Clarifying the meta

Vladan Devedžić*

FON – School of Business Administration,
Department of Information Systems and Technologies,
University of Belgrade,
P.O. Box 52, Jove Ilića 154, 11000 Belgrade, Serbia
Fax: +381-11-461221 E-mail: devedzic@fon.bg.ac.yu
*Corresponding author

Dragan Gašević

School of Computing and Information Systems,
Athabasca University, 1 University Drive,
Athabasca, AB T9S 3A3, Canada
E-mail: dgasevic@acm.org

Dragan Djurić

Faculty of Information Technologies,
Mediterranean University,
Vaka Djurovica bb, Podgorica, Montenegro

FON – Department of Information Systems,
University of Belgrade, Jove Ilica 154, Belgrade, Serbia
E-mail: dragan@dragandjuric.com

Abstract: The word and the concept of meta are used in virtually all disciplines of computing. Professionals often use it intuitively and in specific contexts in which the meaning of a word containing the meta- prefix is well understood, at least by specialists. For example, all experts in artificial intelligence know what metarules are and where they are useful. However, the meta can imply much more than understanding some concepts in a specific domain. This paper surveys some of the most interesting cases of using the concept of meta in computing in order to help build the big picture.

Keywords: modelling; meta-modelling; model-driven architecture; semantic web; modelling spaces.

Reference to this paper should be made as follows: Devedžić, V., Gašević, D. and Djurić, D. (2008) 'Clarifying the meta', *Int. J. Information and Communication Technology*, Vol. 1, No. 2, pp.148–158.

Biographical notes: Vladan Devedžić is a Professor in and the Head of the Department of Software Engineering at FON – School of Business Administration at the University of Belgrade. His research interests focus on the practical engineering aspects of developing intelligent educational systems for the Web, while his long-term goal is to merge ideas from intelligent systems and software engineering.

Dragan Gašević is an Assistant Professor in the School of Computing and Information Systems at Athabasca University. His research interests include model-driven development, ontology development, the Semantic Web, XML-based interoperability, and learning technologies. He received his PhD in informatics and computer engineering from the University of Belgrade.

Dragan Djurić is an Assistant Professor at FON – School of Business Administration at the University of Belgrade, and an artist. His interests include model-driven development, enterprise software architecture, object-oriented development, the Java platform, intelligent information systems, and social-networking software.

1 Introduction

Metacomputing, metaprogramming, metaclasses, metaatributes, metadata, metasyntax, metalanguage, metainformation, metamodelling, metalevel, metareasoning, metarules, metasearch, meta-CASE tools, MetaCrystal, … Had enough? No? OK, here's more – metaphysics, metalogic, metamathematics, metaphilosophy, metatheory, metasystem, metamemory, metaconcept, metacognition, metameaning, metaheuristics, metauncertainty, metaontologies, metamorphosis, ...

The word and the concept of *meta* are used in virtually all disciplines of computing. Professionals often use it intuitively and in specific contexts in which the meaning of a word containing the *meta-* prefix is well understood, at least by specialists. For example, all experts in artificial intelligence know what metarules are and where they are useful. However, the *meta* can imply much more than understanding some concepts in a specific domain. Surveying some of the most interesting cases of using the concept of *meta* in computing can help build the big picture.

2 The meaning of *meta*

In ancient Greek, the word *meta* has several meanings. One of them is preposition taking the genitive case, which translates as the English *with* (Wikipedia, The Free Encyclopedia (http://en.wikipedia.org/wiki/Main_Page)). But there are other meanings as well – *changed in position*, *beyond*, *on a higher level*, *transcending*, etc. All these meanings are referring to the body of knowledge about a body of knowledge or about a field of study (Web Dictionary of Cybernetics and Systems, http://pespmc1.vub.ac.be/ASC/indexASC.html), such as in epistemology, where the prefix *meta-* is explicitly used to mean *about*. *Meta* is also often used for connoting purposes, as in *metamorphosis* where it connotes *change*.

FOLDOC dictionary of computing (http://foldoc.doc.ic.ac.uk/foldoc/index.html) defines *meta* even more precisely as "a prefix meaning one level of description higher. If X is some concept then meta-X is data about, or processes operating on, X". For example, if a computer stores and processes some data, then metadata is data about the data – things like who has produced the data, when, and what format the data is in. If we are studying a language, then metalanguage is a language used to talk about the language. The syntax of the language can be specified by means of a metasyntax.

