predicates representing the transformation, and exported the metamodels and the source model instances as fact-bases, from which the transformations could deduce the target models [7].

Tarzan is the next generation engine which implements the same QVT semantics as the F-Logic-based engine, with a few refinements, as reflected in our first revised QVT submission. It is implemented in Java using Eclipse to access EMF models and metamodels using their generated APIs (these models can be imported from MOF tools as XMI with XSLT applied [15]). It implements pattern matching by expression solving with backtracking. Its inputs are a set of source models and a transformation description, and it outputs are a set of target models.

Object creation is done using object proxies with identity which aggregate the target side property and type changes implied by all of the rules related to an object. At the end of the execution of transformation, real objects are created based on the proxies, and returned to the invoker of the transformation.

5.7 JANE

The ability to easily input and edit models in a graphical tool environment has largely driven the use of UML profiles for models that are not directly UML-based. We are now extending the HUTN approach to the generation of Human Usable Graphical Notations (HUGN). This work facilitates the generation of graphical notations for arbitrary metamodels, in a similar way to the generated grammars and parsers facilitate this approach for textual notations.

JANE is the name of the Pegamento sub-project that is currently developing a tool of the same name which creates Eclipse model editor plugins [23]. Mappings of default box-and-line notations to class-and-association concepts can result in an ugly, but usable, first approximation of an editor for creating models. We are yet to investigate exactly what kinds of customisation model it is necessary to support in order to allow graphical elements to more intuitively reflect model semantics.

6. Applying the MDA pattern to BCA

The Basic MDA Pattern shown in Figure 1 can be applied to the development of tools and implementations for an ECM system at at least two layers of modelling abstraction. Although the BCA is model-based, this does not necessarily make development of a BCA system amenable to MDA approaches. This is discussed specifically in Section 6.3.

In order to demonstrate the relationship between these, and to introduce a notation for comments, we introduce the "Is Instance Of" pattern in Figure 14.

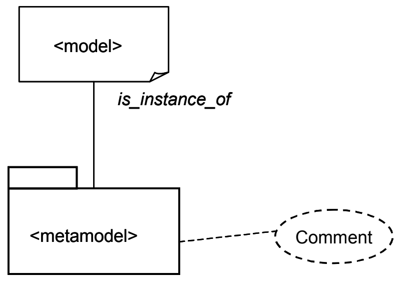


Figure 14: Is Instance Of Pattern and Comments

This relationship is specifically limited to a model being an instance of a metamodel. Comments are purely for annotation purposes, and are not formally manipulated in the Pattern Language.

6.1 Generic Matches

To fill the Contract Repository component of the Business Contract Architecture, the MOF and its standard mappings may be used, with the only requirement being that the BCL's abstract syntax is expressed as a MOF (meta-) model.

