Model-Driven Software Development

Stephen W. Liddle

1 Introduction

Software development is a complex and difficult task that requires the investment of significant resources and carries major risk of failure. According to its proponents, *model-driven* (MD) software development approaches are improving the way we build software. Model-driven approaches putatively increase developer productivity, decrease the cost (in time and money) of software construction, improve software reusability, and make software more maintainable. Likewise, model-driven techniques promise to aid in the early detection of defects such as design flaws, omissions, and misunderstandings between clients and developers. The promises of MD are rather lofty, and so it is only natural to find many skeptics.

As Brooks famously described [Bro95], software engineering will not likely deliver the sort of productivity gains we experience in hardware engineering where we see "Moore's law"-styled doublings every 24 months [Moo10]. Thus, if we accept Brooks' premise, nobody should expect any innovative approach to software development to be a "magical silver bullet" that will increase productivity by an order of magnitude within a decade. Unfortunately, the amount of hyperbole surrounding the various flavors of MD sometimes makes it seem like advocates believe MD to be a silver bullet. Model-driven development is no panacea. However, we believe that *model-driven* is a superior approach to software construction. This chapter examines the current state of the art in model-driven software development.

We begin by characterizing the various approaches to model-driven development (Section 2). Then we examine what modeling is and why we engage in modeling (Section 3). Next, we explore the history of software modeling that has led to current model-driven approaches and discuss what is required

Stephen W. Liddle
Brigham Young University, Provo, Utah 84602, USA e-mail: liddle@byu.edu

to make our models formal and executable (Section 4). With this background laid out, we explore the details of various model-driven approaches to software development. We commence with a reference model (Section 5) and then examine MDA (Section 6), giving special attention to the Executable UML variant of MDA (Section 6.3). In Section 7 we describe the OO-Method approach to MD and the comprehensive OlivaNova tool that implements this approach. We next explore MD in the context of web engineering (Section 8) and then examine the argument for an agile approach to MD (Section 9). We conclude by summarizing available tools, arguments for and against, and directions for future research (Section 10).

2 Overview of Model-Driven Approaches

There are numerous ideas that come under the umbrella of *model-driven approaches*. We take an expansive view of what "model-driven" means. *Model-driven engineering* (MDE) and *model-driven development* (MDD) are generic terms describing an approach where we represent systems as models that conform to metamodels, and we use model transformations to manipulate the various representations (see, for example, [Ken02, Bro04, Béz05, Oli05]). We use the terms MDD and MDE interchangeably.

Although the phrase "model-driven" has been used for decades with respect to software development, one of the earliest mentions of a "model-driven approach" comes from the work done by Embley et al. on Object-oriented Systems Modeling (OSM) (see [EKW92]; the book's subtitle is "A Model-Driven Approach"). Object-oriented methodologies were a topic of lively discussion in the early 1990's, and OSM eschewed any specific software process methodology in favor of letting model creation drive the development process. This is analogous to the idea of maps and directions: when someone needs help driving to an unfamiliar destination, we can either give turn-by-turn instructions on how to drive from their current location, or we can give them the address of the destination and let them use their own map to determine a route. If the path is relatively straightforward and there are no unexpected delays or impediments along the way, the *instructions* approach may be superior. But when exceptions occur, the *map* approach may be superior. In practice, a hybrid approach often gives the best of both worlds: expert guidance based on local knowledge can help travelers avoid common pitfalls, but their ability to read maps provides an improved mental model of the travel process, and makes them more resilient in the face of unexpected challenges. By taking a model-driven approach to software development, OSM focuses developers on creating models as central artifacts of interest, and remains independent of, and neutral with respect to, any particular software process methodology.

A notable MDD initiative is the Object Management Group (OMG) Model Driven Architecture (MDA) [SG03, ME01, ME03]. MDA can be

viewed as an instance of MDD where the core standards and tools are the OMG standards—Unified Modeling Language (UML), MetaObject Facility (MOF), XML Metadata Interchange (XMI), and the Common Warehouse Metamodel (CWM). Because OMG is an influential industry consortium, MDA has gathered considerable attention. However, just as UML is not the only object-oriented modeling language, so also MDA is not the only model-driven approach. There are numerous non-MDA initiatives—commercial and academic—that continue to advance the state of the art in MDD.

Metaprogramming, where a program manipulates itself or another program, often leads to forms of programming that are arguably model-driven, or at least model-based. One class of metapgrogramming, template-based generic programming, starts with a modeling process to create a template from which programs can be generated. The related field of *domain-specific languages* is also inherently a model-based approach. A domain-specific language, in contrast with a general-purpose programming language, models aspects of a particular problem domain and provides special-purpose constructs tailored to the needs of that domain.

Similarly, in the field of modeling, *domain-specific modeling* (DSM) uses a modeling language customized to a particular domain to represent systems, and often includes the ability to generate code for corresponding software systems. CASE tools are forerunners of DSM languages and tools, but they are not the same. A CASE tool is created by a vendor to address a class of software engineering problems, whereas DSM tools let clients create custom domain-specific models and generate code using models and concepts that are specific to the client's particular needs.

Another closely related area is *generative programming*, which seeks to model families of software systems so that they can be created assembly-line style; the central idea is to generate code for each of the desired systems in an automated way from a *generative domain model* [CE00]. Many researchers have seen a need to industrialize software production. The *software factories* work gives an excellent description of this perspective [GS03, GS04]. Research in software architecture has demonstrated that we can increase productivity by developing families of software systems as a product line rather than as one-off creations [CE00, CN01].

MD is also a potential solution to the problem of integrating inherently heterogeneous systems whose development requires the multi-disciplinary talents of workers who are experts in widely differing domains [SK97]. Consider, for example, the software required in a modern automobile. Physical control systems may manage the engine, brakes, and passenger-restraint systems in a mostly automatic and hidden manner, while an in-dash touch-driven display may give access to a more traditional information system that offers satellite navigation and mapping features, media playback, and the viewing of statistical information. The field of *model-integrated computing* [SK97, Szt01, Spr04] brings MD techniques to the problem of engineering these types of systems.

As models become increasingly important, so too does the management of models and particularly of transformations or mappings between models. Research on *generic model management* addresses techniques for treating models and mappings as first-class objects that have high-level operations to simplify their management [BHP00]. The Rondo project provides a working prototype of a programming platform for generic model management [MRB03, Mel04].

If a model is a representation of a system, then in some sense, programming in any language involves some kind of model. A C++ programmer thinks about the subject domain in terms of C++ classes and instances. A SQL programmer views the subject domain through the lens of tables that have rows and typed columns. Whether we explicitly create artifacts we call models—especially *conceptual models*—or whether we implicitly map between our internal mental models of the world and the systems we produce, we are nonetheless involved in a modeling process as we construct software. And so MD is more about raising the level of abstraction of our programming models rather than introducing models into the process in the first place.[1]

Indeed, as Brown points out [Bro04], there is a spectrum of modeling from code-only solutions to model-only solutions (see Figure 1). As we have argued, even in the code-only scenario, developers still create mental models and informal models, but the system representation is entirely in the code. In the second scenario, the developer uses models primarily as a means for visualizing the code; a reverse engineering tool reads the code and displays a corresponding model view of what is captured in the code. In the roundtrip engineering scenario, a tool maintains tight correspondence between model and code; changes made to the code are immediately reflected in the model and vice versa. In the model programming scenario, the model is the code, and the lower level of code is simply generated and compiled behind the scenes; all changes to the system happen at the model level[2] (see Chapter 0 for a detailed discussion of *conceptual model programming*). The final scenario is what happens when either we model without creating an operational system, or we develop models that are never formally tied to the operational system; perhaps we start by creating an ER diagram to generate an initial database schema, but then we evolve the schema independently of the model so that they become disconnected. We view the roundtrip engineering and model programming scenarios as *model-driven*, while the others are at best *model-based* or *model-aware*.

[1] Our argument is a special case of the assertion in [Béz05] that "[m]odeling is essential to human activity because *every* action is preceded by the construction (implicit or explicit) of a model" (emphasis added).

[2] This is analogous to compiling a FORTRAN program by translating it to assembly language, which is then assembled and linked. The assembler version is to "code" as the FORTRAN version is to "model".

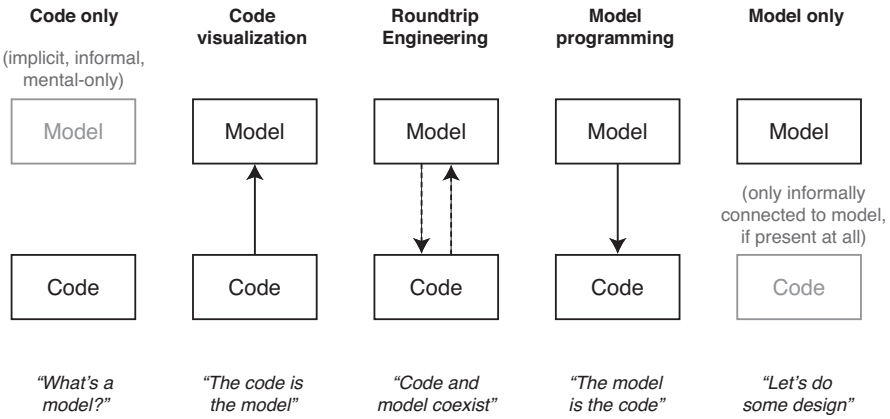

Fig. 1 The Modeling Spectrum (adapted from [Bro04]).

3 Modeling

To understand model-driven software development, it is helpful to review some background on models and modeling. What is a model, and why do we as humans and as software developers build models?

The term *model* is heavily overloaded. A model may be "a set of designs ... for a projected building or other structure," or "a three-dimensional representation" of such a building or structure [OED]. In the mathematical logic sense, a *model* is a "set of entities that satisfies all the formulae of a given formal or axiomatic system" [OED]. Within software development, we also overload the term. In this chapter, when we say "model" without an adjective we mean a diagram or model instance that conforms to a particular modeling language.

Models come in many forms. Three useful categories of models include graphical, mathematical, and textual. A graphical model is a two-dimensional diagram that graphically depicts concepts using a combination of lines, shapes, symbols, and (usually) some text (e.g., an ER diagram). A mathematical model describes some aspect of a system as a formula (e.g., $A = \pi r^2$ is a model for the area of a circle). A textual model describes a portion of a system using narrative and prose (e.g., we can view a "scenario" description as a textual model of a process).

No matter the particular form, common to all models is that they represent some aspect of a system that the modeler is studying or creating [Sei03]. To be useful, such a representation abstracts (separates or summarizes) some aspect of the underlying system. Abstractions are helpful because they let us focus on specific aspects of a system without needing to simultaneously consider the complexity of the full system.

Implicit in our definition of "model" is the fact that it is a written artifact. As humans we form abstractions of the world around us—we classify, generalize, associate, and otherwise construct purely mental models. It is in the writing down of our models that we make it possible to share, discuss, revise, and implement software systems that conform to those models [Bro04].

Software engineers create models for many of the same reasons architects and engineers create blueprints and 3D miniatures:

- Models help us communicate more effectively with the many stakeholders who need to participate in the software development process. For example, a client usually finds it easier to understand a graphical class diagram than, say, C++ source code. Improved communication leads to increased understanding, more reasonable expectations, and a better overall work product.
- Models let us visualize the finished product without requiring its full construction first. By examining the model we can discover design flaws that are far less expensive to resolve up-front rather than after construction has begun (or worse, been completed). In the same way a 3D model of an automobile can be examined in a wind tunnel to tune its aerodynamic performance, a model of a graphical user interface can be placed in front of typical users early on to test usability characteristics.
- Models constitute precise specifications of work to be done. They provide an accurate roadmap of the work, thus allowing project managers to estimate, schedule, and otherwise plan the construction phase.