3 Meta and meta-meta

Implications of the phrase “one level of description higher” are layering and hierarchy. Whenever we describe things at a certain level of abstraction, we use terms and concepts from a higher level of abstraction. For example, to describe a software system that models a process or processes from a real world, we can use UML concepts such as objects, classes, and relations. These are metalevel concepts used to describe real-world ones. The metalevel concepts are, in turn, defined at yet another higher level of abstraction, i.e., at a meta-metalevel. And so on.

A naturally arising question is: how far can we go on that way? In theory – very far. Consider some XML-based web resources that end-users are interested in. They are data. Consider also some other data derived directly from the first data, such as XML metadata and indexes of websites. These are metadata about the first data, but are also data in their own right. Deriving such metadata from data is performed using a metaoperation, such as different web mining and resource discovery functions. In theory, another (or even the same) metaoperation can be used on the metadata to derive meta-metadata. This is inherently recursive (Ip et al., 2005).

However, the recursion is never too deep if it is to make sense in practice. If the web resources the end-users are interested in are e-learning resources such as courses, lessons, lesson plans, curricula, teaching strategies and media (audio and video), then indexes of these resources are of interest as well (e-learning metadata repositories, digital ‘library cards’, and educational web portals). Metadata about these indexes can be of interest as well, and so can be some other kinds of metadata (such as how semantically similar are two or more resources). But it is currently difficult to envision many systems of practical value that would go much further than that in the meta-meta direction.

Although this *meta-meta depth factor* is never too high, different fields of computing implement the meta and meta-meta philosophy differently.

4 Metacomputing

Today, computing resources are often distributed among different sites. It implies different software environments, no file-system sharing (even on machines running the same operating systems), and different security policies. These shortcomings may be mitigated by using single very large computers instead of distributed resources, but it is an expensive solution.

Metacomputing is based on the idea of connecting many modest systems with a total power far outweighing the few supercomputers available. The whole can be greater than the sum of the parts. If a single application can run on two different parallel computers with 100 CPUs each, with some additional effort it can be made to run as a single 200-CPU application on a single logical *metacomputer* that physically comprises both machines. Likewise, there may exist a need to use two applications that are each suited to a different architecture of machine and combine them in a closely coupled manner. The applications can communicate with each other on a single metacomputer from the two machines using standard parallel programming techniques. Hence, metacomputing is essentially distributed computing, and is developed as a natural progress of supercomputing, parallel computing, cluster computing, and network computing.

A geographically separated collection of people, computers and storage connected through a fast network and supported by specific software is called a *metasystem* (http://legion.virginia.edu/presentations/metacomputing/). The software makes the entire collection of machines as easy to use as a single desktop computer. It also makes collaboration of the people involved very easy. High performance, security, fault tolerance, and transparency of such a metasystem are achieved in spite of its distributed nature.

A metasystem can run a *metaapplication* – a multi-component program, pieces of which are separate programs, optimised for running on different hardware platforms and computer architectures. Such a metaapplication can use big data sets that live on geographically remote sites, and can read/write files transparently regardless of execution location.

In other words, metacomputing allows multiple computers to tackle shared tasks. The metacomputer is a collection of computers held together by state-of-the-art technology and 'balanced' so that, to the individual user, it looks and acts like a single computer (http://www.epcc.ed.ac.uk/DIRECT/grid/node10.html).

Large-scale metacomputing involving massive computational tasks using high-speed networks is called *grid computing* (Allan, 2000; Foster, 2003). It is supposed to eventually transform computing from an individual and corporate activity into a general utility. Proposed software implementations for grid computing are layered ones (e.g., see http://www.gridforum/org/iga.html), including:

- low-level support for high-speed network i/o
- different grid services (such as authentification, authorisation, resource location, event services, etc.)
- application toolkits for data analysis and visualisation, distributed computing, collaborations, problem-solving environments
- grid-aware metaapplications implemented in terms of grid services and application toolkit components as building blocks.

