Advances and Applications in Model-Driven Engineering

Vicente García Díaz  
*University of Oviedo, Spain*

Juan Manuel Cueva Lovelle  
*University of Oviedo, Spain*

B. Cristina Pelayo García-Bustelo  
*University of Oviedo, Spain*

Oscar Sanjuán Martinez  
*University of Carlos III, Spain*

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series
Chapter 3

Model-Driven Applications:
Using Model-Driven Mechanism to Bridge the Gap between Business and IT

Tong-Ying Yu
Independent Consultant, China

ABSTRACT

How to bridge the gap between business and Information Technology (IT) has always been a critical issue for both the developers and IT managers. The individualized, differentiated demands by different customers and situations, the constantly changing in both business and IT are great challenges to the applications for enterprises. In this chapter, the authors respectively discuss the left side (computer) in software engineering, with Object-Orientation (OO), Model-Driven Engineering (MDE), Domain-Driven Development (DDD), Agile, etc., and the right side (the business) in Enterprise Engineering (EE) with Enterprise Modeling (EM), and Enterprise Architecture (EA) of the gap. It is shown there are some fundamental problems, such as the transforming barrier between analysis and design model, the entanglement of business change and development process, and the limitation to the enterprise engineering approaches such as EA by IT. Our solution is concentrated on the middle, the inevitable model as a mediator between human, computer, and the real world. The authors introduce Model-Driven Application (MDApp), which is based on Model-Driven Mechanism (MDM), operated on the evolutionary model of the target thing at runtime; it is able to largely avoid the transforming barrier and remove the entanglement. Thus, the architecture for Enterprise Model Driven Application (EMDA) is emerged, which is able to strongly support EE and adapts to the business changing at runtime.

1. INTRODUCTION

The context of this chapter is the applications of Information Technology (IT), mainly, the software which supports the comprehensive operations (business and management) of an enterprise, i.e., the “applications for enterprises” or “enterprise (business) applications”; another conventional name is Information System (IS). For many software developers, “enterprise applications” means a series of technical characteristics, but our focus on the demand, the applying targets and objec-

DOI: 10.4018/978-1-4666-4494-6.ch003
The environment, as well as the openness and collaboration between computers and humans, in particular the essential functional requirements including the adaptation to business changing.

IT and the applying enterprise/business are mutually independent fields, which have its own ecosystem, though IT should always services business in our context. It is not surprising that, there is an innate gap between them, which can be appeared in anywhere the IT needs to be aligned with the business; this is the basic issue we must face. Empirically, the biggest challenges to enterprise applications come from the differences and changes in the demands, also the running environment and the IT itself. First, there are some conflicts between the different needs from different customers, different situations, and developer’s pursuit to generic a solution; the latter is also a factor what makes the product to be bloated and complicated. Second, the uncertainty, constant and rapid changing of business requires the application system has higher flexibility and capability to evolve in runtime; this is a huge challenge to the existing software techniques. Third, IT itself is one of the most complex and rapid changing fields, which leads to an increasing requirements to the maintenance, governance, and the adaptation to the operating environment. Furthermore, in such the applying context, the problems that require solving by the application system is open; this will make a significant impact on the development and the whole life cycle of the application. Moreover, between the business and IT camps, and also the different roles in various phases, there are always the problems to the communication and understanding with the different professionals and standpoints; technically, this requires some common languages or media to represent the issues of business and IT they are faced. In addition, it seems we are already in the age of total computer information processing, that is, almost all data and their derivatives (e.g. forms or documents) should be stored, processed by a computer-based way; papers are only the complementary media in some special circumstances. However, have a review on those problems, it can be seen, such the information islands and the functional silos, which proposed about half a century ago, in our experience, are still not a substantive change, even more serious. Above challenges and issues, with such the cultural and regional differences, and so on, makes that bridging the business-IT gap, establishing strategic alignment or fusion of them, to be still the hard and most important subject for the industry. We believe, there are still more essential approaches that waiting for discovery and validation.

In this chapter, we focus on the role of models for the problem-solving by human with computer, and targeted at the relationship between a certain computer system and the outside world—the people and physical or abstract entities, their states, activities and events—especially in dynamic way at runtime. We start to demonstrate at the inevitable way to connect human, computer, and the real world: models and modeling. And then, around them, have some review on the two side of the gap (see Figure 1), respectively in software engineering and enterprise engineering. Based on the discussions, the Model-Driven Application (MDApp) is introduced, which is based on the principle we called Model-Driven Mechanism (MDM); further, the architecture of Enterprise Model Driven Application (EMDA) is presented.

2. TWO SIDES OF THE GAP AND INEVITABLE MODEL

2.1. Model as Mediator between Computer, Real World, and Human

No matter what challenges the business-IT gap bring us, there is only one core issue, that is, how to utilize computer to solve the real world problem we faced. Regardless of the means, to solve any problem, we must first know what fact there—by model. This principle is not only suit-
able for human, and is also suitable for computer. Smith (1985) depicted the relationship between computer and the real world and showed us a basic view for the relationship between computers (generally may be a program, computer system, even a description, thought) and real world; he put the computers in the left box and the real world in the right box, where the “inevitable model” is placed as mediator between them. His picture also implied the same relationship between human and real world; for our subject, however, all the three relationships are important, so we depict the basic landscape as Figure 1, wherein the lower half layout is logically equivalent to the Figure 1 in Smith (1985, p. 21), and we labeled the relationships.