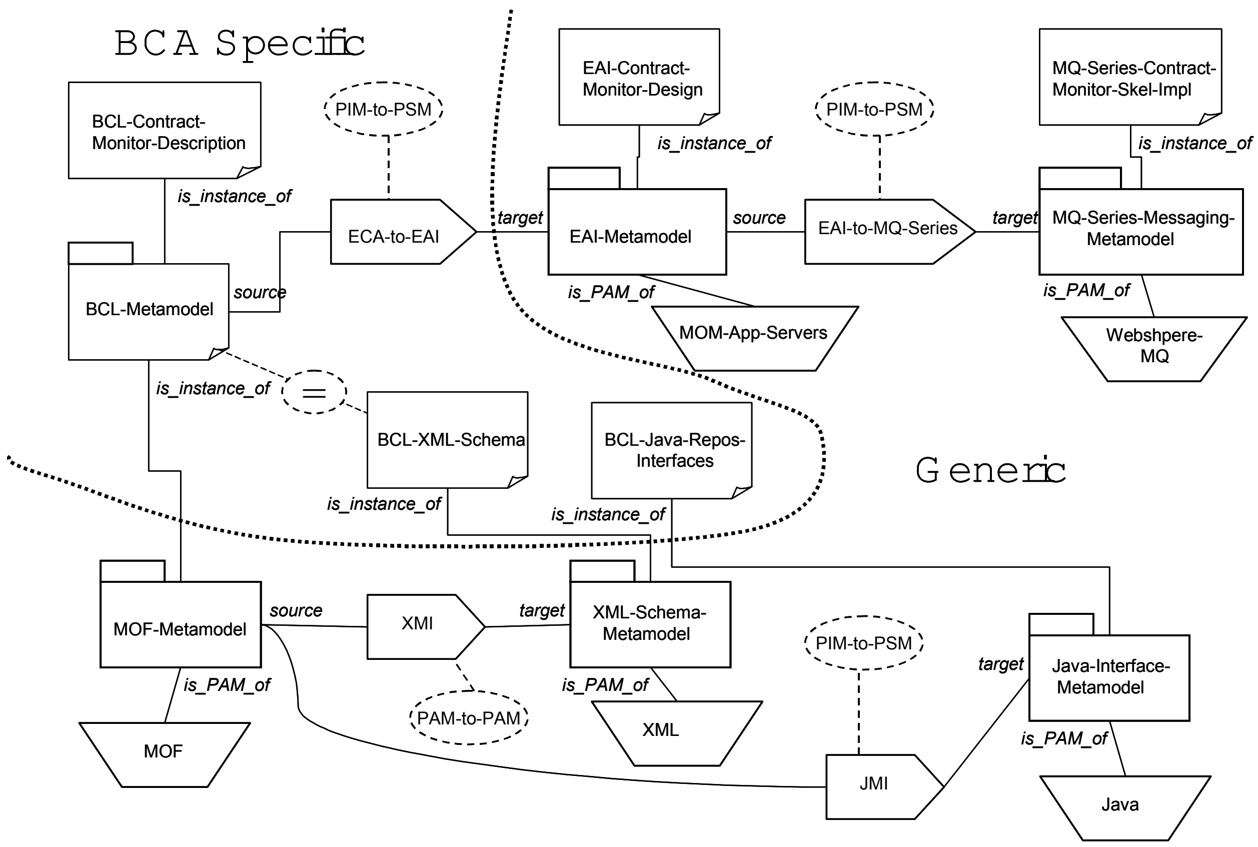


Figure 15: BCL Engine MDA Pattern Matches

6.1.1. Model Transformations

In the case of a BAM designed for use in a Java environment, we would probably use:

XMI – a semantics and structure preserving transformation which creates an XML Schema for documents expressing BCL contracts. We characterise this as a PAM to PAM mapping, as the transformation operates on the metamodelling aspect of the MOF platform, and produces an expression of the document type modelling aspect of the XML platform. We need to instantiate the Basic MDA pattern to represent XMI:

<metamodel-1>/MOF-Metamodel

<metamodel-2>/XML-Schema-Metamodel

<transform>/XMI

<platform-1>/MOF

<platform-2>/XML

Many platforms also generate a streaming function for a repository that is capable of exporting any model in the MOF repository for a metamodel (BCL in our case) and importing valid XMI documents into the repository.

JMI – a mapping from the MOF, which has a repository functionality defined independent of programming language access mechanisms, to Java interfaces defined for access to instances of a particular metamodel. We characterise this as a PIM to PSM transformation and so instantiate the PIM-PSM pattern, and compose it with the XMI pattern instance:

<PIM-Language> = MOF-Metamodel
<PSM-Language>/Java-Interface-Metamodel
<transform>/JMI
<platform>/Java

As with XMI, the tools associated with mapping MOF metamodels to Java interface types are also capable of creating libraries that implement the interfaces, create data repositories, and give access to instances stored in them.

Two Is-Instance-Of patterns are instantiated (and composed by virtue of the equality of the names chosen). They are both example instances that would result from having the BCL Metamodel as the source of the transformation:

<metamodel>/XML-Schema-Metamodel
<model>/BCL- XML-Schema
<metamodel>/Java-Interface-Metamodel
<model>/BCL-Java-Repos-Interfaces

6.1.2. Tool Choices

dMOF fully supports XMI, but version 1.1, which is based on XML DTDs rather than schemas. In addition, it is a CORBA-based tool, and so the pattern for mapping to Java interfaces must go through the intermediate step of mapping the MOF (meta-)metamodel to CORBA IDL, and then applying the IDL to Java mapping.

Netbeans supports JMI directly with its MDR repository, as well as XMI. This implements the transformations in Figure 15 most directly.

However, in order to reuse existing code from previous prototype versions of BCA implemented in WebSphere, we would probably adapt the mappings and models above to use EMF, Tarzan, and Eclipes' other tools and metamodels. As Websphere is a branded version of Eclipse which comes with a more robust J2EE environment, any EMF models and Eclipse plugins that manipulate them may be directly integrated with previous BCA code.

