Chapter 7

MODEL BASED TESTING AND REFINEMENT IN MDA BASED DEVELOPMENT

Ian Oliver
Nokia Research Center
Helsinki
Finland

Abstract The Model Driven Architecture's key principle is that of the mapping. An algorithmic or otherwise mechanical way of generating new more platform specific models from platform independent models with respect to some platform. These mappings are always presented as devices driving the software development. However it is clear that there are a number of uses for mappings and that the idea can be extended to take into consideration not only software development but transformation between differing underlying representations.

Mappings also have a key rôle in the methodology and in the way tests are conducted. Development is coupled with the notion of refinement - that is a mathematically strict way of ensuring certain (critical) properties from the abstract to concrete models. To fully understand and utilise the mappings it is necessary to construct and formalise a framework for these mappings and their meanings (particularly in testing with refinement).

1. Introduction

Testing is probably the most critical issue with regards to software development but is one of the most lacking areas in terms of practise [Binder, 2000]. Technologies such as model based testing [Offutt and Abdurazik, 1999], refinement and so on, are all well known; integrating these together is a critical task for software engineering. There are a number of issues particularly when integrating refinement that need to be discussed.

Model Driven Architecture (MDA) is a proposal by the Object Management Group[1] for a development framework in which the logic of the system is separated from the logic of the underlying platform. The key points about the MDA is that it formalises the relationship between that of a model and that of the mapping between a pair or more of models by encoding algorith-

P. Boulet (ed.) Advances in Design and Specification Languages for SoCs, 107–122.
© 2005 *Springer. Printed in the Netherlands.*

mically methodological ideas and concepts. The idea, while arguably not revolutionary is now practical because of the existence of a standard, extensible modelling language (UML), domain specific meta-models and thus languages, a meta-modelling framework (MOF) and standardised model interchange formats (nominally based upon XML). This notwithstanding the development of sophisticated processes, methods and experience of the software engineer.

Model Based Testing (MBT) is a development concept where the validation and verification tests are generated directly from the models of the system under development. Refinement is a well known, formally defined method for ascertaining whether certain properties of a system are preserved across development. However refinement as seen in methods such as the B-Method is very strict and tied to one particular aspect of the model. Model based testing on the other hand deals with many aspects of a model.

In this paper we describe how model based testing, model driven architecture and the notion of refinement combine. We do not attempt to provide a full mathematical treatment of this composition but to outline a number of important issues when working with these technologies.

2. MDA Taxonomy

The MDA is a complex structure which takes into consideration many aspects of modelling such as the language, semantics and model management. In figure 7.1 we show a *simplified* representation of the MDA meta-model written using UML.

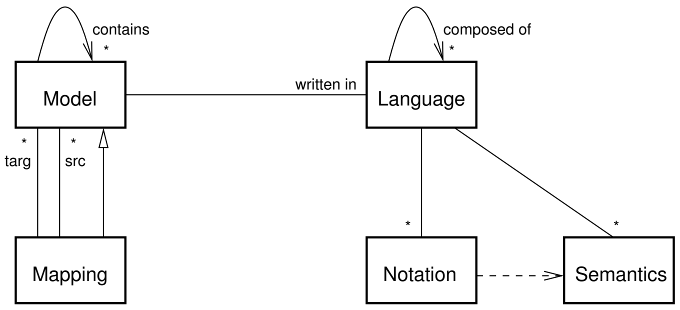

Figure 7.1. MDA Meta-Model

From this model we can clearly see the separation of concerns provided by the MDA and embodied in the technologies on which it is based. For example the UML [OMG, 2002b] makes the distinction between notation and semantics. The UML is supplied with a weak semantics enough to suit nearly all development tasks and extensible enough for it to be customised to most domains.

The meta-model shown here we have reconstructed from our experiences in using the MDA and MDA-like approaches [Oliver, 2002b]. The primary issues are the creation of a structure of mappings and the explicit representation of the structure of a language.

2.1 Mapping Taxonomy

The mapping is the fundamental construct of MDA. The key point about the mappings in the MDA is that they are of semantic nature and not syntactic nature - that is they map the concepts in one language to the concepts in another preserving the meaning. This in unlike the traditional syntactic mappings found in many tools, for example, those that map UML classes to C++ or Java classes - this is of course fine *if* the semantic gap between the diagram and the code is almost non-existent.