Smith implied that, the gap between computers and the real world is able and only able to be connected by model of the real world thing, as well as between human and the world. This view is a background of the exploration to business-IT gap: the IT domain on the left side; the business domain on the right side, and some model as mediator to be connected them.

Another important point shown on Figure 1 is that, under our subject, the question is not only between computer and the world but with human: a computer handles something based on the mediating model setting by human, thus the model is served as a mediator for both the human and computer.

In addition, the “real world” is a relative concept; what thing is in the real world depends on your standpoint: e.g., for computer, or human? When a human takes a computer to handle some affair, it can be regarded that there is something in real word for both. In this case, why we cannot use the same model to represent the real word thing, if it is possible, for both the computer and human? Fortunately, it is not just possible but the best way.

Relevant to our subject, based on Figure 1, the efforts to bridge the business-IT gap can be classified into three groups: on the left side such as (mainly) software engineering, on the right side as enterprise engineering (see Sec. 4), and on the middle for bridge over the gap—the solution based on the mediating model.

2.2. Domains and Models for Enterprise Applications

The term “domain” and “domain model” is overloaded in software field; there are many different definitions (Larman, 2004, Sec. 9.2). Here, for effective discussion, we try to give a set of relatively consistent descriptions for some concepts of domain and model involved in the subject of this
chapter, as consistent as possible to the common usage. The word “domain” is used against “problem” or “subject,” i.e., it is referred to the scope, in which the problem or subject are occurred or involved, and more specifically, it is referred to the set of objects or entities with properties and basic relationships. A point is that the different problems or affairs may be occurred and repeatedly in the same domain; on other hand, the boundary of a domain is usually relative. Accordingly, the models are appeared as two classes: the first class is modeling a target or subject, e.g., a system, a problem, a business process, etc.; the second class is modeling a domain, i.e., “domain model” in general, e.g., the domain of an problem, the domain of an application or system. They reflect two different perspectives, though there may be no need to establish a domain model for such a single, certain problem. To some open problems or subject, however, the relative independent, predetermined domain model is necessary and important. Such the domain model is quite close to the concept of ontology.

The total information processing needs total information modeling; as mentioned above, under our subject, what problems faced by an enterprise application are essentially in changing and uncertain—it is open. The inevitable mediating model thereby should satisfy three conditions: human-understandable / operational, computer-treatable, and evolutionary with the changing of the target as well. Thus, a predetermined domain model, namely enterprise model, is necessary for enterprise applications. Figure 2 shows some typical domains and their relationships for an enterprise application, in which the labels are omitted “the domain of,” e.g. “the domain of a problem.” The dotted line indicates the border is relatively uncertain.

3. SOFTWARE ENGINEERING ON THE LEFT

3.1. Difficulties and Transforming Barrier from Analysis to Design

Kaindl (1999) demonstrated the difficulties in the transition from object-oriented requirement analysis (OOA) to the system design (OOD). He focused on two problems that are particularly difficult in OOD: first, architectures: “Finding an architecture that fits well to solve a given problem is both difficult and important.” Second, model of the domain inside the program: “the OOA model cannot be simply part of the OOD model” (pp.
This is involved in the design decisions about which information about real world objects the deployed system will store and how it will be represented, i.e., how to model the thing in domain inside the program. Such the opinion is appeared as a logical succession from the opinion by Bosh and Molin (1997): “the most complex activity during application development is the transformation of a requirement specification into an application architecture.”

In Kaindl (1999), the OOA model roughly is a problem model in which there are both the elements of computer and other things involved in the problem. From a modeling perspective, the problem is the transformation from the analysis models to a design model. Model transformation is the major strategy to improve the system development in MDE, but it does not effectively address the problem even often avoided the issue (see the discussions about MDE and CIM later). Karow et al. (2006) presented that “the transformation results are potentially insufficient and inappropriate for the design process.” And “an automation of the design starting from the CIM is not possible by utilising currently existent transformation approaches.”

Flowing above accounts, we can describe the problem more generally, that is, in general, there is no simple (liner or automatic) complete transformation from analysis model to design model or code. It can be called transforming barrier between analysis model and design model. The analysis model here can be involved in problem model, domain model, business (process) model (as is; to be), and may also the functional model (as a black-box model of the system, e.g. the use case model) of the application system. Against, the design model is somewhat white-box model of the system which are can be implemented in some executable code.