Meta-meta depth factor here is usually 1 – typical perceived beneficiaries of metacomputing and grid technologies include governments (e.g., tasks, processes, and activities related to disaster response and national defence) and institutions (e.g., hospitals can use relatively low-cost and smaller-scale private grids with central management). However, the factor of 2 is envisioned as well in the form of 'virtual grid' (multi-institution collaboration) and 'public grid' (enormous community of service, resource, and network providers and consumers) (Foster, 2003).

5 Meta in programming

Metaprogramming is the writing of programs that write or manipulate other programs (or themselves) as their data or that do part of the work that is otherwise done at runtime during compile time (http://en.wikipedia.org/wiki/Metaprogramming_(programming)). The most common metaprogramming tool is a compiler. It allows programmers to write code using a high-level programming language and transform it into an equivalent assembly language code or into another equivalent code suitable for interpretation (such as Java byte codes). Another characteristic example of metaprogramming is writing a script that generates code, allowing programmers to produce a larger amount of code and get more done in the same amount of time as they would take to write all the code manually. C++ templates and Common Lisp macros support metaprogramming by defining the contents of (part of) a program by writing code that generates it. Finally, reflection is also a valuable language feature for facilitating metaprogramming.

Recent developments suggest using refactoring as metaprogramming (Thomas, 2005). Originally, refactoring is the process of improving an existing program by transforming it into a new, better version. It is performed either manually, or using modern IDEs like Eclipse (http://www.eclipse.org), or other sophisticated tools that support navigation, querying, and visualisation of static and dynamic program structure and behaviour (e.g., Jquery, http://www.cs.ubc.ca/labs/spl/projects/jquery/). Metaprogramming by refactoring would also require a specific *refactoring language* to allow the developer to express current refactorings as well complex queries and program transformations, such as splitting and combining program fragments.

Typical values of the meta-meta depth factor in metaprogramming are 1 (as in the case of using compilers and interpreters) or 2 (e.g., when using tools like Yacc and Lex to generate compilers and interpreters).

6 Meta in software design

One of the best practices in software design is that of using design patterns as generalised, common solutions to recurring design problems. Some researchers and developers call design patterns meta-level design tools. Others believe that it is more appropriate to talk about patterns as generalised solutions, and pattern instances as domain- or application-specific instantiations of such generalised solutions.

Metapatterns are patterns describing other patterns (Volk, 1995). They are useful when designing a large framework, such as a GUI framework – a set of classes designed to work together in creating GUIs. Each framework contains one or more generic parts that are further configured and adapted by the user when applying the framework. Such parts are often called hot spots. Metapatterns are used to describe generalised design of such hot spots.

Technically, metapatterns are often represented using the notion of abstract (template) classes and methods. For example, the metapattern called Unification describes how to specify only the skeleton of an algorithm, leaving some details open – the trick is to use a template method that relies on one or more abstract (‘hook’) methods. Thus, some steps of the algorithm are deferred to implementation of the abstract methods in subclasses. The user is expected to use subclasses to override the hook methods to achieve the desired functionality.

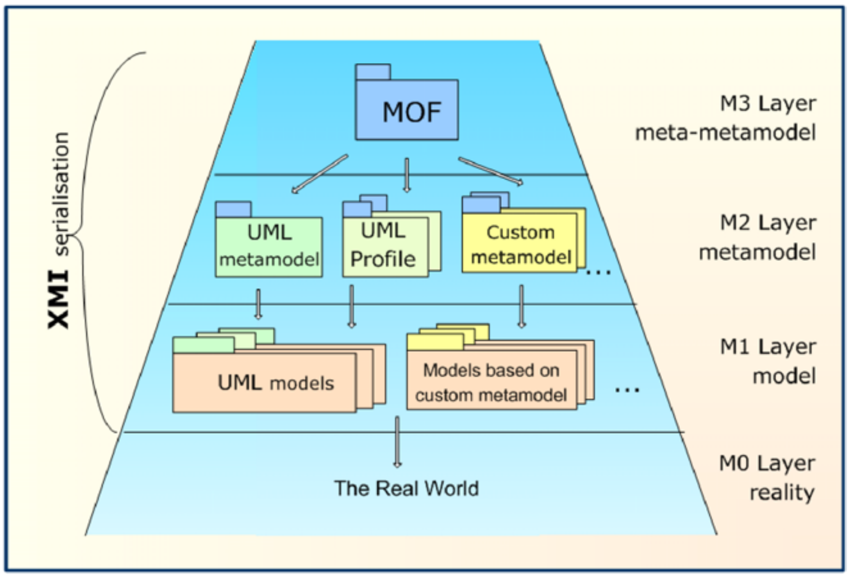
Depending on whether design patterns are considered meta-level design tools or not, the meta-meta depth factor here is 1 or 2.