The MDA as it stands does not define any taxonomy of mappings, this we feel leads to some confusions about what a mapping is and what can be performed by a mapping.

We therefore introduce a *simple* taxonomy[2] of MDA mappings based upon the idea that mappings can be broadly classified into three types: development, transformational and code-generation. These can be seen in figure 7.2 - taxonomy in black, MDA meta-model in grey.

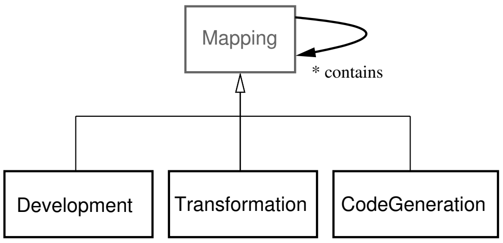

Figure 7.2. MDA Mapping Taxonomy

Mappings may contain other mappings - particularly in the case of development mappings which may utilise a number of transformational mappings to produce their result. This relationship also solves the problem (discussed below) of the transformation vs code-generation mapping - a transformation mapping may contain a (or many) code-generation mapping(s).

Of course there can be much discussion about the structure of this taxonomy and its classifications. One point certainly relates to code generation mappings: are they development, transformational *or* some subtype of a transformational mapping? We do not discuss these issues here as classifications have too much of a philosophical nature.

Refactoring [Fowler, 1999] we consider the situation where we wish there to be no formal mathematical connection between the source and target models of the mappings. Refinement is the strongest of all the properties and insists that strict relationships between the models exist. Retrenchment is presented as a generalisation of refinement that deals with the situation where requirements may change but we wish to preserve the *ideas* of refinement.

In this paper we concentrate more on the notions of refinement and retrenchment and what it means within the context of an MDA development process that requires those properties to hold.

In [Oliver, 2002a] is a description of the space in which a model exists known as the model matrix shown in figure 7.3. Here we can clearly see that a model exists in a many dimensional space corresponding to various aspects of the state and meaning of that model at any particular time. We concentrate here on just the 'vertical' and 'horizontal' axes to which we give the names

- Development or Vertical Mappings (shown in the model axis)

- Transformation or Horizontal Mappings (shown on the support axis)

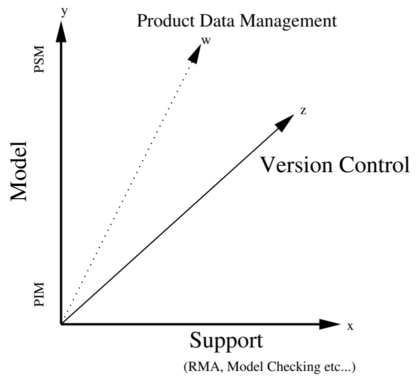

Figure 7.3. Model Matrix

We consider vertical mappings those that are conventionally in MDA parlance thought of as platform independent models (PIM) to platform specific models (PSM) mappings, that is, those that change the abstraction level. Horizontal mappings we consider not to change the abstraction level. Mappings into programming languages are discussed separately.

2.2 Development Mappings

Development mappings are those which change the abstraction level of the model. That is they map platform independent models to platform specific models. It is important to note that the terms 'platform independent' and 'platform specific' are *adjectives* - they describe the relationship between any pair of models in the development process rather than what a particular model is.

An important property of the development mappings is that they do not necessarily imply any change of language in which the models are written. It is conceivable that a model written using, for example UML 1.5 Core, will continue to be written in UML 1.5 Core complete with the same semantics throughout the development process - in this case the development mapping only adds more detailed information into the model.

Normally it is the case that the language - or at least the semantics of the language - changes to reflect the increasing concreteness of the model. For example very platform independent models talk of classes in the broad object oriented sense while platform specific models may introduce more concrete semantics such that the notion of a class becomes closer to that of a database table, VHDL process [Marchetti and Oliver, 2003] or C++ class [Stroustrup, 2000] for example. There is still much open research on development mappings and how they are implemented and what information is supplied. Broadly speaking they can be considered the mapping of the structure, behaviour and other aspects onto the architecture of system (at that level of abstraction) [Boulet et al., 2004, Siikarla et al., 2004].