The value of models and abstractions in software is substantial, as the history of programming languages and operating systems demonstrates [Mah04]. The history of computing is a study in layers of abstraction. Programming progressed from hard-wired computers to stored-program machines, to assembly languages, to high-level languages, CASE tools, object-oriented systems, and domain-specific languages. Operating systems were introduced to manage the complexities of interfacing with the hardware. "Device drivers" abstracted out the challenges of interfacing to storage devices, printers, and other peripherals so developers could concentrate on application development, not low-level hardware control. The abstraction of "processes" introduced multitasking in a way that allowed software developers to avoid dealing with most of the associated complexity. Each step in the evolution of programming languages and operating systems has introduced higher level abstractions into our tools.

Programmers today commonly think in terms of software objects rather than 0's and 1's stored in a particular location. Software developers can focus on the application domain much more readily because the abstractions they use to build their products are conceptually much closer to that application domain. Today's programmers often develop to highly virtualized platforms—whether it be a Java virtual machine or a web-browser environment that uses HTML, CSS, and JavaScript. And not coincidentally, software development

today is also a study in reuse. Developers commonly leverage large libraries—both built-in and external—in their software projects. An underlying reason for these improvements in the state of software development is that models and abstractions have improved significantly over the years.

4 Software Modeling

Since modeling in general has so many uses and benefits, we should expect modeling to be a major research topic in computing, and indeed this is the case. The decade of the 1970's saw the development of formal approaches to data modeling. Abrial [Abr74] and Senko [Sen75], among others, explored binary relationships as an abstraction for data modeling. Falkenberg built on this work and developed the "object-role model" (ORM) framework that used n-ary relationships as a fundamental data modeling construct [Fal76]. Meanwhile, Chen proposed the highly successful Entity-Relationship (ER) model [Che76] that has become nearly synonymous with database design. Tsichritzis and Lochovski [TL82] and Brodie [BMS84] describe much of the early work on data models and conceptual modeling well.

During the 1980's, researchers studied how to improve data models and experimented with so-called semantic data models that introduced additional constructs with more semantic richness than the earlier, simpler models [TYF86, HK91, PM88]. Richer constructs came with more complex notation, and the results were not always an improvement over the simpler predecessor data models.[3] However, research on semantic data models gave way to work on object-oriented (OO) models [SM88, Boo91, RBP$^+$91, EKW92], which researchers debated hotly in the early-to-mid 1990's.

The so-called OO *method wars* led to the proposal of a unified OO model, and the Unified Modeling Language (UML) emerged in 1995 (as the Unified Method version 0.8) and was subsequently standardized by OMG. The latest version, UML 2.2 [UML09b], defines fourteen different diagram types (see Figure 2), including seven that are structural and seven that are behavioral in nature. As with the work on semantic data models, researchers often criticize UML for its complexity, among other complaints [Tho04, HS05, FGDTS06, Küh08, SS09]. However, UML has become not quite universal, but perhaps ubiquitous, in spite of the criticisms.

Where modeling has worked especially well is in the design of database schemas. From an ER or OO schema, it is straightforward to generate a corresponding normalized relational database schema. Many practitioners equate *conceptual modeling* with *database design* because the early conceptual models only addressed structural aspects of information systems. ER and ORM, for example, do not attempt to model behavior.

[3] Bolchini and Garzotto discovered this in the domain of MDWE as well [BG08]; see Section 8.

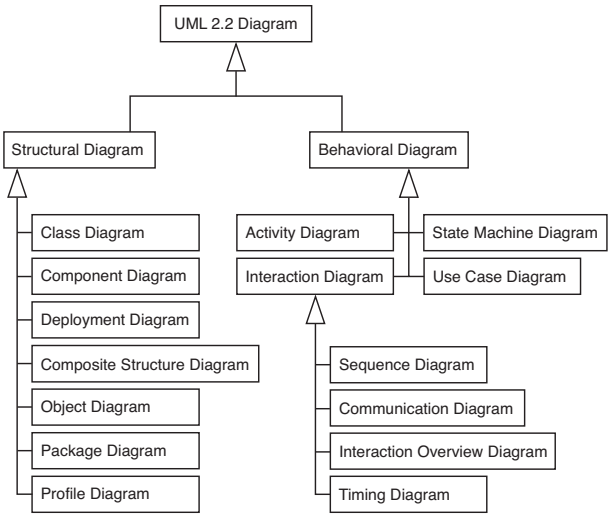

Fig. 2 UML 2.2 Diagram Types (Structural and Behavioral).

In contrast, the OO models have been especially helpful in capturing behavioral aspects of systems. Note that fully half of the diagram types in UML 2.2 address these behavioral aspects (see Figure 2). Generally, the OO paradigm describes *behavior* in terms of the lifecycles of objects (often represented as a state machine) and their interactions with other objects.

When we include system behavior in the model, it becomes possible to generate more than just the system schema from the model; thus we can generate source code for the system, whether in the form of code skeletons, or in the form of fully operational code that compiles into a deployable application.

If we make the behavioral model formal, then it becomes executable. OSM is an example of a modeling language that supports the creation of fully executable models [Lid95, LEW95]. The OSM metamodel is itself expressed formally in OSM [EKW92, Cly93, Lid95, Emb98]. Given a formal metamodel, it is a straightforward process to interpret any particular model instance formally. OSM model instances can be executed simulation-style in a prototyping tool [JEW95] or translated automatically to a model-equivalent language and executed directly [Lid95].

Many other researchers have also advocated software development approaches that begin with executable models. Notable examples include the work of (1) Harel et al. on the Statecharts, STATEMATE, and Rhapsody research line [Har87, HG97, HP98, Har01, GHP02, Har09], (2) Pastor et al. on the OASIS, OO-Method, and OlivaNova group of projects [PHB92, PM07, PEPA08], and (3) Mellor et al. on the Executable UML line of research [MB02, WKC$^+$03, MSUW04, RFW$^+$04]. We address these in subsequent sections of this chapter.

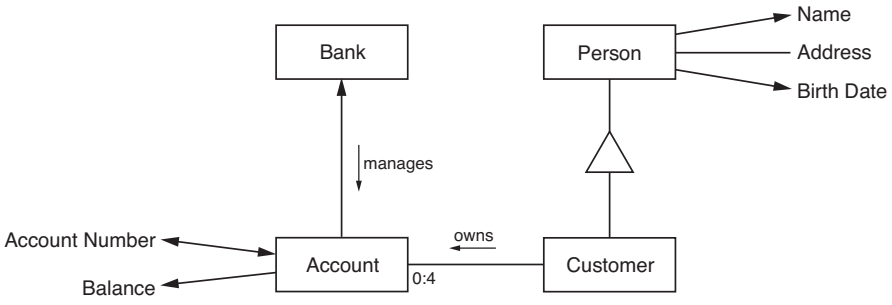

Fig. 3 Banking Example Object-Relationship Model Instance.

What is different about executable conceptual models is that they include behavior and interaction, not just structural components. Furthermore, executable models generally must conform to a precisely, formally specified metamodel so that the model semantics are clear. We explore this concept further by introducing OSM as a reference model.

5 OSM: Making Conceptual Models Formal and Executable

Object-oriented Systems Modeling (OSM) views the software development process as a set of activities with different concerns: analysis, specification, design, implementation, and evolution [EKW92, EJLW94, Lid95, Emb98]. The OSM philosophy is that all these activities should share a single core conceptual modeling language, and shifts in lifecycle phases merely constitute shifts in perspective. Analysis is the study of system, which can be existing or planned; typical analysis-phase activities center around gathering and documenting information regarding user requirements and current system characteristics. Implementation, on the other hand, involves creating a running system that delivers required functions. In OSM, a single core model serves as the basis for all development activities.

OSM has three major views (*diagram types* in UML parlance) for describing object and relationship structure, behavior, and interaction, but all three can be combined in a single seamless model. Figures 3 through 6 give an example. Figure 3 shows a simple object-relationship model instance for the banking domain. Figure 4 shows a simple state net that describes at a high level the behavior of *Account* objects. Figure 5 shows how customers and banks can interact. Figure 6 shows all three views in a single diagram.

OSM notation is fairly consistent with other object-oriented modeling languages. Rectangles represent object sets and lines represent relationship sets. Lexical object sets may either have a dashed border or (as Figure 3 shows)

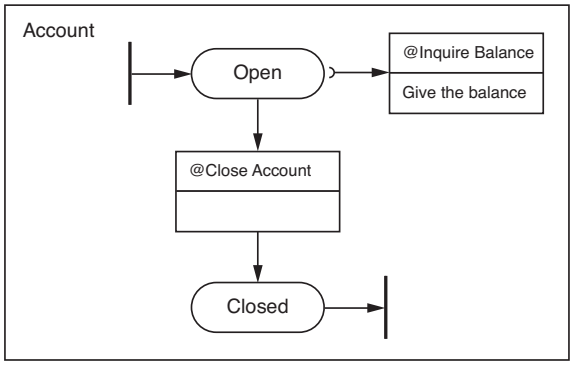

Fig. 4 Banking Example Object Behavior Model Instance (State Net).

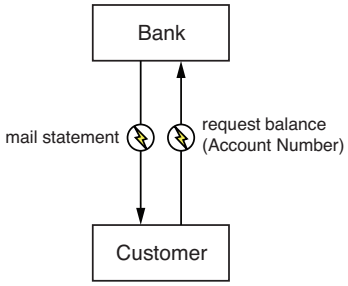

Fig. 5 Banking Example Object Interaction Model Instance.

no border at all. *Min*:*max* numbers near a relationship-set connection indicate participation constraints (e.g., *0:4* near account means that an account can be associated with at most four customers). Names and reading arrows indicate relationship-set names (e.g., *Bank manages Account* and *Customer owns Account* are the two explicit relationship-set names in Figure 3). Arrow heads on relationship-set lines indicate functional relationships (e.g., a person has exactly one name and birth date; an account has exactly one account number and vice versa), while no decoration on the line together with no participation constraint indicates no limit on the number of associations (we can also write this as the participation constraint *0:**). An open triangle represents generalization/specialization (e.g., customer is a person).

A state net (see Figure 4) describes the behavior of an object—its lifecycle from creation to destruction. We write states as rounded rectangles (e.g., *Open* and *Closed*), and transitions as divided rectangles with a transition *trigger* written in the upper portion, and an *action* written in the lower portion of the transition rectangle. A trigger is a boolean expression, and an "@" sign on a trigger indicates the occurrence of an event. When the prior state(s) for a transition are all on, we say that the transition is *enabled*, and it can *fire* when the trigger is true. When a transition fires, it (1) turns

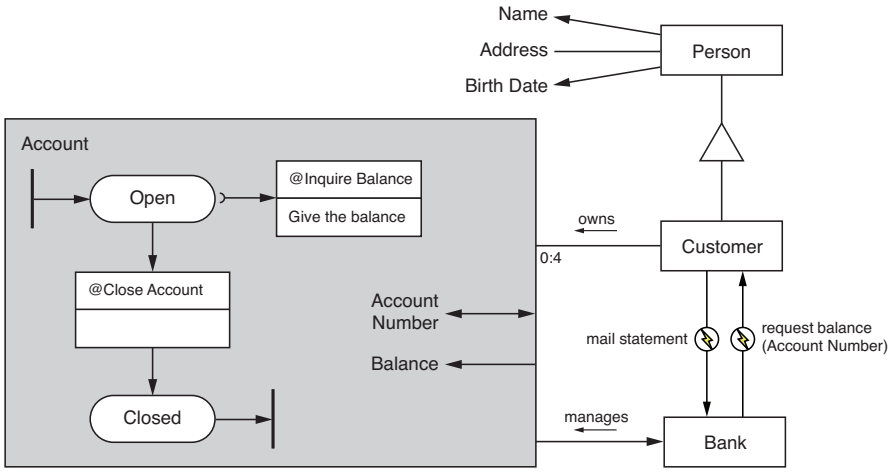

Fig. 6 Combined OSM Diagram Corresponding to Figures 3 through 5.