### 3.2. Object-Orientation and Use-Case Driven

Some one might think that the scenario described in sec. 2.1 is very consistent with the Object-Oriented (OO) approach: Classes / Objects are the models of the real world thing; this relationship is shown in most the examples in OO books or articles. There was a cautious statement in Craig (2004, Sec. 9.3): “This is a key idea in OO: Use software class names in the domain layer inspired from names in the domain model, with objects having domain-familiar information and responsibilities.” However, this issue is indeed relating to the transforming barrier, “The question remains how OOD objects can be both abstractions of something in a problem domain and objects in a solution space” (Kaindl, 1999, p. 95). The direct correspondence between the Classes / Objects in software and the objects in real word is helpful but not enough to solve the transition difficulties. We prefer regard an Object as a container to the model of the real object (if there is—an Object in software can also be not corresponding to any real world thing), i.e., the data of a model is encapsulated in an Object as its private data; in other words, the model is hard-coded, embedded in the code.

Moreover, much OO development methods with UML employ the use-case model as a basis, for example, the unified software development process (UP) directly captures the required functions by the use-case driven, iterative and incremental approach, where domain model and business model are unnecessary but only used as the context to assist in understanding the functionality that captured by use-case model (Jacobson, Booch & Rumbaugh, 1999). The use-case driven, iterative and incremental process is typically a nonlinear process, which is an evidence of the transforming barrier.
3.3. Model-Driven Engineering and Abstract Hierarchy

Model-Driven Engineering (MDE) was regarded as an “important paradigm shift is happening in the field of software engineering” (Bézivin, 2004, p. 21), which might represented “the first true generational shift in basic programming technology since the introduction of compilers” (Selic, 2003, p. 20). MDE extends models and modeling to almost all concerns relevant to the development, which consist of a multi-dimensions modeling space, and lies engineering on their mapping and transformation (Kent, 2002). “Considering models as first class entities and any software artifact as a model or as a model element is one of the basic principles of MDE” (Bézivin, 2006). Although there are a lot of different ways to the research and practice, MDE is mainly developed along some basic principles that outlined by the Model-Driven Architecture (MDA) of Object Management Group (OMG), including: raising the level of abstraction, transforming the model to lower level, code automated or partially automated, and so forth. As discussed in Frankel (2003), it is mainly for filling the gap for in software architecture, e.g., the gap between programming language and machine code, or development environment and execution platform—but not yet the gap between computer and the real world.

In last over ten years, represented by MDA, the field had become a hotspot in software engineering, and attracted much research and development. However, so far, compared with the expectations, its development does not appear to be satisfactory. In the COOMP workshop 2011, Jean Bézivin presented the views that “MDE has reached a standstill” (Dubray, 2011), and even used the word “failure.” It seems to be disappointed, at least, for MDA (of course there is no more influential instance than MDA)—this is appeared somewhat the common sense in MDE community. This situation is somewhat familiar to what we had been seen, e.g., the Computer-Aided Software Engineering (CASE) in 1980s: it “attracted considerable attention in the research and trade literature, it wasn’t widely adopted in practice” (Schmidt, 2006, p. 25), as well as the case to the fourth-generation programming language (4GL). Related to CASE, however, France and Rumpe (2007) pointed out “MDE can and should be viewed as an evolution of early CASE work” (Sec. 3.3). There may be a lot of reasons leading MDE/MDA to be disappointing, one of them may be related to the issue mentioned above, i.e., existing MDE is not directly touch on the difficulties and the transforming barrier.

Furthermore, we can discuss the problem on the concept of abstraction. CASE, 4GL and MDA are all the efforts that somehow raise the level of abstraction for developer but it appears they are a bit less successful than 3GL or the earlier efforts. One of the reasons might be related to the ultimate of raising abstract level. First, there are two abstract hierarchies on the two sides of the gap, one is on the computer operations/constructs, and another is on the real word things. Second, the distinction of the two abstract hierarchies determines the target and the ultimate of the division and degree for abstraction in system development. We argue that there is a most important principle, i.e., it should not only unilaterally raise an abstraction level but matching them: to establish appropriate link mechanism between the two abstract hierarchies. We can see, however, either the existence and distinction or the link mechanism of the two abstract hierarchies are often to be confused and ignored.

France and Rumpe (2007) had proposed a conclusion that MDE is more likely to contribute to software development rather than the inherent software complexity; it seems that is in line with the fact we have seen (they also encouraged research on the use of runtime models, see discussion in sec. 5.5). As shown in many literatures, e.g., Miller & Mukerji, 2003; France & Rumpe, 2007), both MDA / MDD or broader MDE / MDSD are all around to the improvements for software development process, that is, on the left of the gap, and in software productivity and quality.
3.4. Some Issues with the Computation Independent Model

We pay more attention to the CIM in MDA, which is put at the top of the abstract levels. First, there are some ambiguous to what is a CIM. In MDA Guide 1.0.1 (Miller & Mukerji, 2003) defined that “A CIM is a model of a system that shows the system in the environment in which it will operate” (this distinctly refers to the black-box model of the application system, i.e., a system model, the first class model we identified in previous, at sec. 2.2); but in the same document, it also stated that “CIM describing the situation in which the system will be used. Such a model is sometimes called a domain model or a business model.” (i.e., as the second class model) This issue is similar to the ambiguous with so-called analysis model, may be seen as it is a general designation of some different types of models; the key question is, however, which one is the object being modeled by a CIM? Is it on the left or right of the gap? If we regard CIM as analysis model, then the transformation from CIM to PIM will be faced the transformation barrier we discussed above; otherwise, it in fact will be overlapped with the concept of PIM.