2.3 Transformation Mappings and Model Based Testing

Transformation mappings are those which do not change the abstraction level of the model but rather extract information from the model. This is a separate issue from a development mapping where the semantics of the language remains relatively comparable (eg: UML to UML-RT [OMG, 2002b, OMG, 2002a]). In transformational cases we have the situation where the language change can be very great. We can show two interesting examples of this and both relate to the issues surrounding 'model based testing' [Offutt and Abdurazik, 1999] and aspect-orientation [Elrad et al., 2002].

In the first example we can map models written using UML-RT into schedulability analysis models [Oliver, 2003] which may be analysed by using a technique such as rate monotonic analysis [Klein et al., 1993]. Here the nature of the mapping is defined such that each unit of execution (method, transition etc) is mapped into an RMA task along with certain dependencies. This model then requires the presence of another source model detailing the deployment architecture of the UML system model. From this deployment model we can ascertain where the scheduling points are in the model.

In the second example we map UML to a different modelling formalism - the formal specification language B [Abrial, 1995]. B however is not object oriented and its major constructor is that of a 'machine'. This machine construct fits neither the concepts of class, object nor component directly. The transformation mapping between UML and B as described in [Snook et al., 2003]. An example of an application of this mapping can be seen below as B code and the class diagram in figure 7.4[3]

```
MACHINE DSP0
/*" U2B3.6.12 generated this component from Package DSP0 "*/
SETS
  DSP={thisDSP}; CELL; CHANNEL; DSP_STATE={boot,init,idle,traffic}
CONSTANTS
  threshold
PROPERTIES
  threshold : DSP --> INT
DEFINITIONS
  disjoint(f)==!(a1,a2).( a1:dom(f) & a2:dom(f) & a1/=a2 => f(a1) union (a2)=0 )
VARIABLES
  dsp_state, current,dspChannels,powerlevel,cellChannels, broadcasting
INVARIANT
  dsp_state : DSP --> DSP_STATE & current : DSP +-> CELL   ...
INITIALISATION
  dsp_state := DSP * {boot} || current :: DSP +-> CELL ||  ...
OPERATIONS
  gotoinit  =
  BEGIN
     SELECT dsp_state(thisDSP)=boot
        THEN   dsp_state(thisDSP):=init ||
          ANY xx WHERE xx:CELL THEN current(thisDSP):=xx END
        END;
   ...
END
```

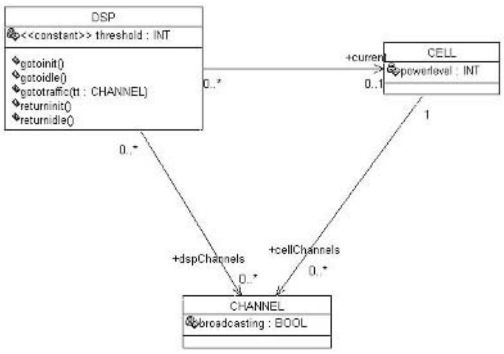

Figure 7.4. Simple Class Diagram

A *desirable* property of transformation mappings is the reversal of the mapping. While moving from one representation to another presents us with the opportunity to explore different aspects of the model, the relaying of the results in these models back to the original is necessary to allow full round-trip modelling between the two representations. In the examples given this means that the underlying semantics between the two representations is isomorphic with respect to the transformed aspects.

The ideas of model based testing can be clearly seen in the above example and how these ideas integrate. Each transformation mapping is a way of generating the test models from one or more aspects of the model. While utilising these models is generally simple for testing purposes, showing how these test models and their results fit together is more complex. This is where the ideas of refinement help in defining how the test models should be utilised with regards to the development of the system under test.