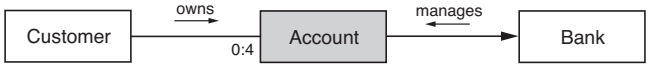

Fig. 7 Collapsed High-level View Corresponding to a Portion of Figure 6.

off prior states, (2) executes the transition action (if any), and (3) turns on any subsequent states. Arrows between states and transitions identify prior and subsequent states, and indicate flow of control. When the tail of an arrow leaving a state has a half circle slightly separated from the state, this indicates that the state is not turned off when the transition fires (i.e., a new, concurrent thread of control can begin when the transition fires). For example, when a balance inquiry transition executes, the corresponding account still remains in the *Open* state, and the *Give the balance* action executes on its own thread. A transition with no prior states indicates an *initial transition* that creates an object, while a transition with no subsequent states indicates a *final transition* that destroys an object. Initial and final transitions may be written as vertical bars as in Figure 4.

An object interaction model instance documents communication or interaction between objects, as Figure 5 shows. In this example, banks mail statements to customers and customers request account balances from banks. When a customer requests an account balance, he or she also indicates the corresponding account number. An arrow with a lightning-bolt symbol at the center of a circle indicates an interaction; the tail of the arrow indicates the interaction origin, while the head indicates the destination.

Figure 6 shows a unified version of Figures 3 through 5. The primary difference is that in Figure 6 we have represented *Account* as a high-level object set with *Account Number*, *Balance*, and two relationship sets nested inside. OSM has fully-reified high-level constructs, meaning that high-level

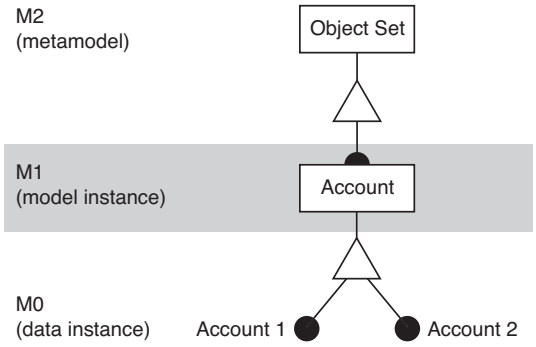

Fig. 8 OSM Three-Tier Seamless Model.

object sets, relationship sets, states, transitions, and interactions are first-class elements that can be treated just like their non-high-level forms. High-level components are helpful for organizing a model and displaying simplified views (see Figure 7, for example).

OSM has a number of features that make it well suited to model execution. As we observed earlier, OSM has a precise formal foundation; notably, the OSM metamodel is itself expressed in OSM and translates directly to first-order predicate calculus [EKW92, Emb98]. Figure 8 shows the layers of OSM's modeling hierarchy. The metamodel (level M2) contains constructs such as *Object Set*, *Relationship Set*, *State*, and *Transition*. The model instance (level M1) contains domain constructs, such as *Bank*, *Customer*, and *Account* in our running example. The data instance (level M0), contains objects and relationships such as particular accounts, their balances, customers, and relationships among these objects. This three-tier model is *seamless* because a single underlying formalism expresses them all, and elements at one level are directly connected by is-a relationships with elements at the next level (*Account 1* and *Account 2* are instances of *Account*, and *Account* is an instance of *Object Set*). Constraints and expressions in OSM can refer to elements within any of the three levels as needed. Furthermore, OSM is computationally complete [Lid95], and so can directly model any algorithm or data structure.

Additionally, OSM embraces a concept called *tunable formalism* [CEW92], allowing users to work at levels ranging from informal to mathematically rigorous. Formal models generally do not provide enough expressiveness or allow the varying levels of detail and completion that practioners need to build real systems. On the other hand, model execution cannot be built on informal models. Because of OSM's precise formal foundation, we can interpret any model instance uniformly both at a summary level and at a detailed level. For example, the model instance in Figure 7 has a completely consistent formal interpretation regardless of whether we include or ignore the details nested within the high-level *Account* object set. Similarly, triggers

associated with transitions could be written in a natural language at first. In this form the statements are merely incomplete w.r.t. model execution: they represent propositional statements whose truth cannot be computed automatically. If we desire automatic computation (as opposed to consulting an external oracle), we simply replace the incomplete statements with an executable refinement.

For example, the natural-language action description *Give the balance* in Figure 6 could be refined to an executable construction by specifying for the interaction a return parameter named *balance* and then replacing the natural-language phrase with the statement `balance := self.Balance`. The right-hand side of the assignment statement is a query that finds the *Balance* object associated with the current *Account* object. Assigning that object to the return parameter completes the action.

OSM has a rapid prototyping tool, IPOST, that allows developers to begin with an analysis-oriented model instance, and then gradually refine it with formal expressions for the various triggers and actions [JEW95]. Using IPOST, a developer can initially populate a model instance with objects and relationships, and then as the system executes, the developer can successively refine it. IPOST automatically generates graphical dialogs to simulate interactions and the firing of transitions. IPOST can simulate the model instance in Figure 7, but it must ask the user (the external oracle) to interpret the effect of the natural-language expression *Give the balance*, whereas we can directly execute the statement `balance := self.Balance` automatically.

OSM was designed to address the poor integration of OO systems across several dimensions, including the following:

1. the software development lifecycle and the models, languages, and tools used to develop software;
2. the so-called *impedance mismatches* between the semantics of persistent objects and behavioral protocols for objects, between declarative and imperative programming paradigms, and between visual and textual styles of programming; and
3. the reification of abstract objects, particularly meta-information and high-level abstractions of low-level modeling components.

OSM addresses the lifecycle issues by using a single modeling and development environment for all activities; changes in development phases or activities are merely shifts in perspective for OSM. Furthermore, the concept of *model-equivalent language* addresses the impedance mismatch issues. In essence, a language L is model-equivalent with respect to a model M if each progam written in L has a corresponding model instance M whose semantics are one-to-one with the program, and vice versa [LEW95, LEW00]. The executable statement described above is written in OSM's model-equivalent language, OSM-L. Using OSM-L, programming becomes just a shift in perspective to focus on efficient algorithms and structures. A "program" is just an alternative view of a "model", and it is easy to iterate rapidly from one

version of the system to another. Also, given OSM's first-class, fully-reified abstract elements (high-level object sets, relationship sets, states, transitions, and interactions), OSM provides the considerable expressiveness and flexibility needed for MDD. OSM does not have high-quality commercial tool support, but it does serve as a complete reference model for MDD.

6 Model-Driven Architecture (MDA)

We now give an overview of MDA (Section 6.1), discuss the MDA Manifesto (Section 6.2), describe Executable UML (Section 6.3), and point to further MDA readings (Section 6.4).

6.1 MDA Overview

The Object Management Group (OMG) is an industry consortium established in 1989 with the goal of defining standards for interoperability for distributed object systems. Their initial effort revolved around the Common Object Request Broker Architecture (CORBA) middleware standard. Their next major standard was the Unified Modeling Language (UML), adopted as a standard at UML version 1.1, in 1997. Following adoption of the UML standard, OMG began to work on its model-driven architecture initiative. OMG adopted the Model Driven Architecture (MDA) standard in 2001 [OMG]. In a nutshell, MDA is model-driven development that uses the core OMG standards (UML, MOF, XMI, CWM).

The three primary goals of MDA are (1) portability, (2) interoperability, and (3) reusability, and the key abstraction for delivering on these goals is "architectural separation of concerns" [ME03].

MDA describes three main layers of architectural abstraction, called *viewpoints*: computation independent, platform independent, and platform specific. As Figure 9 shows, MDA describes systems using models that correspond to the three viewpoints. A *computation independent model* (CIM) describes a system environment and its requirements using terminology that is familiar to practitioners in the system domain. A *platform independent model* (PIM) describes a system's structure and functions formally, and yet without specifying platform-specific implementation details. At the lowest level of the MDA architecture, a *platform specific model* (PSM) includes details that are important to the implementation of a system on a given platform. By *platform*, MDA means a cohesive set of subsystems and technologies on which a system can execute (such as Sun's Java EE or Microsoft's .NET platforms, for example).

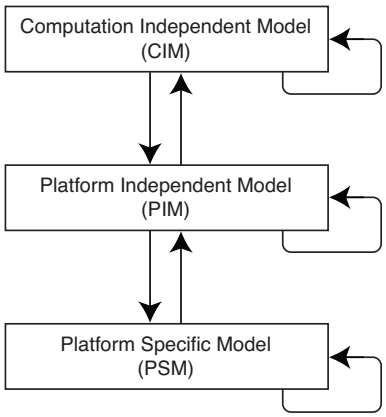

Fig. 9 MDA Architectural Layers and Model Transformations.

As Figure 9 suggests, model mappings or transformations are a key aspect of MDA. Each arrow in Figure 9 represents a transformation from one model to another. Mappings happen at many levels and for many purposes, and MDA does not try to specify precisely how mappings can occur, but a key aspect of MDA transformations is that each mapping may involve the addition of information external to the source model. For example, when mapping from a PIM to a PSM that targets the Java EE platform, the transformation would likely need to combine a sizeable Java EE model that includes formal descriptions of various Java EE abstractions—such as messaging and storage frameworks—with the PIM to generate Java code that implements the PIM abstractions within the Java EE framework. The resulting PSM could then be compiled, deployed, and executed on a Java virtual machine.

Transformations are not merely one way, CIM-to-PIM and PIM-to-PSM. There are mappings between models up and down as Figure 9 suggests. CIM-to-CIM or PIM-to-PIM mappings represent model refinements, such as the transformation that occurs when moving from an analysis phase into a design phase [ME01]. A PSM-to-PSM transformation may be required in order to configure and package the elements of a PSM for deployment to the desired target environment. A PSM-to-PIM transformation may be required when refactoring or reverse-engineering a system.

MDA does not expect that there will be only one CIM, one PIM, and one PSM for any given system. Each model only captures a single view of the system, and a complete system may consist of many CIM's and PIM's. One of the main benefits of taking a model-driven approach is that the implementation step, PIM-to-PSM transformation, can presumably be done relatively easily for multiple platforms. Thus, there may be many PSM's corresponding to each of the target platforms.