Second, In fact, it appears that the subject about CIM, including the transformation from CIM to PIM, did not get the enough attention; MDA transformations are only focused on the PIM and PSM (Fouad et al., 2011). The similar views can be seen from more documents (e.g., Kardoš & Drozdová, 2010; Sharifi, Mohsenza-deh, & Hashem, 2010). Nowadays, the situation is still as stated in Karow et al. (2006): “CIM has not achieved much attention so far. […] there is no model type in MDA covering the requirements for the software system in development, which is the actual source model on the transition from analysis to design.” And, CIM “is disregarded by most MDA methodologies and tools, which start their transformation process on the level of platform independent software models (PIM).” In our views, the most efforts of MDE are limited on the left side but not virtually across over the gap; it in fact a little avoid the essential issue, the analysis and design model transformation barrier; the ambiguous and disregard to CIM may be related to some key factors which cause the MDE/MDA to “miss the boat.”

3.5. Domain-Driven Design

Nowadays, many architectural approaches or designs take the domain / business model to a more important position, for example, the Domain-Driven Design (DDD). This is a good example that tried to across the transforming barrier, as stated by Evans (2003): DDD “discards the dichotomy of analysis model and design to search out a single model that serves both purposes” (ch. 3); notice that the model is not use case model or system model but domain model. On the other hand, DDD emphasizes to bind model and implementa-
Model-Driven Applications

ing, design (both the jobs and the roles), this will lead to more serious entanglement of business changing and development process (see next section). From software engineering view, that is not a progress but somehow a regress.

DDD is not current MDE, in comparison, it is using programming language as modeling language rather than the opposite and the driven model is domain model but not the system model (this is close to MDApp, see later). On our perspective, by embedding domain model in an isolate layer, DDD has taken a small step to cross the gap – as a step forward toward our approach, the MDApp.

3.6. Agile and Entanglement of the Business Change and Development Process

The primary intention of Agile software development is to adapt to the “changing requirements, even late in development” (Beck et al., 2001). In addition to the part of the attitude, however, its methodological basis is still the “iterative and incremental” that the practices can be found from 1950s (Larman & Basili, 2003). However, the emergence and continued popularity of Agile methods in fact show another problem up: which can be described as entanglement of the business change and software development process. In fact, faced the constant and rapid changing business, a faithful implementation to the Agile principle will inevitably lead to a permanent close cooperation between the developers and business people; on the other hand, the different enterprises’ different demands inevitably require them make cooperation across different enterprises (i.e. the system developer and the client). To permanently keep such a close collaborative relationship, such development process is only suitable for customized development with a long-term contract, or within the same enterprise. Such passive bundling becomes always heavy shackles on both the client and the developer.

4. ENTERPRISE ENGINEERING ON THE RIGHT

4.1. What is Enterprise Engineering

Enterprise Engineering (EE) was defined as “a body of knowledge, principles, and practices having to do with the analysis, design, implementation, and operation of the enterprise” by SEE in 1995 (Liles, Johnson, & Meade, 1996). For about half a century, in the context of IT applications, there was emergence of many of ideas or approaches for the enterprises towards information age, such as CIM (Computer Integrated Manufacturing) or enterprise integration, Enterprise Modeling (EM), business-IT strategic alignment, Business Process Reengineering (BPR), Enterprise Architecture (EA), Business Process Management (BPM), and so on. Although they respectively have different origin, all have some background of IT application, and to regard an enterprise as complex system and take engineering methods into the whole life-cycle of enterprise, such as the planning, design, construction, governance, and transition, and so on. Therefore, they can be naturally gathered under the general concept of EE (Liles et al., 1996; Yu, 2002b; Cuenca, Boza, & Ortiz, 2010). Naturally, a common feature of such the approaches is the combination of business and IT/IS application. That is what the keynote of the early monograph by James Martin: using EE to align people, IT, and strategy (Martin, 1995).

4.2. Enterprise Modeling

As the necessary foundation for a mature engineering discipline, Enterprise Modeling (EM) is naturally the “major element in Enterprise Engineering” (Liles & Presley, 1996). The conventional EM can be traced back to 1960s-1970s, it is originated from the fields such as the industrial automation, Computer Integrated Manufacturing (CIM), Information Systems (IS), and Enterprise Integration (EI), etc. To 1980s, it has produced
rich achievements, including much different EM methods or tools, as well as enterprise modeling languages. Although they are essentially developed as a branch of the IT application, they do not seem to get too much direct use in the general application development. We believe that this is related to the issue of the architecture style of application system so far, as well as all the issues in software engineering discussed previously.