2.4 Programming Language Mappings

Programming language mappings present an interesting difficulty in this taxonomy as it is unclear exactly whether their are developmental or transformational in nature. If we take the case of a traditional UML modelling tool where class diagrams are annotated with methods, types, syntax and code that are of a given programming language, then certainly the mappings are transformational in nature. This is primarily because the mappings do not increase the concreteness of the model but just translate it in to a pure C++ or Java form - the model then is a just a graphical form of the programming language and the mapping purely syntactic in nature. A syntactic mapping places a fixed set of semantics on the nature of the relationship while a semantic mapping requires more information (this may be fixed of course) to resolve into what structure the relationship may be. In the case of Java this could mean Vector, Hashtable etc.

If the mapping requires information (from the architecture or platform definitions) to generate the more platform specific model in order to complete the mapping then the mapping is vertical in nature. Here the mapping is more semantic in nature usually.

3. Example of a Refinement Based Methodology

The PUSSEE project[4] constructed a methodology for the development of hardware systems using UML and formal development processes. The methodology discussed can be placed into an MDA context where the relationships between the models being produced are realised as MDA mappings. Pictorially the development process can be seen in figure 7.5

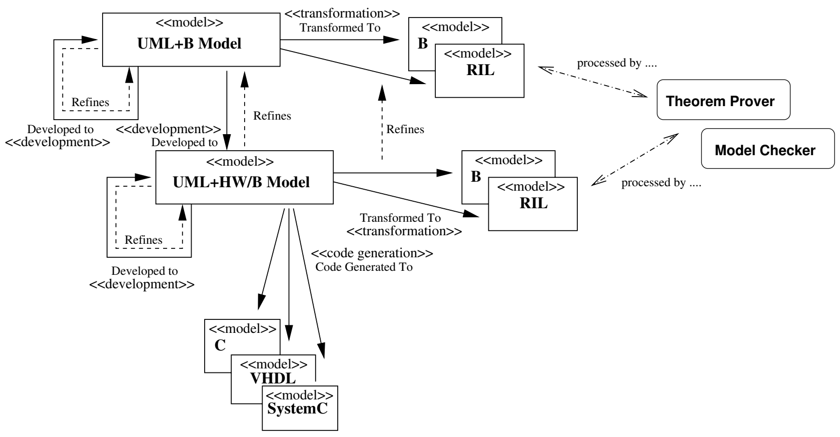

Figure 7.5. The PUSSEE Process

The formal aspects of the methodology are realised by utilising a profile of the UML [OMG, 2002b] known as UML-B [Snook et al., 2003] which integrates the B formal specification language [Abrial, 1995] into the UML as its action and constraint languages. These UML-B models are then verified for internal consistency and refinement properties [Morgan, 1990] by theorem proving mapping from the UML-B to a pure B form (which the user does not necessarily see). Subsequently this is then code generated into a target language such as SystemC, C or VHDL [Hallerstede, 2003]. In addition to this the models may also be mapped to the Raven Input Language (RIL) for model checking with Raven [Ruf, 2001]. Refinement is ensured between the B and RIL models. If both refine then this implies refinement of the UML-B model.

4. Refinement in MDA

Refinement [Morgan, 1990] is a property between models that states that one model is a) linked to a former, less-refined model and that b) the refined model reduces the state-space and non-determinism of the former model. Of course in reality the refinement link between any pair of models is more complex [Back, 1998].

Refinement therefore is a *property* of a relationship between two models (and not a development tactic as is sometimes believed). Referring back to the process described in figure 7.5 refinement is preserved if the following is true:

$$\forall m_1, m_2 : \mathit{Model} \mid m_2 \in \mathit{DevelopedFrom}(m_1) \bullet m_1 \sqsubseteq m_2$$

However this simple equation contains not enough detail - we must consider precisely **which aspects** of the model we wish to consider for refinement. This is particularly necessary in the case of UML which has no in-built method nor refinement semantics. In the case of embedded systems, only a few critical aspects at certain stages of development are amenable to refinement. For an embedded system this might be just the schedulability characteristics [Klein et al., 1993] or performance constraints. In the methodology described earlier this might then be described:

As the PUSSEE-*method* is based upon the B-Method then the refinement semantics are take directly from the B-Method. Here the refinement operator takes its semantics directly[5] from B-Method (specifically defined between the B models).