7 Meta Object Facility (MOF)

Model-Driven Architecture (MDA) is an increasingly popular trend and standard-in-the-making in software engineering (Miller and Mukherji, 2003). It defines three viewpoints (levels of abstraction) from which a certain system can be analysed. Starting from a specific viewpoint, we can define the system representation (viewpoint model). The representations/models/viewpoints are *Computation-Independent Model* (CIM), *Platform-Independent Model* (PIM) and *Platform-Specific Model* (PSM).

MDA also defines a four-layer system-modelling architecture, Figure 1. At the M0 layer are things from the real world. The models of the real world are at the M1 layer (e.g., system models specified using UML notation). They are represented using the concepts defined in the corresponding metamodel at the M2 layer (e.g., UML metamodel). The best part here is the topmost, M3 layer – the meta-metamodel based on the *MOF* standard, which defines both the metamodels at the M2 layer and itself! Hence, the meta-meta depth factor here is strictly 2. In fact, MOF defines an abstract language and a framework for specifying, constructing and managing technology-neutral metamodels. It is the foundation for defining any modelling language, such as UML. MOF also defines a framework for implementing repositories that store metadata (e.g., models) described by metamodels. The purpose of the four layers with common meta-metamodel is to support multiple metamodels and models and their scaling – to enable their extensibility, integration and generic model and metamodel management.

Figure 1 MDA four-layer MOF-based metamodelling architecture (see online version for colours)

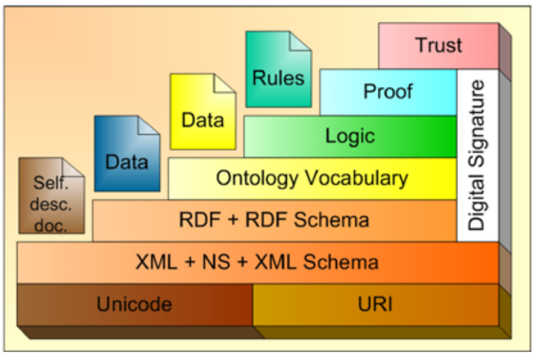


8 Meta on the Semantic Web

The Semantic Web (http://www.semanticweb.org) was originally conceived as a layer of machine-understandable metadata on top of the existing web. The idea is that by annotating existing documents and information on the web with some metadata, agents will be able to understand what the information is and what it is about. It is possible only if along with additional descriptions in the form of metadata, webpages and documents also contain links to a web of ontologies (Devedžić, 2002) that provide vocabularies and meanings of terms and concepts used in metadata. In spite of the fact that a number of useful ontologies are still lacking, the Semantic Web is a trendy topic in computing.

Both syntactically and semantically, there is a lot of meta involved with the Semantic Web. That fact is best understood having in mind the Semantic Web 'layer cake', Figure 2. Note that higher-level languages use the syntax and semantics of the lower levels. All Semantic Web languages use XML syntax; in fact, XML is a metalanguage for representing other Semantic Web languages. For example, XML Schema defines a class of XML documents using the XML syntax. RDF provides a framework for representing metadata about Web resources, and is XML-based as well. RDF Schema, OWL and other ontology languages also use the XML syntax.

Figure 2 Tim Berners-Lee's vision of the Semantic Web 'layer cake' (see online version for colours)



Semantically, a crucial concept related to the ontologies today is *Ontology Definition Metamodel (ODM)* (Đurić et al., 2005). ODM specifies all essential concepts used to define ontologies (e.g., Resource, Class, Property, etc.). Thus, ODM is essentially a meta-ontology that includes multiple ontology languages, and currently grows quickly to include mappings to different ontology languages like OWL, RDFS, etc. A similar MOF-compatible ontology metamodel is implemented in Protégé, a popular Semantic Web tool that supports import, export, and conversion from/to different ontology languages through plug-ins. Protégé can be even adapted to support a new ontology language by inserting new *metaclasses* and *metaslots* into a Protégé ontology. Hence Protégé is a meta-tool for ontology development and knowledge acquisition.