In any case, for our goal—building enterprise application system on evolutionary enterprise models—EM will become one of the most important bases, in the meantime, the new applications will also become the most important use of EM but not just for integration or communication and understanding. Fox and Gruninger (1998) defined that “An enterprise model is a computational representation of the structure, activities, processes, information, resources, people, behavior, goals, and constraints of a business, government, or other enterprise.” It clearly implied the duality of the model for computer and human; but it seems those refined work has some distance to the general business applications. Our objective, however, is first to achieve an comprehensive information system which is able to be constantly evolving, to support the day-to-day business and management activities in an open, fast changing environment, not the senior activities like as deducing on model.

4.3. Enterprise Architecture

In contrast with EM, the rise of Enterprise Architecture (EA) is a bit later, but the development trends obviously over the former, has become one of the most active fields in EE. It appears that the benefits of EA increasingly being discovered and summarized and it is gained more and more recognition and adoption from the practitioners such as the CIOs. (Tamm, Seddon, & Shanks, et al., 2011).

As a practice-oriented young approach, there are much different understandings about EA; TOGAF (The Open Group Architecture Frame-work) is relatively representative one. According to TOGAF 9 (2011), EA is divided into four architecture domains: business, data, application, and technology, in which the latter two are in fact belonged to IT field. That is, although the name has changed, from early “IT architecture” or “IT planning” etc. becomes current “enterprise architecture,” it is actually still a strategic tools to IT application. We can see that (e.g., Baudoin, Covnot & Kumar, et al., 2010), how to build a strong and balance alignment between business and IT, from the top strategic level to the bottom IT infrastructures, are always the major concerns of EA researchers and practitioners.

In addition, EM is the necessary basic tool which be contained in EA, and EA also be used by EE (Cuenca et al., 2010), the problems encountered in EM will still exist within EA.

The EA approach, however, is not able to change the inflexible software, cannot break the gridlock from the entanglement of the business change and development process. Thus, each application silo or island becomes some indecomposable “granularity” in EA implementation. The application architecture thereby becomes a very macro planning on the big granularity of the silos or islands. The data architecture is also only used for integration, as well as the enterprise modeling. Such the problems largely limit the role of EA, and lead to that EA seems only useful to huge enough enterprise with a highly complicated IT environment. For example, the findings in Tamm, et al. (2011)

5. MODEL DRIVEN APPLICATION AS BRIDGE

5.1. Using Enterprise Model to Drive the Application

In order to adapt to different customers and situations, much generic or semi-generic application software provides a lot of optional parameters to
Model-Driven Applications

reach a certain extent of customization. Such the parameters are, in fact, described or corresponding to some properties of the enterprise—it is actually some fragments of a complete enterprise model, though it seems usually too simple, from a modeling point of view. This fact prompted us the way which can be used to handle a more complete enterprise model, thereby, to make the user to be able to control or change the function or behavior of an application through modify the models, to adapt to the changing of business and other elements of an enterprise. Based on such thoughts and some attempts in actual application system development, we suggested the idea for a new generation of enterprise application (information system) based on enterprise-model-driven (Yu, 1999). This idea has attracted the attention of some local people in the industry; some successful commercial products appeared in a few years; and some industry researchers classified it as business system infrastructure platforms (ChinaLibs, 2002).

5.2. Model-Driven Mechanism

Based on the initial idea, we further found that, the approach that build a system on some evolutionary model is according to certain general principles, we called as Model-Driven Mechanism (MDM): it makes all or part of functions and behaviors (or the structure and form) of a system to be in control of or mastered by model (Yu, 2005, 2007). MDM is shown as three relative parts in a system (see Figure 3):

The model (or applied model for emphasize) models the target thing related to the functionality or behavior of the system, which is conformed to the modeling knowledge.

The modeling knowledge including the specifications, rules, languages, notations or / and formats, etc. to the applied model; it is roughly the metamodel of the applied model (see the discussion later).

The operational device accesses the applied model and does some actions (functions or behaviors) on / for the target thing being modeled by the applied model, which is according to the modeling knowledge.

As shown on Figure 3, the three parts of MDM consists of a closed triangle relationships, this is related but different to the metamodeling architecture well recognized in MDE which illustrated in e.g. Atkinson and Kühne (2003), Bézivin (2004, especially the Figure 5: The 3+1 MDA Organisation). The modeling knowledge is the key to determine the role or effect of MDM.

Figure 3. Model-driven mechanism (MDM)
It may include any type of metamodels of / for the applied model. Here, we emphasis on the relation to existing work of MDE, so, in the rest of the chapter we will principally use the term “metamodel” in a relative wide sense, for example, it may be referred to logical data model (scheme), conceptual model such as E-R model, or as the “ontological commitment” (Kurtev, 2007), and so on. According to the needs of a system, the different metamodels (modeling knowledge) may be used either individually or multiply.

Moreover, there are some basic characteristics around the applied model of MDM (Yu, 2002a, 2005, 2007):

- **The Independence of Applied Model**: It is independent of media and the system, and can be modified, reproduced, transmitted or transferred in some general, system-independent way.