Given a model M_1 in UML-B we transform that model into its corresponding B representation, that model we denote B_1. Usage of a suitable B theorem prover proves the consistency of the model (but not necessarily its validity). If we develop M_1 to M_2 with some development mapping d which has the property that its target model(s) must refine the source model we are stating that any B representation of M_2, ie: B_2 must refine B_1 in the sense defined by the B-Method. Therefore we can state that $M_1 \sqsubseteq M_2$ iff $B_1 \sqsubseteq B_2$.

The case is similar when we map to our second target aspects - RIL. Again the method is similar: M_1 is mapped to R_1, M_2 is mapped to R_2, and the development step d preserves refinement iff $R_1 \sqsubseteq R_2$.

However the refinement property of the development step d and thus $M_1 \sqsubseteq M_2$ is **only** true if we are only considering *single* aspects of the model. The notion of refinement therefore has to be extended to take into consideration the multiple-aspects that may be explored during the development of the model.

5. Model Refinement Generalisation

We have so far presented the semantics of refinement of a model based upon the refinement of particular models that are transformed via an MDA mapping from the source model. We can generalise this approach and thus define firstly a number of types of refinement, that is refinement of particular aspects, and then a more global idea of refinement based upon the whole set of aspects being modelled and investigated.

Given a model M at any point in time we have a set of transformations T that extract particular aspects from the model. Not all available transformations need be applied to a model at every level of abstraction; deciding which to apply is governed by the development method employed.

We define a development mapping $d : Model \rightarrow Model$ over which refinement is preserved,ie: $m \sqsubseteq d(m)$. We also define model M_n where n denotes some point in time. For simplicity here we assume that n is an integer greater than zero and that time is a set of discrete values 1,2,3 and so on: $M_{n+1} = d(M_n)$. At any point in time we extract using a transformation mapping on a model a number of aspects of that model. For example given a model M where the set of transformation mappings T is $\{B, RIL\}$ we obtain A_B and A_{RIL} as models describing those particular aspects.

Our previous definition of refinement:

$$M_n \sqsubseteq M_{n+1} \Rightarrow \forall t : T \bullet t(M_1) \sqsubseteq t(M_2)$$

is too strong and can not take into account that not all transformations are applied. At any point in time we have a model M_n and a set of transformation applications A_n where $A_n \subseteq T$. Then refinement can be defined as:

$$(M_n, A_n) \sqsubseteq (M_{n+1}, A_{n+1}) \Rightarrow$$
$$A_n \subseteq A_{n+1} \quad \wedge \quad \forall a \in A_n \cdot a(M_n) \sqsubseteq a(M_{n+1})$$

If A_{n+1} contains more transformation mappings than A_n these can not be checked themselves for refinement as those members do not have any meaning in M_n and thus across the pair of models.

This definition also ensures that as more aspects are checked then their properties must refine across all subsequent model pairs ensuring the property that refinement is transitive.

This now gives us the definition of refinement across multiple aspects. Of course what refinement means for a particular aspect of the model still depends upon the methodology being employed for that aspect. In the case of B and B-method this is already defined. In the case for schedulability analysis [Klein et al., 1993] then as timing figures for a model decrease then refinement is preserved. Some aspects do not necessarily have a well defined semantics for refinement.

As we shall now see there are other considerations to make when working with refinement and the interaction between aspects when working with multiple aspects.

6. Other Considerations

Refinement as a property of development is highly desirable, however there are circumstances when the requirements do change and this then causes refinement to break. There are two particular tactics for dealing with this situation.

Also we have to consider the interaction between aspects. Aspects are often considered to be orthogonal in nature, however, there are situations where a change in the model which affects one aspect has repercussions for another aspect - this can be seen for example when working with schedulability and performance characteristics.

6.1 Requirements Volatility

One of the major problems with refinement is that it assumes that the models in question can be built in such a way that the final concrete model is a refinement of the first, most abstract model. While this approach has some very real advantages it is often the case that the initial set of requirements changes so that one model may not refine the previous source abstract model.

One solution to this is that of *retrenchment* [Poppleton and Groves, 2003, Poppleton and Banach, 2002] which introduces a relaxation of the rules of refinement to allow for the situation where the models do not refine. This situation can be easily catered for with MDA by providing information about the retrenchment through a separate model.