The meta-meta depth factor here is 1 both in terms of syntax (e.g., there is a concept of *meta-schema* – a schema that defines the validity of XML Schema definition documents) and in terms of semantics (ODM).

9 Software (meta)modelling and modelling spaces

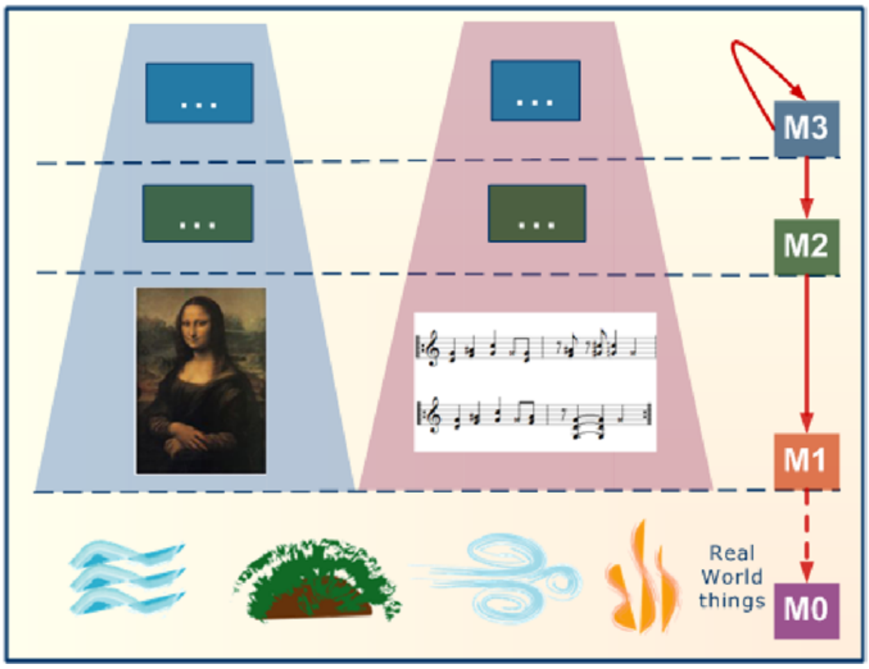
In its essence, software engineering is all about creating models. Some hard-core programmers would say: "Hey, that is not true – we use UML diagrams rarely, and even when we do, it is only because our managers forced us to". But, what else is our precious Java, C++ or even assembly code if not a model? A model is a simplified abstraction of reality. We build software to make a simplified snapshot of reality that we are interested in.

How do we define software models? By using metamodels, which define concepts that are the building blocks of models. These metamodels are defined using some other concepts that belong to meta-metamodel. We could continue this meta-meta layering but it is not necessary – meta-metamodel can be self-defined. MDA is a standard that clarifies these meta-meta relations using a well-structured, layered architecture.

MDA is primarily targeted at MOF-based and UML-related design approaches. However, it can be generalised to encompass not only object-oriented systems but also earlier, well-known modelling approaches. Using this generalisation, we could identify various *modelling spaces* depending on the meta-metamodel used in describing metamodels – MOF modelling space, EBNF modelling space, etc. Even models from everyday modelling spaces can fit in – art, literature, etc.

Figure 3 shows some common examples of models put in a layered architecture conceptually inspired by MDA – a famous painting (*Mona Lisa*, also known as *La Gioconda*, painted by Leonardo da Vinci in the 16th century), and a part of a written music score (of the song 'Smoke on the Water'). A noble Renaissance woman and a rock song are things from the real world, at the M0 layer. A painting and a music sheet are obviously abstractions of real-world things. Therefore, they are models and can be put at the M1 (model) layer. Metamodels are used to define these models. In the case of a music score, which is a written interpretation of a song, the metamodel (M2) is a set of concepts – stave, note, etc. – and rules that define what each note means and how we can arrange them on five staves. In this context, the meta-metamodel (M3) includes self-defined concepts that then define concepts: stave, note, etc. Although this architecture is imprecise and definitely not perfect from the perspective of music theory, at least it captures a formal interpretation of music.