- **The Timeliness of Applied Model**: It works at runtime within the system.

- **The Evolvability of Applied Model**: It can be continuous changing when they are working in a system at runtime.

- **The Program-Model Decoupling**: From a programming perspective, the operational device accesses the data of applied model at just runtime; it *never* references them in the programs / code when programming but only the types or the variable name specified in the metamodel.

- **The Metamodel in System**: In contrast to the applied model, the metamodel will be embedded in the system, e.g., to be hard-coded in the program in a single MDM structure, or accessed from a repository in runtime as in such a multiple MDM structure (see latter).

MDM has very universality in many kinds of advanced systems, such as computer / software-intensive systems, intelligent systems, etc. It is a fundamental answer to the basic question about models: how does a model work? More discussion about MDM will be presented in a specific paper.

### 5.3. Model-Driven Applications

From the standpoint of software, the concept of MDM is very close to the topic of applications. First, We defined *Model-Driven System* (MDS) as a class of systems that the main functions and behaviors are achieved on MDM, namely if all the functions and behaviors of a system can be defined, controlled and changed through MDM at runtime, then the system is a full model-driven system (Yu, 2005, 2007). Further, it is presented that a *Model-Driven Application* (MDApp for short) is a model-driven system, in which the relevant things (the target objects or affairs, such as the business) to the application system are modeled by the applied models that the functions or behaviors of the system will be operated / based on, or in control of, these models. In other words, MDApp is the system which is run on some evolutionary models; one can achieve some operations for some target thing via add the model of the thing that conforms to the metamodel. For an MDApp, can say that *what can be done is what can be modeled*.

MDM is the basic structural feature of MDApp. As a software-intensive system, the device may appropriate to be divided into some modules; a basic structure of MDApp is shown as Figure 4.

In Figure 4, the operational device in MDM is separated as three typical modules: the Modeler provide the function to modeling the applied model; the Model Driver (somebody may be used to call it engine) access the data of the applied model and provide the basic operations (e.g., via APIs, services or compiler/interpreter to some language) for it; and the Operator to achieve the functions of the system. For instance, an autopilot system as Figure 5; in this case the operator acts on the real target thing directly.
In addition, as we mentioned above, MDM also the fundamental of MDE / MDA; the author believe that this is a useful complement to the current theory of MDE / MDSD. Figure 6 shows that a typical code-generation system is an application of MDM (as an example, it is to be simplified, which shows only one existence of MDM in the basic structure).

Database and the applications are important existing instances of MDApp, as well as some BPMS which support run on the business process model directly. Figure 7 illustrates the database application which the functions are on the data, that is, a simplified architecture of database environment with MDM. The concepts in the figure are referenced to Elmasri & Navathe (2003, in the Figure 2.3).

Furthermore, Figure 7 also shows a very important aspect of MDApp, i.e., the multiple structure of MDM overlapped: as in the case of database architecture, the stored data are the models of real world which are conforms to some schemes, and they are evolvable that supported by the DB Processor; the schemes (catalog) are models of the data and it is also evolvable that supported by the DB Manager—both are can be updated by some users on the Database Management System (DBMS). Hence, the term “applied

---

**Figure 4. A basic structure of model-driven application (MDApp)**

**Figure 5. Autopilot system based on MDM**
model” for MDApp will not only be limited to the model in a certain MDM structure but can also be metamodel in another MDM structure. The applied model for MDApp is always user-changeable and has its metamodel (perhaps a meta-metamodel in a multiple MDM structure) and the corresponding operational device (the model driver). In addition, the division of metamodel and model may be relative: it indeed depends on the device—how to program for the metamodel, which is the most important job for the software of an MDApp.

Especially, MDM can also be used for the user interface, report format, system deployment and configuration, driving the peripherals or equipments which are connected to computer directly, and so on; some of the targets are probably a part of the system itself, which are typically not been regarded as so-called real world. To apply MDM on both the model of outside target and model of own target is indeed a very important point for MDApp.

Based on MDM, an MDApp makes the computer operations (functions) to be relied on the models which are resided in the computer and represented the target things (e.g., business) on the right side of the gap; such the models can be created and updated—evolutionary—by human (e.g., business analyst) in the runtime of the application system. What business can be handled

---

**Figure 6. MDM in MDA style code generation system**

**Figure 7. A simplified architecture of database environment with MDM**
on the MDApp are what can be modeled by the business analyst—according to the metamodels; a well-designed platform may be capable to add new model drivers to support certain new business (models). In addition, the applied models can be classified into different groups strictly depend on the metamodel and be processed by the special model driver as well as the modeling interface / tool. This is the basis to support the separation of the concerns by different stakeholders.

5.4. Enterprise Model Driven Application

On the illustration of MDApp, it is clear that how to build an application on the enterprise model, which can be called Enterprise Model Driven Applications (EMDAs). A primary structure for EMDA is illustrated on Figure 8, in which the applied model will mainly include enterprise model. In an actual system, the primary (MDM) structure may be repeatedly used for different model-metamodel pairs; the relationships between them are dependent on the relationships between the models. Around these core construct, the system can still be constructed with different architectural styles, e.g., SOA.