Retrenchment requires the use of the notion of concession which defines how the refinement relationship is weakened. In [Popplpeton and Banach, 2004] a semantics for retrenchment is given such that the retrenchment can be expressed in terms of a refinement with respect to some universal model in which the concessions are expressed. We can utilise this by creating an addition model which contains the concessions. This approach though is still experimental and support for dealing with the concession model does not exist at this time.

Another approach is to rebuild the models by layering the features that have been specified in the model [Back, 2002, Back, 2003]. A situation where the refinement property of the development step can not be attained then the model must be refactored [Fowler, 1999]. These techniques represent methods by which one can overcome certain refinement restrictions. It may be possible in the case of retrenchment to factor in the retrenchment ideas into the MDA structure and refinement definition given in this paper. At present we have not explored this in detail but the ideas are presented here to outline the future areas of research.

6.2 Aspect Interaction

Our definition of refinement so far assumes that the aspects of the model are orthogonal in nature. Some pairs (or more) of aspects may interact in such a way that a change in one aspect may cause a failure of the refinement obligation in another, seemingly, unrelated aspect.

This can trivially be seen if we are dealing with extracting memory usage information where an change of functional specification may cause the amount of memory consumed to increase, thus breaking the trivial memory refinement property.

Aspect interaction is one thing that makes model based testing and the approach discussed here more difficult. It is always necessary to examine the interactions. Categorically [Barr and Wells, 1990] this can be achieved by formulating a universal structure that describes the interaction between the two aspects. This is done by constructing a forgetful functor to each of the aspects to build the model of interaction. This is analogous to the retrenchment universal model in [Popplpeton and Banach, 2004].

In most situations we can avoid the necessity of constructing such models by ordering the transformation mappings for testing in such a way that certain aspects are only meaningful for refinement (of the particular system under development) after certain amounts of development have already been completed.

For example, for schedulability analysis or memory usage analysis, these aspects may need to be kept until later stages of development before refinement can be utilised.

7. Methodology

MDA places much more emphasis on method rather than just notation or process. The current MDA releases do not discuss in detail[6] the relationship between method and mappings which in our opinion is critical to the uptake and use of MDA principles.

We are developing a model of this relationship which is used to help in the development of methods which support MDA. The outline of this modelling framework can be seen in figure 7.6.

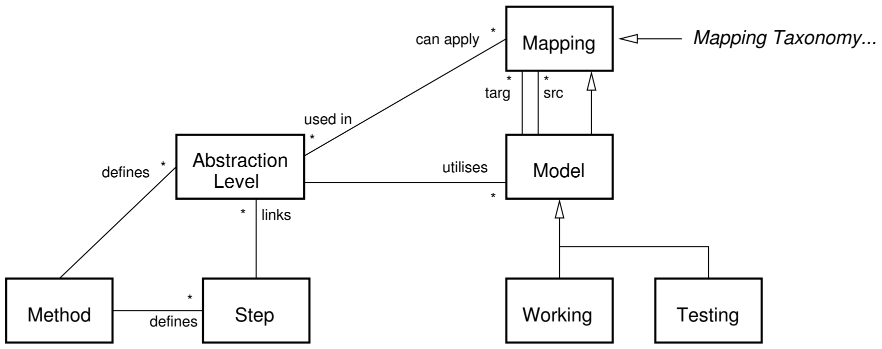

Figure 7.6. Method Meta-Model

We consider a method to consist of a number of method steps which act upon various abstraction levels (at least one or two). At each abstraction level there are a number of mappings that can be *sensibly* applied in that context. This is the first step towards a 'component based methodology' - an idea where methods are composed of a number of discrete techniques that can be composed together [Ambler, 2003].

We divide the concept of model into two parts, one for models of the system being built and one for models that facilitate some kind of testing. These basically correspond to those models generated by development mappings and those generated by transformation mappings respectively and thus demonstrating the relationship between the two basic types of mapping and the types of model produced.

The development properties related to formality becomes more interesting as we can assert on the sets of mappings for each abstraction level properties about how strict the mappings have to be with regard to more abstract models. It can be seen that weaker mappings (refactorings mainly) are best placed at very high abstraction levels where prototyping and very exploratory work can be carried out while stronger mappings such as those implying refinement are used at more concrete levels.