Figure 3 A few common models put in an MDA-inspired layered architecture (see online version for colours)



Things get harder in the case of painting. Is it possible to specify a metamodel that can define such a complex and ambiguous model as a masterpiece painting? A simplistic view, based on physical appearance only, may lead to the definition of this metamodel in terms of concepts like line, circle, colour, etc. The meta-metamodel would then be a set of concepts used to define line, circle, colour and their meanings. However, *Mona Lisa*, like any other artwork, is much more than just lines and colours. It has much to do with human psychology and experience and can be interpreted in many ways. It is much more difficult, if not impossible, to define a formal metamodel or meta-metamodel in this case. We may anticipate that they exist, but they are extremely complex and implicit.

Another important issue is that, although *Mona Lisa* or written notes are models, they are also things from the real world. We can hold them in our hands (if the guards in Louvre let us do this, in the case of *Mona Lisa*!) and they can be items entered in the information system that stores information about art.

The previous analysis shows us that something can be taken as a model if it is an abstraction of things from the real world, but it is simultaneously a thing from the real world. Whether we take it as a model or as a real-world thing depends on the context, i.e., on the point of view. Also, models can be defined using metamodelling concepts formally or implicitly.

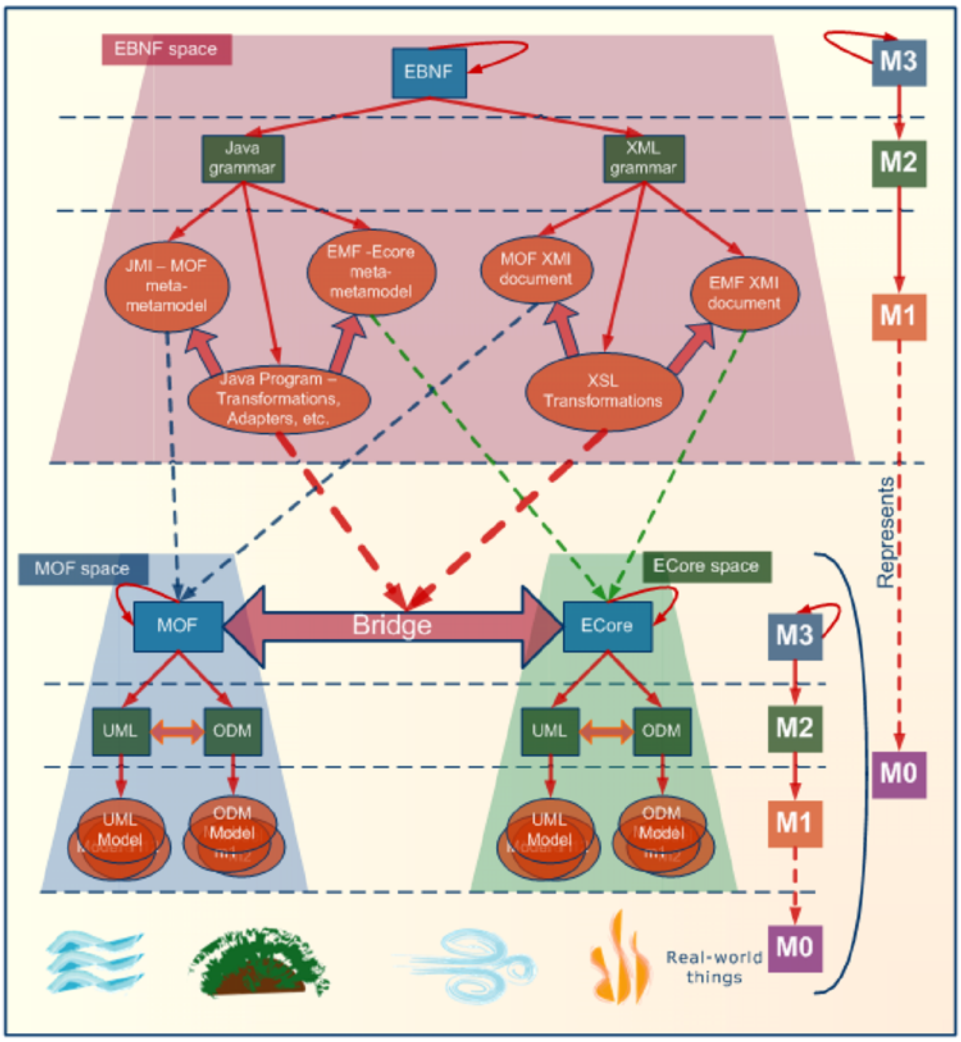
Any software is primarily a model of the real world, but in some phases of its development, it is also treated as the thing from the real world. That often leads to confusion with UML and programming language code – both are models, but at same stages of development UML models are used to represent code as a thing from the real world.