As shown in Figure 8, ignore some conventional part for the similar system, the special parts of an EMDA system are including (Yu, 2005):

- **EMDA Platform**: A software platform, the fundamental functionality is to provide the access mechanism to all the models (data), the enabling, governance and configuration mechanism to the model drives, functional engines and tools, and to the operational interfaces.

*Figure 8. The primary structure of enterprise model driven applications*
Model-Driven Applications

- **Applied Models**: The first class of applied models is the enterprise models, which may comprise of such as business (process) model, organizational model, resource model, and so on. The organization and modeling of enterprise models will be one of most important, special subjects of the development of EMDA—in principle, it is only depended on the metamodels (modeling knowledge) which are supported by the model drivers, without the knowledge about programming or the principles to the work of software; we can inherit the rich heritage of the existing enterprise modeling research. Other applied models may include UI model, report model, configuration model, etc. A practicable solution may have a multi structure of MDM; hence, an applied model may be also a metamodel of other applied model.

- **Metamodels**: Each model has its own metamodel(s). The storage way to a metamodel is depended on the MDM structure and the driver. Whatever, the metamodels are the shared specifications for the modeling of its applied models and the design of its model drivers, functional engines and modeling tools.

- **Model Drivers**: A model driver achieves the basic operations for certain model according to its metamodel(s). It is the key artifact that determines what the system can do—with the applied model.

- **Functional Engines**: A functional engine is a loadable module which is running over the different model drivers to provide special function(s), e.g., to does the operations on / for the target things.

- **Interactive Interface Layer**: Including / supporting the interface to human, equipment, or other computer systems. As mentioned above, we can also apply MDM for this via modeling the interface, e.g., the human-computer interface. SOA is an obvious way to the implement of this layer.

- **Enterprise Engineering Tools**: The fundamental function of EE tools is modeling. On the basis, it can be extended to add variety of advanced features, such as more advanced supporting to enterprise/business design for the needs from real enterprise engineering or business engineering. Other functions may include model management, import/export, and conversion, etc. In addition, an EMDA can also be provided more special engineering/modeling tools, if MDM is used for, e.g., UI modeling, report modeling, and so on.

5.5. Discussions

Our contribution is to discover the way that how to dynamically connect the behaviors (functions or operations) of a computer system (on the left side) to the target things in real world (on the right side) through the inevitable models of the things, where the models are working as a part of the application system and (can be) setting by human in real time. The models in application system are evolutionary; thereby the behaviors of the system can be evolved to follow the changes of the target things via the models, and can be reflected the understanding and intention of the users in time.

As illustrated in MDM, the independence of applied model is one of the most important feature of MDApp, compared to traditional structures (both the structural and object-oriented, the domain model is indeed mixed with code) and MDApp, some newer architecture styles such as DDD displays a transitional features.

For an enterprise, a comprehensive EMDA as an MDApp implements a real separation of the changing of business and the software (the platform); thus, the desired Enterprise Engineering (EE) can be achieved which is based on dynamic enterprise (business) modeling, done by pure
enterprise engineers / business analysts, and supported, combining with the enterprise information system. The enterprise models hence become the main role in EE but not only for the integration between application systems or as the reference for understanding the enterprise / business. The Enterprise Architecture (EA) can more deeply be placed on the planning of concrete business process / function (or say, in finer granularity) since they are evolutionary on the EMDA platform by the modification of the enterprise models, that is, it is able to adaptation to the changes of the enterprise (business) without changing the software.

One important issue is the separation of the concerns about business and computing techniques. The business analysts are required for only the knowledge and skills for modeling the business and use the modeler tool; the programmers of the MDApp platform are also no longer to be needed to understand the business but merely the metamodels. These with the runtime feature and the evolvability of the models fundamentally remove the root of the entanglement of business change and development process.

To the development of enterprise applications, compared to the traditional solutions which programmed to specific business functions directly, the transforming barriers will be effectively avoided; this is not by the automatic transformation from some analysis models or CIMs into design models or code but by bringing the operations of an application system directly based on the evolutionary enterprise models. In our views, the transforming barrier is essential; for MDApp, however, the system analyst / architect faced only the models of a general platform and the model drivers but not the huge cases of concrete business from different customers and situations. We can still use any traditional or exiting approaches to developer the software for MDApp, the difference is, the developers will work on a higher abstract level: the meta-level, thus the use-cases or the functional points they must be programming for will be greatly reduced.

For the topic of raising abstract level, say, a model is also an abstraction, and for MDApp with MDM, the abstract level for programmers is not simply, blindly raised but to be matched with the abstractions (the applied models) of the real word things, such as the business, via its metamodel, that is, the two abstract hierarchies on the two sides of the gap are be matched in an MDApp; for the multi-structure of MDM (e.g., in the Figure 7), it is more obvious and meaningful.