6.2 BCL-Specific Matches using EAI

The aim of the EAI metamodel (Section 0) is to model messaging-based application integration, in which adaptors are attached to existing applications in order to expose events occurring inside them. Data is exchanged with other applications as messages delivered via a message-oriented application server platform. These messages can be manipulated in various ways to make them applicable to all parties in the integrated application. This is a convenient match with the BCA, which intercepts events from parties to a contract, and using the BCL, expresses conditions upon which parties are notified of relevant occurrences, such as contract violations or fulfilments, corrective actions, or state changes.

6.2.1. The BCL and EAI patterns

The benefit of using the EAI metamodel to express an ECM design is that mappings exist, or are under development, from the EAI metamodel to a range of platforms such as Websphere-MQ, Oracle Application Server 10, and Fujitsu Siemen's openSeas. In Figure 15 we have chosen MQ-Series Messaging – now a part of Websphere – for the same reasons as discussed above for favouring Eclipse: tool integration and code reuse. We have fully instantiated the EAI to MOM App Server Pattern pattern shown in Figure 6:

```
<EAI-to-MOM>/EAI-to-MQ-Series
<MOM-App-Server-PAM>/MQ-Series-Messaging-
                                Metamodel
<MOM-App-Server>/MQ-Series-Messaging
```

and composed it with a BCL restatement of the PIM-PSM pattern:

```
<PIM-Language>/BCL-Metamodel
<transform>/BCL-to-EAI
<platform> = MOM-App-Servers
<PSM-Language> = EAI-Metamodel
```

Then three Is-Instance-Of patterns are instantiated (and composed by virtue of the equality of the names chosen) to represent BCA example instances at various stages of transformation:

```
<metamodel>/BCL-Metamodel
<model>/BCL-Contract-Monitor-Description

<metamodel>/EAI-Metamodel
<model>/EAI-Contract-Monitor-Design

<metamodel>/MQ-Series-Messaging-Metamodel
<model>/MQ-Series-Contract-Monitor-Skel-Impl
```

The final instance is a skeleton implementation that deals with the event-handling aspects of the BCL-Contract-Monitor-Description.

6.2.2. The Meta-level Bridge

The link between the generic MDA pattern matches relating to the MOF, and the BCL-specific pattern matches can be made using the Is-Instance-Of pattern:

```
<metamodel>/MOF-Metamodel
<model>/BCL-Metamodel
```

Note that the BCL (meta)model plays the role of model in relation to the MOF (meta-)metamodel, but the role of metamodel in relation to the BCL contract monitoring description.

6.3 BCL MDA Pattern Mismatches

Figure 10 shows the ECM that implements the BCA as a jigsaw with missing pieces; a template framework which requires actual contract definition parameters for it to operate on. In [25] the language's own templating and instantiation definition mechanisms are also explained. Effectively they allow a contract to be parameterised, and only partially instantiated before it takes effect, and the monitoring conditions in the BCL description can be executed in the ECM.

Even though the BCL can be represented as a metamodel, and contracts and their monitoring expressed as models, there is no easy way to treat templating, as an exemplar of MDA: either as an expression of a model/interpreter paradigm, or as the dynamic re-interpretation of a contract as more information about its progress becomes available.

7. Conclusion

We have provided a basic MDA pattern that expresses the relationships between models and platforms. We have also shown how to restate and compose this pattern to express a number of more complex MDA possibilities. We have used this as an analysis tool to investigate the application of the Pegamento MDA models and tools to the development of engines for contract monitoring, as expressed in the BCA.

The lessons learned are that MDA applies both in a generic way to any domain language expressed as a metamodel, as well as allowing for transformations to be defined mapping the domain semantics to appropriate platforms. We also see that model-based approaches are not always amenable to an MDA development solution. In particular, MDA does not directly address template-based models.

8. Acknowledgements

The work on modelling and transformation owes a great debt to the vision of Kerry Raymond, a DSTC founder and constant re-inventor.

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).

9. References

[1] A.Kleppe, J. Warmer, W. Bast, *MDA Explained*, Addison-Wesley, April 2003.

[2] David S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley Publishing Inc., 2003.

[3] J. Miller and J. Mukerji (editors), *MDA Guide Version 1.0.1*, document number omg/2003-06-01, OMG, 2003.

[4] F. Budinsky, et. al., *Eclipse Modelling Framework: A developers guide*, Addison-Wesley, 2004.