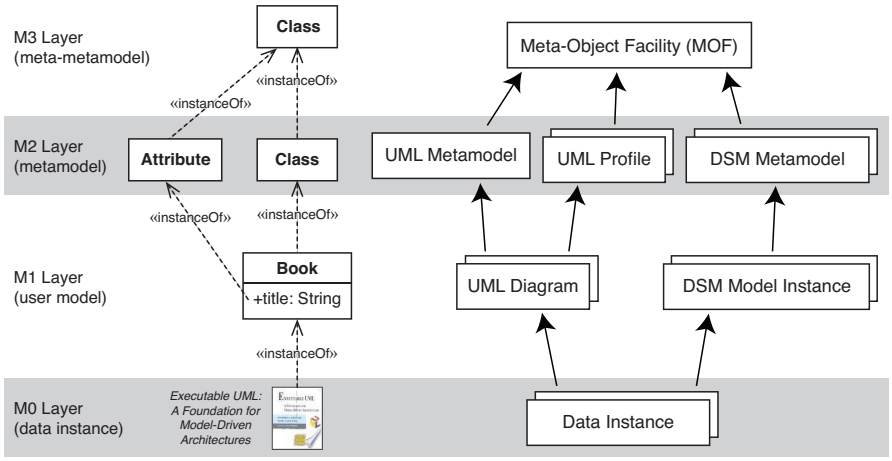

Fig. 10 MDA Modeling and Metamodeling Layers.

The MDA Guide discusses a wide variety of transformation types, techniques, and patterns [ME03]. As with MDD in general, the concept of model transformations is central to the MDA philosophy.

Also central to the MDA philosophy is the role of modeling layers and metamodels. Figure 10 illustrates some important MDA dimensions. UML, and hence MDA, has four modeling layers:

M3: The *meta-metamodel* layer; describes concepts that appear in the metamodel, such as *Class*. For UML, MOF describes the M3 layer.

M2: The *metamodel* layer; describes concepts that make up a modeling language; examples include the UML metamodel, the Executable UML profile, and a domain-specific metamodel created and customized for a particular company or industry segment.

M1: The *user model* or *model instance* layer; class diagrams, statecharts, and other such artifacts are M1-layer elements.

M0: The *data instance* layer; objects, records, data, and related artifacts exist at this level.

In contrast, recall that OSM has three modeling layers because the OSM metamodel is itself defined using OSM, and thus the M2 and M3 layers collapse for OSM. Regardless of the specific structure, a formal metamodel is vital to MDD.

As Figure 10 suggests, an MDA process may use any of a number of different UML profiles or domain-specific metamodels, rather than using UML exclusively for all modeling activities. While developers usually produce UML diagrams using UML or UML profiles, it is also possible to create an MDA process that uses a MOF-conforming domain-specific metamodel to then perform domain-specific modeling tasks within the MDA framework.

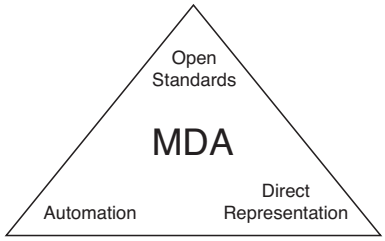

Fig. 11 Basic Tenets of the MDA Manifesto (adapted from [BBI+04]).

6.2 An MDA Manifesto

In 2004, proponents of and major contributors to the MDA initiative working at IBM Rational Software published an "MDA Manifesto" describing the tenets that motivate MDA [BBI+04]. Figure 11 illustrates the three basic tenets of the MDA Manifesto: (1) direct representation, (2) automation, and (3) open standards. We summarize each of these in turn.

The principle of *direct representation* expresses a desire to shift the focus of software development away from the technology domain and toward the concepts and terminology of the problem domain. The goal is to represent a solution as directly as possible in terms of the problem domain. The expectation is that this will lead to more accurate designs, improved communication between various participants in the system development process, and overall increased productivity.

The principle of automation endorses the concept of using machines to perform rote tasks that require no human ingenuity, freeing software developers to focus on creative problem-solving work. Just as database developers today give little thought to the implementation of B-trees, so too should MDA developers be able to ignore technological aspects of graphical interfaces, web services, or any of a hundred other elements of an underlying technology platform. It may be nice to know that a database index is implemented using a B-tree or that a particular communication link is implemented via a WSDL/SOAP web service, but dealing directly with the underlying implementation is not productive *per se*; it is the solving of business problems that creates value and thus constitutes productivity.

Building on open standards is important not only because standards promote reuse, but also because they cultivate the building of an ecosystem of tool vendors addressing the various needs of MDA. Since MDA has such a large vision, it is difficult—perhaps impossible—for a single vendor to provide everything that is required to carry out the vision. According to the manifesto authors, a successful ecosystem requires a few large vendors who can develop comprehensive tools, along with many medium-sized vendors and hundreds of small niche vendors. In order to attract such vendors, the ecosystem must provide standards that form the basis for solid interoperability. This turns

out to be one of the points of criticism of MDA, i.e., that vendors have implemented MDA in such a way that even though they conform to the UML and XMI standards, their products are still not interoperable. The manifesto authors point out that this was a downfall of the CASE industry of the 1980's—vendors trying to "go it alone" [BBI+04, FP04].

The manifesto authors describe the MDA ecosystem as a gradually evolving framework that will improve over time. Indeed, it is clear that an enormous amount of energy has been invested in MDA by a number of vendors, researchers, and practitioners over the years. Much of that work is available as open source, such as the Eclipse Modeling Framework (EMF), which integrates with the popular open source Eclipse IDE [Ecl]. An ecosystem of MDA vendors does exist; what remains to be seen is how effective that ecosystem will be over time.

In a more recent follow-up to the MDA manifesto, one of the authors observes that the slow pace of MDA adoption is the result of challenges in three general areas: (1) technical hurdles such as complex tools, vendor lock-in, and lack of a sound theory of MDD, (2) cultural hurdles such as insufficient practitioner awareness of MDD benefits and enormous inertia for alternative software development tools and techniques, and (3) economic hurdles such as the long-term nature of payback for an investment in MDD. Selic concludes that the way forward for MDA may lie in the areas of education, research, and standardization [Sel08].

6.3 Executable UML

One of the most concrete instances of MDA is Executable UML (xUML, also sometimes labeled xtUML for Executable/Translatable UML) [MB02, WKC+03, RFW+04]. The main idea of Executable UML is to define a UML profile that specifies a well-defined subset of UML that includes a precise *action semantics language* (ASL) [WKC+03] that can be used in the procedures associated with states in a model's statechart diagrams. When developers use ASL to specify the underlying state procedures, we can directly compile and execute the full xUML model.

Figure 12 shows a portion of an xUML class diagram. Observe that we associate role names and multiplicity constraints with each association connection. In xUML it is also conventional to name each association with a simple label of the form *Rn* so it is easy to refer to associations uniquely. In our example, *R1* refers to the association *Bank manages Account* (or *Account is managed by Bank*), while *R2* refers to the *Customer owns Account* association. Attributes may have an associated tag as shorthand for an OCL uniqueness constraint. In Figure 12, for example, *id* on *Bank* and *Account*, and *email* on *Customer* have the tag $\{I\}$, which indicates that each corresponding attribute must have a unique value within its class. Additionally,

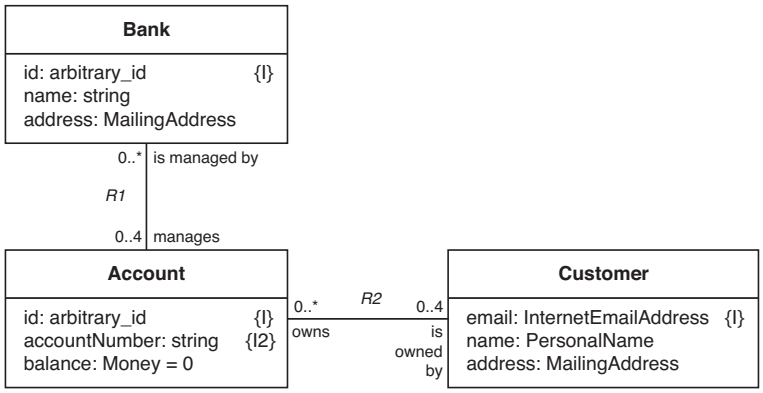

Fig. 12 Executable UML Class Diagram Example.

the tag {*I2*} on *accountNumber* indicates that it also must be unique within *Account*.

Figure 13 illustrates a small portion of a typical xUML statechart diagram for our running banking example. Notice that the state procedures are all written as formal action-language statements that are straightforward to compile into an executable system. As Figure 13 shows, when an account is created it receives an *openAccount* event with two parameters: a customer who will own the new account, and an initial balance. After connecting itself to the given customer, the account initializes its balance to the given amount and sends itself a *ready* signal, which causes the account to advance to the second state, *Waiting for Activity*. In this state, when a *depositOrWithdraw* event occurs, the account updates its balance by the given amount and generates a *done* signal, which causes the account to return to the *Waiting for Activity* state.

As with OSM, it is possible to represent xUML model instances at varying degrees of completion [CEW92]. For example, Figure 13 shows compilable statements in each of the state procedures. However, it is typical in the first version of an xUML statechart to write the procedures informally, as natural-language statements. These can easily be encoded as comments in an action language (e.g., the comment `//`*Connect new account with customer* in Figure 13). For initial stages of work, it is sufficient to capture this sort of behavior requirement informally; in later stages developers refine the model to the point that all requirements are expressed formally. Significantly, even in the early stages when the model is incomplete, it is still possible to simulate the system as far as it is specified. This ability makes it possible to apply agile software development principles to Executable UML [MB02, MSUW04].

Since ASL looks so much like ordinary programming, how can Executable UML really claim any advantage over an ordinary high-level language like C# or Java? The answer may seem subtle, but it is key to understanding the benefit of model execution: ordinary code links computation inextricably

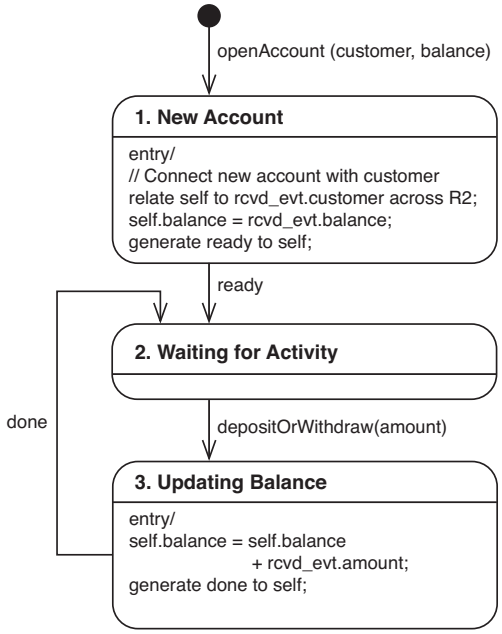

Fig. 13 Executable UML Statechart Diagram Example.

with data structure, while action models and languages separate the two. As Mellor explains (see [MSUW04], p. 95), a common way to find the sum of the last ten transactions is to loop through the transaction history data structure and accumulate the sum with each loop iteration. The action semantics approach to this problem is to divide the problem into (1) retrieving the last ten transaction amounts, and (2) computing their sum. In this way, with action semantics it is possible to change the underlying data structure without affecting the algorithm. This is a key benefit to a model-driven approach: by separating data structures cleanly from algorithms, at translation (or compile) time we can choose different underlying data structures without impacting the algorithmic specifications. Algorithms written according to action semantics are thus written at a higher level of abstraction.

Several tools that support Executable UML include BridgePoint by Mentor Graphics [Bri], iUML by Kennedy-Carter [iUM], and Kavanagh Consultancy's OOA Tool [OOA] (which as of this writing could be downloaded at no charge, but only includes a model editor and not a model compiler).

Executable UML succeeds by narrowing the UML concepts it supports and by focusing on real-time embedded applications. While it is possible to do general-purpose modeling and development with xUML, it excels with applications that have rich and interesting object behavior lifecycles, which is typical in real-time embedded systems.