As mentioned above, the applied models can also be of the (part of) system itself, thus one can evolve the (part of) system by modify the corresponding model. This aspect appears to be relevant to, in principle, some research since about 2006 on runtime models, which brings the use of models to the entire life-cycle of software especially during runtime (France & Rumpe, 2007), this research approach more emphasized on software evolution and self-adaptation, the core mechanism is reflection (de Lemos et al., 2011; Vogel, Seibel, & Giese, 2011; Blair, Bencomo, & France, 2009). In contrast, MDApp has functional adaptability to the changing of the target, e.g., the business; in the meantime, the system itself can also be evolved by use MDM on the model of system. Furthermore, all the adaptations on MDApp are achieved by the modeler; this may be somewhat different to the self-adaptation. Whatever, it is interesting topic that the relationships between MDM / MDApp and reflection and such the research for models at runtime, as well as model execution.

6. CONCLUSION

What behind business-IT gap is computer-world gap; adhering to the picture depicted by Smith (1985), the approaches for filling the gap are divided into two groups, which can be called
left side approaches and right side approaches. We extended the picture to present human; the inevitable mediating model between computer and the real world therefore to be for the three, which representing the target thing in the real world for both human and computer. For filling the gap, many existing efforts on the two sides but not really cross over the gap. On the left side, in software engineering, there are still some fundamental problems do not be removed, such as the transforming barrier between analysis and design model and the entanglement of the business change and development process, although we have developed a lot of approaches, such as OO, MDE, DDD, Agile. On the right side, the existing EE approaches such as EA and EM are not able to change the problem from the solution systems themselves: the inflexibility, the application silos or islands, etc.; this greatly limits the role of EA or EM, makes them difficult to be grounding—can only be used for such the IT planning, governance, or application integration in giant or special enterprises.

Our target is on the right, the real world; the problems for enterprise application are essentially opened. Thus, we must do modeling for the world continuously. The human-understandable/operational and computer-treatable models therefore become the inevitable core artifacts to establish dynamic connection between the two sides of the gap; the conventional, left or right approaches do not really fully utilize of the duality. Model-Driven Mechanism (MDM) is the fundamental principle to make the model working as a part of the application system at runtime; the Model-Driven Application (MDApp) based on MDM is the bridge cross over the gap, which is able to avoid such the transforming barrier, disengage the entanglement, and thereby release more the productivities on the two sides. An MDApp based on evolutionary enterprise models, that is, an Enterprise Model Driven Application (EMDA), can enable real enterprise/business engineering, and bring such the EE/EA approach to be grounding.

This chapter is just a preliminary introduction to MDM and MDApp; it needs more, specific illustration on almost every aspect. We argue that this is a valuable, significant direction for either study or practice.

ACKNOWLEDGMENT

I would like to thank Vicente García Díaz and the editorial team for inviting me to participate in this book. They might have to pay more efforts to my writings because it was a new, hard work for me; all errors are mine. I would also like to thank Markus Kerz, Vincent Hanniet, Andreas Leue, Rui Curado, Andriy Levytskyy, and more friends from the MDE community and blogs on Internet, who gave wise and inspiring discussions and comments, which were very instructive and useful to the work.

REFERENCES


doi:10.1007/3-540-47884-1_16.


doi:10.1007/978-3-642-21210-9_22.


ENDNOTES

1. In this chapter, “enterprise(s)” refers to “one or more organisations sharing a definite mission, goals, and objectives to offer an output such as a product or service” (defined in ISO15704, cited from Kosanke & Martin, 2008). This meaning is used for some important concepts, such as enterprise architecture, enterprise modeling, and enterprise engineering, etc. Accordingly, we prefer use business (uncountable) in the sense that is closer to the transactions or affairs in/of an enterprise.

2. Smith’s statement is: “Mediating between the two is the inevitable model” (Smith, 1985, p. 21); it is quite close to Morrison and Morgan (1999): “models themselves function as mediators”(p. 36). Hence, we adopt the term “mediator” to refer this role of model.

3. This indeed involves some deep topics worth more exploration. There are some preliminary discussions, e.g., *Three Spaces for Entities and Models of Applications*, http://thinkinmodels.wordpress.com/2011/10/28/three-spaces-for-entities-and-models-of-applications/

4. Following this opinion, the model of a domain is quite closed to an ontology.

5. For example, see the discussion on Model Driven Software Network at http://www.modeldrivensoftware.net/forum/topics/have-models-filed

6. For MDE, e.g., said by Frankel (2003): “MDA is about using modeling languages as programming languages rather than merely as design languages” (Preface, xv).

7. SEE, the Society for Enterprise Engineering, was active in the mid-1990s.

8. This is an interesting point waiting for more discussion.

9. The initial illustration for this topic is published here: http://thinkinmodels.wordpress.com/2012/01/19/using-model-driven-mechanism-to-explain-model-driven-software-development/

10. This viewpoint directly appeared in most of the literature (e.g., France & Rumpe, 2007; Kühne, 2006; Favre, 2004).