[5] David Hearnden, Kerry Raymond, Jim Steel, *"Anti-Yacc: MOF to Text"*, IEEE Enterprise Distributed Object Computing Conference (EDOC'2002), Lausanne, Sept 2002

[6] Jim Steel, Kerry Raymond, *"Generating Human-Usable Textual Notations for Information Models"*, IEEE Enterprise Distributed Object Computing Conference (EDOC'2001), Seattle, Sept 2001

[7] Keith Duddy, Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel*, "Model Transformation: A Declarative, Reusable Patterns Approach* Proc. 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC'2003), Brisbane, Sept 2003, pp 174-185

[8] James Cole, John Derrick, Zoran Milosevic, Kerry Raymond, *Author obliged to submit a paper before July 4: Policies in Enterprise Specification*, Policy 2001 Workshop, Bristol, Jan 2001.

[9] S. Kulkarni, Z. Milosevic, *Enterprise Integration through WebServices based Contracts Architecture*, OMG WebServices workshop, San Jose, March 2002.

[10] A.Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond and A. Wood, *Processes, Roles and Events: Concepts for an O-O Enterprise Architecture*, UML 2000, York, UK.

[11] K. Duddy (editor), *UML Profile for Patterns v1.0*, OMG document ptc/03-09-08, OMG 2003.

[12] K. Duddy (editor), *Enterprise Collaboration Architecture v1.0*, OMG document ptc/03-09-05, OMG 2003.

[13] K. Duddy (editor), *Flow Composition Model v1.0*, OMG document ptc/03-09-10, OMG 2003.

[14] K. Duddy (editor), *UML Profile for MOF v1.0*, OMG document ptc/03-09-11, OMG 2003.

[15] Anna Gerber, Kerry Raymond, *"MOF to EMF: There And Back Again"*, Proc. Eclipse Technology Exchange Workshop, OOPSLA 2003, Anaheim. USA, Oct 2003, pp 66-70.

[16] Stephen Crawley, Scott Davis, Jaga Indulska, Simon McBride, Kerry Raymond, *"Meta-meta is better-better!"*, IFIP Workshop on Distributed Applications and Interoperable Systems (DAIS), Cottbus, Germany, September-October 1997.

[17] K. Czarnecki and U.W. Eisenecker, *Generative Programming – Methods, Tools, and Applications*, Addison-Wesley, 2000.

[18] Z. Milosevic, G. Dromey*, On Expressing and Monitoring Behaviour in Contracts*, EDOC2002 Conference, Lausanne, Switzerland

[19] D. Luckham, *The Power of Events*, Addison-Wesley, 2002

[20] Jean-Michel Bruel, Brian Henderson-Sellers, Franck Barbier, Annig Le Parc, and Robert B. France. *Improving the UML metamodel to rigorously specify aggregation and composition*, in Proceedings of OOIS, pages 5--14. Springer-Verlag, August 2001.

[21] Z. Milosevic. *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Computer Science Dept. The University of Queensland, October 1995.

[22] ISO\IEC IS 15414, Open Distributed Processing-Enterprise Language, 2002.

[23] B. Moore, et. al., *Eclipse Development using the Graphical Editing Framework and the Eclipse Modelling Framework*, IBM Redbooks, February 2004.

[24] Oracle Contracts, http://www.oracle.com/appsnet/products/contracts/content.html.

[25] Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, S. Neal, *A unified behavioural model and a contract language for extended enterprise*, Data Knowledge and Engineering Journal, Elsevier Science, to appear.

[26] S. Neal, J. Cole, P. F. Linington, Z. Milosevic, S. Gibson, S. Kulkarni, *Identifying requirements for Business Contract Language: a Monitoring Perspective*, IEEE EDOC2003 Conference Proceedings, Sep 03.

[27] W-Jan van den Heuvel, H. Weigand, *Cross-Organisational Workflow Integration using Contracts,* Decision Support Systems, 33(3): p. 247-265

[28] Z. Milosevic, A. Josang, T. Dimitrakos, M.A. Patton, *Discretionary Enforcement of Electronic Contracts*. Proc. EDOC '02. pp(s): 39 -50. IEEE CS 2002

[29] Z. Milosevic, S. Gibson, P. Linington, J. Cole, S. Kulkarni, *On design and implementation of a contract monitoring facility*, the first IEEE Workshop on Electronic Contracting, San Deigo, July 2004.

[30] ww.jcp.org/jsr/detail/40.jsp

[31] www.omg.org/cgi-bin/apps/do_doc?ptc/03-09-04

[32] www.dstc.edu.au/Products/CORBA/dMOF

[33] www.oasis-open.org/committees/legalxml-econtracts/charter.php