8. Conclusions

We have presented here the outline of how one may utilise and add the concept of refinement into MDA mappings to construct a formal development method and described a simple taxonomy of MDA mappings, their basic properties and their relationship to methodology. Model Driven Architecture is a very young and immature field although the concepts behind MDA can be traced back to the CASE idea of the 1970/80s. One of the reasons behind this immaturity is a large number of preconceived and unworked ideas regarding what a model is and what a mapping is.

Two areas of particular interest at the moment are extending the taxonomy to take into consideration operations [Alanen and Porres, 2003] such as model union, intersection and so on.

The secondary area of interest is the semantics of (a) method. In figure 7.6 we only define a structure state desirable properties of commutativity across the models produced with respect to the refinement and development relationships [Barr and Wells, 1990]. As the number of methods that do exist are plentiful the material to draw from here is large. One area of contention is that of the idea of a method step and a mapping - it is not clear whether these terms are actually different or whether they are isomorphic in nature.

Refinement offers much advantages when developing systems (usually in software) with regards to the ensured preservation of necessary requirements of the system being modelled. The addition of this concept into the MDA framework provides a placeholder to introduce the ideas of formal development in an MDA context. This we feel makes the idea more acceptable to the engineer who can still work with familiar notations and if truth be said in a desired rigorous way. Techniques such as retrenchment can be applied similarly and offer a weaker approach when required.

The problem of feature interaction can be overcome in two ways, one by letting the methodology take care of things and the other more rigorously through defining what is the interaction between two aspects. Both approaches we believe are complimentary and still require work before advances can be made.

For the developer of a method a placeholder for the semantics of the refinement required in a given context provides the chance to introduce formal development into a given MDA based method.

9. Acknowledgements

This work is partially supported by the EU Project PUSSEE - IST 2000-30103. Many thanks to Jean-Raymond Abrial for some useful comments regarding refinement and methodology.

Notes

1. `http://www.omg.org/mda`
2. This by no means is a complete or "one true" taxonomy - many variations do exist but have yet to be either documented or demonstrated. We defer the argument about what constitute a good or correct taxonomy here to concentrate on the basic framework rather than a philosophical discussion
3. NB: B code listing is shown only partially for space reasons and the ellipses (...) show the missing code):
4. EU Project: IST-2000-30103,Paradigm Unifying System Specification Environments for proven Electronic design, `http://www.keesda.com`
5. actually only in part but we shall discuss this later
6. and nor should they possibly

References

Abrial, J-R (1995). *The B-Book - Assigning programs to Meanings*. Cambridge University Press. 0-521-49619-5.

Alanen, Marcus and Porres, Ivan (2003). Difference and union of models. In *Lecture Notes in Computer Science 2863: <<UML>> 2003 Conference*. Springer. October 20-24.

Ambler, Scott W. (2003). The right tool for the job. *Software Development*, 11(12):50–52.

Back, Ralph (1998). *Refinement Calculus: a Systematic Introduction*. Springer-Verlag. 0387984178.

Back, Ralph-Johan (2002). SFI: A refinement based layered software architecture. In George, C and Miao, H, editors, *Formal Methods and Software Engineering: 4th International Conference on Formal Engineering Methods, ICFEM 2002 Shanghai, China, October 21-25. Lecture Notes in Computer Science 2495*. Springer.

Back, Ralph-Johan (2003). Software construction by stepwise feature introduction. In Bert, D., Bowen, J.P., Henson, M.C., and Robinson, K., editors, *ZB 2002: Formal Specification and Development in Z and B: 2nd International Conference of B and Z Users, Grenoble, France, January 23-25. Lectures Notes in Computer Science 2272*. Springer.

Barr, Michael and Wells, Charles (1990). *Category Theory for Computing Science*. International Series in Computer Science. Prentice Hall. 0-13-120486-6.

Binder, Robert V (2000). *Testing Object-Oriented Systems - Models, Patters and Tools*. Addison-Wesley. 0-201-80938-9.