6.4 MDA Readings

There is a large body of literature that describes MDA. In addition to the OMG publications [OMG, SG03, ME01, ME03], there are a number of helpful books. Kleppe et al. [KWB03] discuss MDA and walk through an extended example of how it actually works. Frankel [Fra03] provides a thorough discussion of strengths and weaknesses of MDA, and especially pays attention to enterprise-scale issues. Mellor et al. [MSUW04] give a concise and clear guide to MDA, and advocate an agile approach to MDA that leverages the strengths of model execution. Nolan et al. [NBB+08] describe MDA from the perspective of the IBM Rational software group that has invested a significant amount of energy in the MDA initiative, including creating commercial tools and software development methodologies around MDA and related standards. Stahl et al. [SVC06] give comprehensive practical guidance. Olivé gives a useful presentation of the underlying concepts, theory, and formalisms of UML and MDA, along with a practical case study [Oli07].

In *The MDA Journal* [FP04], Frankel and Parodi capture the lively debate from a number of blog columns originally published on the BP Trends web site that elucidate the discussion of general-purpose versus domain-specific approaches to MDD and respond to various other criticisms of MDA. This book also contains the MDA manifesto [BBI+04].

Chapter 4 of [GDD06] gives a nice overview of MDA, paying special attention to metamodeling, UML profiles, and model interchange via XMI. Chapter 16 of [DW05] gives a good discussion of what it means to be platform independent and offers criticisms of UML and MDA.

Brown et al. have written several excellent summaries of MDA, issues surrounding MDD in general, and the IBM Rational tools that support MDA [Bro04, BIJ06, Bro08]. Also see [BBG05], which includes two of Brown's MDA papers [BCT05a, BCT05b]. Meservy and Fenstermacher give a concise summary and analysis of MDA [MF05]. Uhl [Uhl08] deals with practicalities of implementing MDD in general at the enterprise level.

A 2003 issue of *IEEE Software* provides a number of helpful articles on MDD and MDA [MCF03]. Selic identifies a number of pragmatic issues surrounding MDD and discusses how tool vendors are addressing them [Sel03]. Seidewitz explores what models mean, and gives a thorough discussion of how we use models and metamodels [Sei03]. Atkinson and Kühne describe the linguistic and ontological dimensions of MDA-style metamodeling and explain how the second version of MDA improves its clarity with respect to these dimensions [AK03]. Sendall and Kozaczynski describe various kinds of model transformations and call for an executable model transformation language [SK03] (see [Mel04]). Kulkarni and Reddy propose "template abstraction" as a means for separating concerns at the model and code levels for improved reuse and system evolution. Finally, Uhl and Ambler engage in a point/counterpoint debate over whether MDA is "ready for prime time", with Uhl claiming it is and Ambler expressing skepticism and asserting that

agile MDD is a better approach [Uhl03, Amb03]. Similarly, a 2008 issue of the UPGRADE journal provides a number of helpful MDA and MDD articles [BVMGMR08].

Finally, Milicev's work [Mil09] is really about *an* executable UML, not Mellor's Executable UML. Milicev links Java, OQL, and other PSM-specific elements into an end-to-end approach to MDD. It is comprehensive, but more platform-specific than most approaches.

7 OO-Method

A significant MDD initiative is OO-Method [PM07] and its realization as the OlivaNova tool suite [CAR]. OO-Method builds on the OASIS formal specification language, which is based on dynamic logic and process algebra [PHB92] and supports precise specification of modeling constructs, or *conceptual patterns*. OO-Method emphasizes the specification of conceptual patterns in precise, unambiguous terms, followed by the combination of *architectural patterns* with the system model. As with OSM and xUML, formal underlying semantics in combination with a sufficient execution model give OO-Method the abilty to compile and execute models directly. The OlivaNova tool includes two main components: the modeler for developing system models, and the transformation engine, which is a model compiler. The OlivaNova transformation engine is one of the most robust commercially available model compilers, and is able to target a number of platforms and architectures.

OO-Method defines four main model types: object model, dynamic model, functional model, and presentation model. The first three constitute the core with which developers create a *conceptual schema*, and the fourth lets developers model how users can interact with the modeled system.

The OO-Method object model contains primitives for capturing structural information. It uses a mostly UML-like notation, with the notable addition of constructs that capture *agent relationships*. In order to invoke a method, an object must first be classified as an agent for that method. In this way, OO-Method supports non-uniform service availability [Nie93], a key aspect of dynamic OO types that some approaches ignore. Agent relationships add another dimension of richness to the encapsulation structure of a system. Figure 14 illustrates the graphical notation for an agent relationship between *Customer* and the *depositFunds* and *withdrawFunds* methods of *Account*.

The OO-Method dynamic model includes fairly typical state transition and object interaction diagrams, but unlike UML statecharts, OO-Method places the specification of service functionality in a separate functional-model layer. Whereas xUML associates procedures with states, OO-Method places these service specifications, which it calls *evaluations*, in the functional model. The functional model specifies how the state of objects can change during their lifecycles. An evaluation has an event that triggers it, an attribute (of some

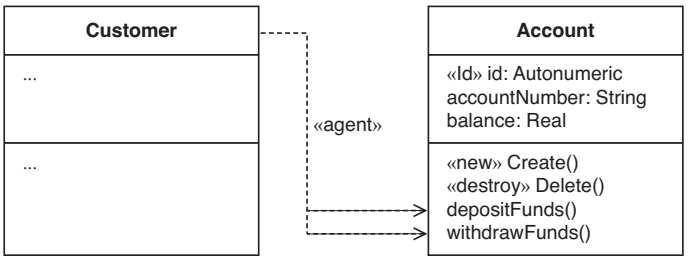

Fig. 14 Agent Relationships in the OO-Method Object Model.

class) that it affects, a condition that may modify when the evaluation can occur, and an evaluation effect that describes the result of performing the evaluation. For example, given the partial class diagram in Figure 14 we may wish to specify an evaluation that automatically issues a service charge when an account that has a negative balance attempts a funds withdrawal. Such an evaluation could specify event *withdrawFunds*, attribute *balance*, evaluation condition $balance < 0$, and evaluation effect $balance = balance - 10$.

A distinctive aspect of OO-Method is its presentation model, which specifies and describes how users can interact with the system. The OO-Method presentation model is essentially a collection of patterns that specifies the user interface as an abstract model that has three levels of patterns: (1) system access structure, (2) interaction units, and (3) basic supporting elements. The framework for the presentation model is an *action hierarchy tree* that defines the hierarchical structure through which users access system functions (e.g., it could be implemented as a menu hierarchy in a typical GUI application). Nodes of the action hierarchy tree are *interaction units* that describe scenarios through which users interact with the system to carry out specific tasks. Basic element patterns support further specification of the user interface as we illustrate below.

OO-Method includes four general kinds of interaction units: service, population, instance, and master/detail. A *service interaction unit* (SIU) models human/computer interaction that results in the execution of a service in the system. Figure 15 shows a simple SIU for depositing funds into an account. Input fields allow the user to specify the account number, amount to deposit, and an explanatory note. A button next to the account entry field lets the user look up the account number from a list. Lower level patterns may also be associated with SIU's. For example, we could add several patterns to the *amount* field in Figure 15, such as (1) an edit mask `##,###.##`, (2) a help message *Please enter the amount you want to deposit*, and (3) an underlying datatype of *Real* for the entered value.

A *population interaction unit* (PIU) specifies patterns for displaying and interacting with collections of objects (such as lists). Figure 16 shows an example for a list of accounts. In addition to displaying the details of an underlying collection (population), this PIU has a filter pattern that lets

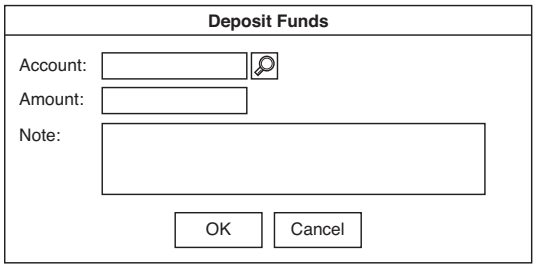

Fig. 15 Deposit Funds Service Interaction Unit for Abstract Graphical Interface.

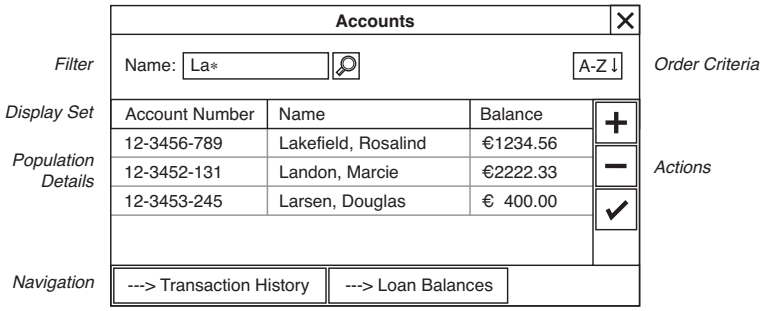

Fig. 16 Accounts Population Interaction Unit for Abstract Graphical Interface.

the user display only accounts whose owner name matches some expression ("La*" in the example) and an order criteria button that allows the user to sort the results according to various terms. The PIU in Figure 16 also has a set of action buttons that invoke specific SIU's (e.g. to add or remove an account) and a set of navigation buttons that move from the current dialog to some other interaction unit (e.g. a transaction history PIU). The other basic-element pattern in Figure 16 is a display-set pattern that indicates which fields associated with accounts should appear in the user interface.

Instance interaction units specify patterns for displaying and interacting with individual objects. Instance and population interaction units are similar, with the exception that the former displays information only about a single object, while the latter displays information about a collection of similar objects. The fourth major category of interaction unit is the *master/detail interaction unit* (MDIU), which models the common scenario where a collection of objects is associated with some other object (e.g., a list of transactions associated with a particular account). Often, the "master" portion of an MDIU is an instance interaction unit and the "detail" portion is a population interaction unit.

With its presentation model, OO-Method is suitable for modeling general-purpose applications that perform typical graphical user interface interactions. Further developing the presentation model to cover additional in-

teraction scenarios is a particularly interesting area of ongoing research [MMP02, PVE+07, PEPA08, AVP10].

OO-Method constitutes an MDA-like approach to model-driven development. It does not use the OMG standards, and so it is not pure MDA. However, we believe that OO-Method could be recast as a UML profile, and thus become pure MDA should its creators choose such a strategy. CARE Technologies [CAR] has put a significant amount of resources into commercializing OO-Method and refining the OlivaNova model execution tool. We see this as one of the more promising model-driven software development projects. It is possible to compile models into complete, operational business systems today using the OlivaNova technology.

8 Model-Driven Web Engineering (MDWE)

Web engineering is a discipline that is ripe for model-driven development because web applications fall into a fairly small set of typical patterns (such as document-centric, workflow-based, transactional, and so forth [KPRR06]) and the architectural concerns of web applications—as opposed to applications in general—are relatively narrow. In response, researchers have created a number of comprehensive approaches to model-driven web engineering (MDWE). Figure 17, adapted from [SK06] and [WSSK07], gives a concise history of many prominent MDWE initiatives, showing how web modeling languages have evolved over time. Wimmer and Schwinger et al. identify five major groupings of MDWE methods:

- *Data-oriented* approaches such as RMM [ISB95], WebML [CFB00, CFB+03, BCFM08], and Hera [Hou00, FHV01, VFHB03, HvdSB+08] have their origins in database systems, and focus on data-intensive web applications.
- *Hypertext-oriented* methods such as HDM [GPS93], HDM-lite [FP00], WSDM [TL98, TCP08], and W2000 [BGP01] originate from work in hypermedia design, and handle nicely the hypertext nature of web applications.
- *Object-oriented* approaches follow in the tradition of OO modeling, and include such methods as OOHDM [SR95b, SR95a, RS08], UWE [HK00, KKZB08], OOWS [PAF01, FPP+08], and OO-H [GCP01].
- *Software-oriented* methods take an approach similar to traditional software development. Web Application Extension (WAE) and its WAE2 extension exemplify this approach [Con03].
- *MDE-oriented* methods explicitly take a model-driven approach to web application development and emphasize the automatic generation of source code from web application models. Examples of this category include Webile [RMP04], WebSA [MG06], MIDAS [VnM04], and Netsilon [MSFB05].