10 Meta to meta …

An XSLT stylesheet is an XML document defining a transformation from one class of XML documents into another, thus specifying a meta-to-meta transformation at the syntactic level. A much more comprehensive approach is enabling transformation from one modelling space to another. Software engineers and developers often need to analyse and describe their artefacts and the domains their software models from different perspectives. Figure 4 illustrates this idea using the MDA four-layer system-modelling architecture. It is possible to model the same set of real-world things in two different modelling spaces, but in another way. In this case, the relation between these modelling spaces is oriented towards pure transformation, bridging from one space to another. Examples of such parallel modelling spaces are MOF and ECore (ECore is a meta-metamodel very similar to but also different from MOF; although ECore is a little simpler than MOF, they are both based on similar object-oriented concepts). In such cases, things like XSLT and Java programs can be used to perform the actual transformations.

It is also possible to take models from one modelling space as real-world things to be modelled and represented in another space. Both MOF and ECore spaces are represented in other, more concrete modelling spaces. For example, they can be implemented using repositories, serialised into XMI, etc. Figure 3 shows them as Java program codes and XML documents in the EBNF space. Models from the MOF space are modelled in Java code according to JMI standard, and in XML according to the MOF XMI.

Figure 4 Transformations between modelling spaces (see online version for colours)



11 … and beyond

There are numerous other advanced domains of computing in which meta-level computational tasks are well developed and largely applied. For example, *metaheuristics* is a top-level general strategy, which guides other heuristics to search for feasible solutions in domains where the task is hard (e.g., Tabu Search, simulated annealing, and genetic algorithms). In dialogue modelling and analysis, *meta-statements* are turns that are about the discussion rather than part of the discussion pertaining to the topic – "What does John mean?" is a meta-statement about John's statement.

A well-known researcher in the field of metadata related to learning technology systems commented recently: "Metadata … I hate that word. We can use the word *description* instead". Still, the prefix *meta-* used in so many words in computing should not be confusing and repelling at all if we reflect a little upon it. On the contrary, the *meta* opens a range of new ideas and perspectives if we only give it a little *metacognition* effort – thinking about our own thoughts about the *meta*, reflecting, exploring and linking, and basically understanding our own thoughts.

References

Allan, R.J. (2000) *Parallel Application Software on High Performance Computers – Survey of Computational Grid, Meta-computing and Network Information Tools*, January, Online Available: http://www.dl.ac.uk/TCSC/Subjects/Parallel_Algorithms/steer_survey/.

Devedžić, V. (2002) 'Understanding ontological engineering', *Communications of the ACM*, Vol. 45, No. 4, April, pp.136–144.

Đurić, D., Gašević, D. and Devedžić, V. (2005) 'Ontology modeling and MDA', *Journal of Object Technology*, Vol. 4, No. 1, January–February, pp.109–128.

Foster, I. (2003) 'The grid: computing without bounds', *Scientific American*, Vol. 288, No. 4, April, pp.78–85.

Ip, M.C., Morrison, I. and Mason, J. (2005) *Metasearching or Megasearching: Toward a Data Model for Distributed Resource Discovery*, January, Online Available: http://www3.dls.au.com/metadata/DataModel.html.

Miller, J. and Mukerji, J. (Eds.) (2005) *MDA Guide Version 1.0.1*, January, Online Available: www.omg.org/docs/omg/03-06-01.pdf.

Thomas, D. (2005) 'Refactoring as meta programming?', *Journal of Object Technology*, Vol. 4, No. 1, January–February 2005, pp.7–11.

Volk, T. (1995) *Metapatterns*, Columbia University Press, NY.