Boulet, P., Cuccuru, A., Dekeyser, J.-L., Dumoulin, C., Marquet, Ph., Samyn, M., Simone, R. De, Siegel, G., and Saunier, Th. (2004). MDA for SoC design: UML to SystemC experiment. In Müller, Wolfgang and Martin, Grant, editors, *Proceedings of UML-SoC 2004, DAC2004, San Diego, California, June 6*.

Elrad, Tzilla, Aldawud, Omar, and Bader, Atef (2002). Aspect-oriented modeling: Bridging the gap between implementation and design. In Batory, D, Consel, C, and Taha, W, editors, *Generative Programming and Component Engineering: ACM SIGPLAN/SIGSOFT Conference, GPCE 2002, Pittsburgh, PA, USA, October 6-8, 2002. Lectures Notes in Computer Science 2487*, pages 189–201. Springer.

Fowler, Martin (1999). *Refactoring: Improving the Design of Existing Code*. Addison Wesley. 0201485672.

Hallerstede, Stefan (2003). Parallel hardware design in B. In Bert, Didier, Bowen, Jonathan P, King, Steve, and Waldén, Marin, editors, *Proceedings of ZB2003: Formal Specification and Development in Z and B. Lecture Notes in Computer Science 2651. Third International Conference of B and Z Users, Turku, Finland, June 2003*, pages 101–102. Springer.

Klein, Mark H., Ralya, Thomas, Pollak, Bill, Obenza, Ray, and Harbour, Michael González (1993). *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers.

Marchetti, Michele and Oliver, Ian (2003). Towards a conceptual framework for UML to hardware description language mappings. In *Proceedings of FDL03, Frankfurt, Germany, Sept 2002.*

Morgan, Carroll (1990). *Programming from Specifications*. Prentice-Hall.

Offutt, Jeff and Abdurazik, Aynur (1999). Generating tests from UML specifications. In *The Second International Conference on The Unified Modeling Language, Fort Collins, Colorado, USA, October 28-30*. Springer.

Oliver, Ian (2002a). Experiences of model driven architecture in real-time embedded systems. In *Proceedings of FDL02, Marseille, France, Sept 2002.*

Oliver, Ian (2002b). Model driven embedded systems. In Lilius, Johan, Balarin, Felice, and Machado, Ricardo J., editors, *Proceedings of Third International Conference on Application of Concurrency to System Design ACSD2003, Guimarães, Portugal*. IEEE Computer Society.

Oliver, Ian (2003). A UML profile for real-time system modelling with rate monotonic analysis. In Villar, Eugenio and Mermet, Jean, editors, *System Specification and Design Languages*. Kluwer Academic Publishers. 1-4020-7414-X.

OMG (2002a). *Response to the OMG RFP for Schedulability, Performance and Time*. Object Management Group, revised submission edition.

OMG (2002b). *Unified Modelling Language Specification)*. Object Management Group, version 1.5 edition. OMG Document Number ad/02-09-02.

Poppleton, Michael and Banach, Richard (2002). Controlling control systems: an application of evolving retrenchment. In *Lecture Notes in Computer Science 2272*. Springer-Verlag.

Poppleton, Michael and Groves, Lindsay (2003). Software evolution with refinement and retrenchment. In *Refinement of Critical Systems Workshop, RCS03*. Department of Computer Science, Åbo Akademi University, Turku, Finland.

Popplpeton, M R and Banach, R N (2004). Requirements validation by lifting retrenchments in B. In *Proceedings of ICECCS2004: IEEE International Conference on Engineering of Complex Computer Systems, Florence, Italy.*

Ruf, J. (2001). RAVEN: Real-Time Analyzing and Verification Environment. *Journal of Universal Computer Science*, 7(1):89–104.

Siikarla, Mika, Koskimies, Kai, and Systä, Tarja (2004). Open MDA using transformational patterns. In *Model Driven Architecture: Foundations and Applications MDAFA 2004, June 10-11 Linköping, Sweden*.

Snook, Colin, Butler, Michael, and Oliver, Ian (2003). Towards a UML profile for UML-B. Technical Report 8351, University of Southampton.

Stroustrup, Bjarne (2000). *The C++ Programming Language - Special Edition*. Addison-Wesley. 0-201-70073-5.