Moreno et al. divide the various MDWE initiatives into two broad groups: those that follow the ER modeling style and those that take an object-

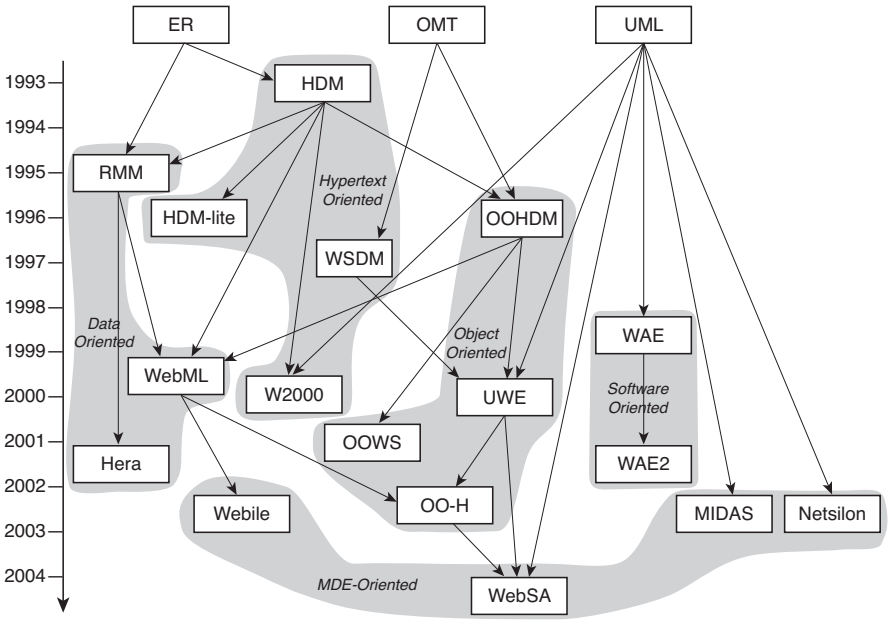

Fig. 17 History of Web Modeling Languages, adapted from [SK06, WSSK07].

oriented approach [MRV08]. In any case, the different categories of MDWE methods reflect diversity both in modeling-language origins and target web application types.

What distinguishes web applications from other types of applications is the prominence of navigation as a construct that should be modeled explicitly and carefully [RSL99]. Navigation modeling essentially consists of describing the information units that users need to navigate along with the structure of the navigation space (which nodes are reachable from which other nodes). It is important to note that the structure of navigation nodes is not the same as the structure of conceptual items in the problem domain. Navigation nodes are likely to consist of views that combine information from parts of multiple domain objects. Furthermore, the navigation space structure is not ideally characterized merely by nodes and links. That works well for simple navigation structures, but typical web applications are more complex and are better modeled by higher-level abstractions such as sets, lists, and navigation chains [RPSO08].

There are other distinguishing aspects of web applications as well. For example, personalization is quite common in web applications now, whereas it is less common in traditional application development. Presentation issues tend to be emphasized in web applications, though the same issues also exist for traditional applications. Web applications typically combine rich media from various sources, and must run properly on a wide variety of differ-

ent browsers, operating systems, and physical devices. The context within which web applications operate evolves quickly, and so do web applications themselves [Mur08]. A repeating them of MDWE is that there are common patterns associated with web applications, and it is helpful to document and reuse these common patterns [GPBV99].

To illustrate MDWE, we examine OOHDM, one of the earliest MDWE methods. The OOHDM method specifies four activities: (1) conceptual modeling, (2) navigation design, (3) abstract interface design, and (4) implementation. After identifying actors, performing use case analysis, and creating a conceptual model of the problem domain—all of which are common to other types of OO development—the OOHDM developer moves to navigation design, which is of particular interest for MDWE. The details are extensive [GSV00, RS08], but briefly, for each user profile, the OOHDM developer creates a navigational class schema and then a context schema that describes web application's navigation design.

An OOHDM *navigational class schema* consists of *nodes*, which are views over conceptual classes that contain information we want the user to perceive, and *anchors*, which are objects that allow the triggering of links. OOHDM structures the navigational space into sets of nodes it calls *navigational contexts*. A unique aspect of navigational contexts is that intra-set navigation is often desirable, as so we define each navigational context in terms of (1) its elements, (2) its internal navigational structure (e.g., can the set be accessed sequentially with next/previous links), and (3) its associated access structures, called *indexes*.

Figure 18 shows an abbreviated example of an OOHDM navigation context diagram for a part of our running banking example. Rectangles with solid borders indicate navigational contexts, while dashed rectangles denote access structures (indexes). Shaded rectangles represent classes (*Account*, *Summary*, and *Activity* in the example). The arrows with black dots at the origin leading from *Main Menu* indicate *landmarks* that are accessible globally from all contexts (implemented, perhaps, as a global menu of links on the top or side of the page). The small black box on *By Account* is a shorthand notation indicating that there is an associated index for this context. Since *Summary* and *Activity* are nested in the same scope, the user can navigate freely between the two views (if this were not desired, we would draw a dashed line between the two).

The various MDWE methods have different levels of support for model-driven development (see especially [RMP04, VnM04, MSFB05, SD05, MG06, SK06, SRS+08, Mur08]). UWE and WebML have some of the more comprehensive tool sets for MDD, though most methods have some tool support. WebRatio Enterprise Edition [WR], by Web Models, is an XML and Java-centric tool that integrates with the Eclipse IDE and supports WebML modeling [WR]. WebRatio generates Java Enterprise Edition (JEE) code from WebML and BPMN models. UWE has a MagicDraw plugin called MagicUWE and an Eclipse plugin called UWE4JSF, among other tools [UWE].

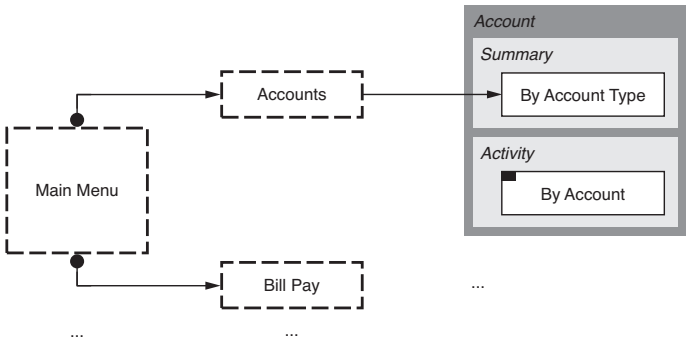

Fig. 18 OOHDM Navigation Context Diagram Abbreviated Example

VisualWade generates PHP code for OO-H web application models [VW], but is a bit dated. The HyperDE tool for OOHDM generates systems for the Ruby on Rails platform, and is suitable for "creating semantic web prototype applications" [HDE].

As MDWE methods have evolved and matured, researchers have expressed increasing concern over macro-level issues such as whether new refinements make the method too complex and cumbersome [MMKV08, SRS+08]. Some have also expressed concern that MDWE is in a state similar to the OO "method wars" of the 1990's and now call for consolidation and standardization efforts [VKC+07, WSSK07]. The MDWEnet initiative [VKC+07] is working on responding to some of these concerns, and we anticipate that researchers will continue to work to bring together the various approaches where feasible. Interestingly, some of the researchers behind HDM and W2000 subsequently decided that a simpler approach would be more effective for various aspects of design, and they proposed the significantly simpler IDM [BG08], which takes a dialog-oriented perspective and advocates conceptual, logical, and page-design activities. We expect that there will be a fair amount of continued refinement and consolidation research in MDWE.

Many workshops and conferences that have published significant MDWE-related work including workshops on the World Wide Web and Conceptual Modeling (WWWCM'99 [CEK+99] and WCM2000 [LMT00]), Web-Oriented Software Technology (IWWOST 2001–present), and Model-Driven Web Engineering (MDWE 2005–present), and the International Conference on Web Engineering (ICWE 2001–present), among others. Several books nicely summarize the state of the art in MDWE [KPRR06, MM06, RPSO08] (note especially that [RPSO08] distills the work published in IWWOST and WCM). Furthermore, an extensive survey by Schwinger et al. [SRS+08] provides excellent details and analysis of MDWE initiatives.

9 Agile MDD

A criticism often leveled against MDD in general, and MDA in particular, is that it is too complex and difficult to use. Ambler argues that the so-called "generative" approaches to MDD, such as MDA, are too complex for the current generation of developers [Amb03]. His argument is that only after a large up-front effort—including a steep learning curve for the chosen modeling languages and tools—can we create the sophisticated models that are required to be able to generate code for the various platforms we want to target.

In contrast, the agile software development movement advocates making customer satisfaction the highest priority, and agilists see early and continuous delivery of useful software as the path to achieving this goal. They value "**individuals and interactions** over processes and tools, **working software** over comprehensive documentation, **customer collaboration** over contract negotiation, and **responding to change** over following a plan" [Agi]. The whole premise of creating complex models and then generating code from those models seems completely counter to these agile principles.

However, as one of the original signatories to the Agile Manifesto points out, there is no conflict between "agile" and "modeling" *per se* [Mel05]. The conflict is in how we often use models. If models are not executable, the reasoning goes, then they cannot be agile. If a model is supposed to be a blueprint against which we later build a software system, then we must first go through the effort of creating the blueprint, and then we must go through a second process of construction. This is a heavyweight, non-agile approach. However, if our models are executable, then we can immediately use them in the way we typically use code (prototyping early and often), and thus the same agile principles that apply to programming apply equally well to modeling [MB02, MSUW04, Mel05].

Ambler believes that MDA is flawed and will not succeed for most organizations. He takes a pragmatic approach and questions MDA along the following dimensions (among others) [Amb]:

- It takes a high level of education and training for developers to use MDA tools. UML (and related standards) are overly complex and may not be what the industry really needs anyway. The MDA standards are incomplete and still evolving.
- Tool vendors have historically been unwilling to create truly interoperable model-sharing standards (CORBA, also an OMG standard, suffered from tool vendors who would announce support and then implement the standard in a proprietary way).
- The industry has seen other approaches, like I-CASE in the 1980's and CORBA in the 1990's, that made similar promises but never fully delivered. Why will MDA be any different?

- Business stakeholders do not ask us to develop detailed, sophisticated, platform-independent models using a precise industry-standard modeling language to describe their business. Developing complex models is not what they request—they want working systems that deliver value.

Ambler's answer is to advocate agile MDD, which replaces the task of creating extensive models with agile models that are "just barely good enough" to drive the software development process.

We share most of the concerns Ambler expressed. MDA is built on a complex set of standards, and those standards do indeed continue to evolve. It is difficult to achieve true interoperability between different vendors' products in spite of their implementation of XMI import/export. History is full of failures in this arena. However, it is possible to apply agile techniques in an MDA framework, and indeed it has been done successfully [MB02]. When we shift from the use of models as sketches or blueprints to the use of models as the executable system itself, many of the difficulties Ambler points out simply go away. Furthermore, we caution readers not to confuse MDA, which Ambler specifically criticizes, with the broader concept of MDD; weaknesses (or strengths) of MDA do not necessarily apply to MDD in general.

10 Conclusions

A wide variety of model-driven methods have been proposed over the years and continue to be developed. MDA is certainly one of the more prominent approaches to model-driven software development, but it is by no means the only method. Model-driven techniques have been applied in a range of domains, and have been particularly well accepted in the field of web engineering. Some researchers advocate an agile approach to MDD because the traditional approach to modeling suffers from the same problems as the waterfall approach to software development.

There is an ecosystem of model-driven software development researchers, vendors, and practitioners. IBM Rational has been a major player in this field, creating many research advances and commercial tools for MDD. The Eclipse project has been prominent as well, with the Eclipse Modeling Framework and numerous related plugins. Some types of MDD have better tool support than others. For example, Executable UML has several good tools (BridgePoint, Rhapsody, and iUML), and the OlivaNova suite is an excellent and comprehensive model compiler.

On the other hand, many tool vendors have struggled to make sustainable model-driven tools. OptimalJ by Compuware was recently discontinued by its vendor, Compuware, even though OptimalJ was generally regarded as technically strong. A search of the web for model-driven tools yields many links to projects that are no longer active. Nonetheless, there are a number of

active vendors with high quality tools available today. Altova's UModel, Artisan's Studio, Borland's Together, Gentleware's Apollo and Poseidon tools, IBM Rational's various tools (e.g., Rhapsody, Rose, and Software Architect), No Magic's MagicDraw, SparxSystems' Enterprise Architect, and Visual Paradigm's tool suite are some (but not all) of the active vendors with quality tools. We expect to see continued energy and innovation in the MDD tool vendor market in the coming years.

The question remains whether model-driven approaches to software development can deliver on the promise of increased productivity, quality, reusability, and maintainability. The skeptics are abundant, particularly among proponents of agile techniques, and the vast majority of software today is still developed using non-model-driven methods. However, the industry has been moving inevitably toward model-driven approaches, and we expect it will continue to do so. We answer the skeptics by taking an expansive view of what "model-driven" means; the phrase is not owned by any one vendor or consortium, and it does not require cumbersome or unwieldy solutions, even though that is what many early MDD proponents delivered. The move toward model-driven approaches is really the same phenomenon that has been occurring in computing for decades—a move to ever higher levels of abstraction.

In his classic essay on software engineering, "No Silver Bullet—Essence and Accidents of Software Engineering", Fred Brooks observed that there are two kinds of complexity: "essential" and "accidental" [Bro95]. His central point was that some complexity aspects of software systems are intrinsic or inherent (essential), while other aspects are artificially (accidentally) complex. Furthermore, essential complexity cannot be removed from the software development process. Therefore, unless accidental complexity accounts for at least 90% of the effort required to develop complex systems, we will never see a "silver bullet" that increases productivity be an order of magnitude.

For example, a software system capable of making highly accurate weather forecasts has significant inherent complexity because the environmental model is quite involved, gathering the many necessary inputs is a difficult distributed process, and the algorithms that manipulate that model and its inputs are computationally complex. However, the particular tools we might use today to build such a system have some measure of accidental complexity. Consider the productivity improvement that comes with using an integrated development environment to develop in a modern object-oriented programming language with its extensive libraries of functions as compared to programming in COBOL or FORTRAN on punch cards. By introducing an environment that conveniently speeds up the edit-compile-run-test cycle, we remove some of the accidental programming complexity that software developers of the 1970's and 1980's experienced. Much of that complexity is now removed with graphical toolbar buttons and library objects. We can store our programs on convenient, stable, solid-state storage that fits in our pockets rather than on paper cards and bulky reels of magnetic tape.

Have we seen order-of-magnitude-scale productivity increases over the years? Yes, certainly we have; but Brooks is still correct. First, he limited his prediction to a one-decade timespan, so if the ten-fold productivity improvement comes over a period of more than ten years, his thesis holds. Second, because our tools improve dramatically over time, we are able to tackle increasingly difficult and challenging tasks, and so we shift the ratio of essential to accidental complexity gradually and quite naturally. Thus, as our capabilities increase, so too does the essential complexity of the systems we choose to build. As the essential complexity increases, we naturally begin to devise additional mechanisms for dealing with that complexity, and consequently the accidental complexity increases as well. Consider the case of the OMG standards: the UML 2.2 specification [UML09a, UML09b] is 966 pages long! As many critics have argued, there is certainly considerable accidental complexity in UML and the other standards around which MDA is built.

This is the context in which we should examine today's model-driven software development approaches. Just as with evolution in nature, ideas in the model-driven arena have variable quality, with only a subset leading to improvements. Nevertheless, the industry has been moving inexorably toward improved abstractions, and it will continue to do so. This is the natural arc of evolution for software development.

References

[Abr74] J.-R. Abrial. Data semantics. In *IFIP Working Conference Data Base Management*, pages 1–60, 1974.

[Agi] Agile manifesto. http://www.agilemanifesto.org.

[AK03] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, Sept.–Oct. 2003.

[Amb] Examining the model driven architecture (MDA). http://www.agilemodeling.com/essays/mda.htm.

[Amb03] S.W. Ambler. Agile model driven development is good enough. *IEEE Software*, 20(5):71–73, Sept.–Oct. 2003.

[AVP10] N. Aquino, J. Vanderdonckt, and O. Pastor. Transformation templates: adding flexibility to model-driven engineering of user interfaces. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 1195–1202, 2010.

[BBG05] S. Beydeda, M. Book, and V. Gruhn, editors. *Model-Driven Software Development*. Springer, Berlin, 2005.

[BBI+04] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic. An MDA manifesto. *The MDA Journal: Model Driven Architecture Straight from the Masters*, pages 133–143, 2004.

[BCFM08] M. Brambilla, S. Comai, P. Fraternali, and M. Matera. Designing web applications with WebML and WebRatio. In Rossi et al. [RPSO08], pages 221–261.

[BCT05a] A.W. Brown, J. Conallen, and D. Tropeano. Models, modeling, and model driven development. In S. Beydeda, M. Book, and V. Gruhn, editors, *Model-Driven Software Development*, pages 1–17. Springer, Berlin, 2005.

[BCT05b] A.W. Brown, J. Conallen, and D. Tropeano. Practical insights into MDA: Lessons from the design and use of an MDA toolkit. In S. Beydeda, M. Book, and V. Gruhn, editors, *Model-Driven Software Development*, pages 403–432. Springer, Berlin, 2005.

[Béz05] J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[BG08] D. Bolchini and F. Garzotto. Designing multichannel web applications as "dialogue systems": the IDM model. In Rossi et al. [RPSO08], pages 193–219.

[BGP01] L. Baresi, F. Garzotto, and P. Paolini. Extending UML for modeling web applications. In *HICSS*, 2001.

[BHP00] P.A. Bernstein, A.Y. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[BIJ06] A.W. Brown, S. Iyengar, and S. Johnston. A rational approach to model-driven development. *IBM Systems Journal*, 45(3):463–480, 2006.

[BMS84] M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer, New York, 1984.

[Boo91] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, Menlo Park, California, 1991.

[BPS02] Michael J. Butler, Luigia Petre, and Kaisa Sere, editors. *Integrated Formal Methods, Third International Conference, IFM 2002, Turku, Finland, May 15-18, 2002, Proceedings*, volume 2335 of *Lecture Notes in Computer Science*. Springer, 2002.

[Bri] Bridgepoint. http://www.mentor.com/products/sm/model_development-/bridgepoint.

[Bro95] F.P. Brooks. *The Mythical Man-Month: Essays on Software Engineering (Anniversary Edition)*. Addison-Wesley, Boston, Massachusetts, 1995.

[Bro04] A.W. Brown. Model driven architecture: Principles and practice. *Software and System Modeling*, 3(4):314–327, 2004.

[Bro08] A.W. Brown. MDA redux: Practical realization of model driven architecture. In *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, pages 174–183, Feb. 2008.

[BVMGMR08] J. Bézivin, A. Vallecillo-Moreno, J. García-Molina, and G. Rossi. Editor's introduction: MDA at the age of seven: Past, present and future. *UPGRADE: The European Journal for the Informatics Professional*, IX(2):4–6, April 2008.

[CAR] CARE-technologies web site. http://www.care-t.com/.

[CE00] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, Boston, Massachussets, USA, 2000.

[CEK+99] P.P. Chen, D.W. Embley, J. Kouloumdjian, S.W. Liddle, and J.F. Roddick, editors. *Advances in Conceptual Modeling: ER'99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November 15-18, 1999, Proceedings*, volume 1727 of *Lecture Notes in Computer Science*. Springer, 1999.

[CEW92] S.W. Clyde, D.W. Embley, and S.N. Woodfield. Tunable formalism in object-oriented systems analysis: Meeting the needs of both theoreticians and practitioners. In *Proceedings of the 1992 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92)*, pages 452–465, Vancouver, Canada, October 1992.

[CFB00] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, 33(1–6):137–157, 2000.

[CFB+03] S. Ceri, P. Fraternelli, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers, San Francisco, California, 2003.

[Che76] P.P. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Cly93] S.W. Clyde. *An Initial Theoretical Foundation for Object-Oriented Systems Analysis and Design*. PhD thesis, Brigham Young University, 1993.

[CN01] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[Con03] J. Conallen. *Building Web applications with UML, 2nd Edition*. Pearson Education, Inc., Boston, Massachusetts, 2003.

[DW05] D. Draheim and G. Weber. *Form-Oriented Analysis*. Springer, Berlin, 2005.

[Ecl] Eclipse project web site. http://www.eclipse.org.

[EJLW94] D.W. Embley, R.B. Jackson, S.W. Liddle, and S.N. Woodfield. A formal modeling approach to seamless object-oriented systems development. In *Proceedings of the Workshop on Formal Methods for Information System Dynamics at CAiSE'94*, pages 83–94, The Netherlands, June 1994.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[Emb98] D.W. Embley. *Object Database Development: Concepts and Principles*. Addison-Wesley, Reading, Massachusetts, 1998.

[Fal76] E.D. Falkenberg. Concepts for modelling information. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 95–109, 1976.

[FGDTS06] R.B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-driven development using UML 2.0: promises and pitfalls. *Computer*, 39(2):59–66, Feb. 2006.

[FHV01] F. Frasincar, G.-J. Houben, and R. Vdovjak. An RMM-based methodology for hypermedia presentation design. In *Advances in Databases and Information Systems, 5th East European Conference, ADBIS 2001, Vilnius, Lithuania, September 25-28, 2001, Proceedings*, volume 2151 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2001.

[FP00] P. Fraternali and P. Paolini. Model-driven development of web applications: the AutoWeb system. *ACM Trans. Inf. Syst.*, 18(4):323–382, 2000.

[FP04] D.S. Frankel and J. Parodi, editors. *The MDA Journal: Model Driven Architecture Straight from the Masters*. Meghan-Kiffer Press, Tampa, Florida, 2004.

[FPP+08] J. Fons, V. Pelechano, O. Pastor, P. Valderas, and V. Torres. Applying the OOWS model-driven approach for developing web applications. the internet movie database case study. In Rossi et al. [RPSO08], pages 65–108.

[Fra03] D.S. Frankel. *Model-Driven Architecture: Applying MDA to Enterprise Computing*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[GCP01] J. Gómez, C. Cachero, and O. Pastor. Conceptual modeling of device-independent web applications. *IEEE MultiMedia*, 8(2):26–39, 2001.

[GDD06] D. Gašević, D. Djurić, and V. Devedžić. *Model Driven Engineering and Ontology Development*. Springer, Berlin, 2006.

[GHP02] E. Gery, D. Harel, and E. Palachi. Rhapsody: A complete life-cycle model-based development system. In Butler et al. [BPS02], pages 1–10.

[GPBV99] F. Garzotto, P. Paolini, D. Bolchini, and S. Valenti. "Modeling-by-Patterns" of web applications. In *Proceedings of WWWCM'99, Lecture Notes in Computer Science, 1727*, pages 293–306, Paris, France, December 1999. Springer.

[GPS93] F. Garzotto, P. Paolini, and D. Schwabe. HDM—a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, 11(1):1–26, 1993.

[GS03] J. Greenfield and K. Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *OOPSLA'03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27, New York, 2003. ACM.

[GS04] J. Greenfield and K. Short. *Software Factories : Assembling Applications With Patterns, Models, Frameworks, And Tools.* John Wiley, Inc., Indianapolis, Indiana, 2004.

[GSV00] N. Güell, D. Schwabe, and P. Vilain. Modeling interactions and navigation in web applications. In S.W. Liddle, H.C. Mayr, and B. Thalheim, editors, *Conceptual Modeling for E-Business and the Web, ER 2000 Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling, Salt Lake City, Utah, USA, October 9-12, 2000, Proceedings*, volume 1921 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2000.

[Har87] David Harel. Statecharts: A visual formulation for complex systems. *Scientific Computer Programming*, 8(3):231–274, 1987.

[Har01] D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 36(1):53–60, January 2001.

[Har09] D. Harel. Meaningful modeling: What's in the semantics of "semantics"? *Communications of the ACM*, 52(3):67–75, March 2009.

[HDE] HyperDE web site. http://www.tecweb.inf.puc-rio.br/hyperde.

[HG97] D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, (7):31–42, July 1997.

[HK91] R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):605–625, 1991.

[HK00] R. Hennicker and N. Koch. A UML-based methodology for hypermedia design. In *UML 2000 – The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2–6, 2000, Proceedings*, volume 1939 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2000.

[Hou00] G.-J. Houben. HERA: Automatically generating hypermedia front-ends. In *Engineering Federated Information Systems*, pages 81–88, 2000.

[HP98] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The Statemate Approach.* McGraw-Hill, Inc., New York, NY, USA, 1998.

[HS05] B. Henderson-Sellers. UML—the good, the bad or the ugly? perspectives from a panel of experts. *Software and System Modeling*, 4(1):4–13, 2005.

[HvdSB+08] G.-J. Houben, K. van der Sluijs, P. Barna, J. Broekstra, S. Casteleyn, Z. Fiala, and F. Frasincar. HERA. In Rossi et al. [RPSO08], pages 263–301.

[ISB95] T. Isakowitz, E.A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, 1995.

[iUM] iUML. http://www.kc.com/PRODUCTS/iuml/.

[JEW95] R.B. Jackson, D.W. Embley, and S.N. Woodfield. Developing formal object-oriented requirements specifications: A model, tool and technique. *Information Systems*, 20(4):273–289, 1995.

[Ken02] S. Kent. Model driven engineering. In Butler et al. [BPS02], pages 286–298.

[KKZB08] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. UML-based web engineering. In Rossi et al. [RPSO08], pages 157–191.

[KPRR06] G. Kappel, B. Prýýll, S. Reich, and W. Retschitzegger, editors. *Web Engineering: The Discipline of Systematic Development of Web Applications.* John Wiley & Sons, Hoboken, New Jersey, 2006.

[Küh08] T. Kühne. Making modeling languages fit for model-driven development. http://www.mm.informatik.tu-darmstadt.de/ kuehne/publications/papers/making-fit.pdf, 2008.

[KVH07] Nora Koch, Antonio Vallecillo, and Geert-Jan Houben, editors. *Model-Driven Engineering 2007. Proc. of the 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007)*, volume 261 of *CEUR Workshop Proceedings*, Como, Italy, July 2007. CEUR-WS.org.

[KWB03] A.G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecure: Practice and Promise*. Pearson Education, Inc., Boston, Massachussetts, 2003.

[LEW95] S.W. Liddle, D.W. Embley, and S.N. Woodfield. Unifying Modeling and Programming Through an Active, Object-Oriented, Model-Equivalent Programming Language. In *Proceedings of the Fourteenth International Conference on Object-Oriented and Entity-Relationship Modeling (OOER'95), Lecture Notes in Computer Science, 1021*, pages 55–64, Gold Coast, Queensland, Australia, December 1995. Springer.

[LEW00] S.W. Liddle, D.W. Embley, and S.N. Woodfield. An active, object-oriented, model-equivalent programming language. In M.P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, pages 333–361. MIT Press, Cambridge, Massachusetts, 2000.

[Lid95] S.W. Liddle. *Object-Oriented Systems Implementation: A Model-Equivalent Approach*. PhD thesis, Department of Computer Science, Brigham Young University, Provo, Utah, June 1995.

[LMT00] S.W. Liddle, H.C. Mayr, and B. Thalheim, editors. *Conceptual Modeling for E-Business and the Web, ER 2000 Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling, Salt Lake City, Utah, USA, October 9-12, 2000, Proceedings*, volume 1921 of *Lecture Notes in Computer Science*. Springer, 2000.

[Mah04] M.S. Mahoney. Finding a history for software engineering. *IEEE Annals of the History of Computing*, 26(1):8–19, jan-mar 2004.

[MB02] S.J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachussets, 2002.

[MCF03] S.J. Mellor, A.N. Clark, and T. Futagami. Model-driven development—guest editor's introduction. *IEEE Software*, 20(5):14–18, Sept.–Oct. 2003.

[ME01] J. Miller and J. Mukerji (Eds.). Model driven architecture (MDA). http://www.omg.org/cgi-bin/doc?ormsc/01-07-01.pdf, 2001.

[ME03] J. Miller and J. Mukerji (Eds.). MDA guide version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2003.

[Mel04] S. Melnik. *Generic Model Management: Concepts and Algorithms*. Springer, Berlin, 2004. Lecture Notes in Computer Science, no. 2967.

[Mel05] Stephen J. Mellor. Editor's introduction: Adapting agile approaches to your project needs. *IEEE Software*, 22(3):17–20, 2005.

[MF05] T.O. Meservy and K.D. Fenstermacher. Transforming software development: an MDA road map. *Computer*, 38(9):52–58, sept. 2005.

[MG06] S. Meliá and J. Gómez. The WebSA approach: Applying model driven engineering to web applications. *Journal of Web Engineering*, 5(2):121–149, 2006.

[Mil09] D. Milicev. *Model-Driven Development with Executable UML*. Wiley Publishing, Inc., Indianapolis, Indiana, 2009.

[MM06] E. Mendes and N. Mosley. *Web Engineering*. Springer, Berlin, Germany, 2006.

[MMKV08] N. Moreno, S. Meliá, N. Koch, and A. Vallecillo. Addressing new concerns in model-driven web engineering approaches. In J. Bailey, D. Maier, K.-D. Schewe, B. Thalheim, and X.S. Wang, editors, *WISE*, volume 5175 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[MMP02] P.J. Molina, S. Meliá, and O. Pastor. User interface conceptual patterns. In *Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS'02)*, Rostock, Germany, June 2002.

[Moo10] Moore's law. http://en.wikipedia.org/wiki/Moore's_law, May 2010.

[MRB03] S. Melnik, E. Rahm, and P.A. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'03)*, San Diego, California, 2003.

[MRV08] N. Moreno, J. Raúl Romero, and A. Vallecillo. On overview of model-driven web engineering and the MDA. In Rossi et al. [RPSO08], pages 353–382.

[MSFB05] P.-A. Muller, P. Studer, F. Fondement, and J. Bézivin. Platform independent web application modeling and development with Netsilon. *Software and System Modeling*, 4(4):424–442, 2005.

[MSUW04] S.J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, Boston, Massachussets, 2004.

[Mur08] S. Murugesan. Web application development: Challenges and the role of web engineering. In G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, editors, *Web Engineering: Modelling and Implementing Web Applications*, pages 7–32. Springer, Berlin, 2008.

[NBB+08] B. Nolan, B. Brown, L. Balmelli, T. Bohn, and U. Wahli. *Model Driven Systems Development with Rational Products*. IBM Redbooks, 2008.

[Nie93] O. Nierstrasz. Regular types for active objects. In *Proceedings of the 1993 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, pages 1–15, 1993.

[OED] Oxford english dictionary. http://www.oed.com/.

[Oli05] A. Olivè. Conceptual schema-centric development: A grand challenge for information systems research. In *CAiSE*, pages 1–15, 2005.

[Oli07] A. Olivè. *Conceptual Modeling of Information Systems*. Springer, Berlin, Germany, 2007.

[OMG] Object Management Group home page. www.omg.org.

[OOA] OOA Tool. http://www.ooatool.com/OOATool.html.

[PAF01] O. Pastor, S.M. Abrahão, and J. Fons. An object-oriented approach to automate web applications development. In *Electronic Commerce and Web Technologies, Second International Conference, EC-Web 2001 Munich, Germany, September 4-6, 2001, Proceedings*, volume 2115 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2001.

[PEPA08] O. Pastor, S. España, J.I. Panach, and N. Aquino. Model-driven development. *Informatik Spektrum*, 31(5):394–407, 2008.

[PHB92] O. Pastor, F. Hayes, and S. Bear. OASIS: An object-oriented specification language. In *Proceedings of International Conference on Advanced Information Systems Engineering (CAiSE'92)*, pages 348–363, Manchester, United Kingdom, 1992.

[PM88] J. Peckham and F. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[PM07] O. Pastor and J.C. Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer, New York, New York, 2007.

[PVE+07] I. Pederiva, J. Vanderdonckt, S. España, J.I. Panach, and O. Pastor. The beautification process in model-driven engineering of user interfaces. In *Proceedings of the 11th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT 2007)*, Rio de Janeiro, Brazil, September 2007.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[RFW+04] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie. *Model Driven Architecture with Exectuable UML*. Cambridge University Press, Cambridge, UK, 2004.

[RMP04] D. Di Ruscio, H. Muccini, and A. Pierantonio. A data-modelling approach to web application synthesis. *Int. J. Web Eng. Technol.*, 1(3):320–337, 2004.

[RPSO08] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, editors. *Web Engineering: Modelling and Implementing Web Applications*. Springer-Verlag London Limited, 2008.

[RS08] G. Rossi and D. Schwabe. Modeling and implementing web applications with OOHDM. In Rossi et al. [RPSO08], pages 109–155.

[RSL99] G. Rossi, D. Schwabe, and F. Lyardet. Web application models are more than conceptual models. In P.P. Chen, D.W. Embley, J. Kouloumdjian, S.W. Liddle, and J.F. Roddick, editors, *ER (Workshops)*, volume 1727 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 1999.

[SD05] H.A. Schmid and O. Donnerhak. OOHDMDA – an MDA approach for OOHDM. In *Web Engineering, 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005, Proceedings*, volume 3579 of *Lecture Notes in Computer Science*, pages 569–574. Springer, 2005.

[Sei03] E. Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, Sept.–Oct. 2003.

[Sel03] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, Sept.–Oct. 2003.

[Sel08] B. Selic. MDA manifestations. *The European Journal for the Informatics Professional*, IX(2):12–16, April 2008. http://www.upgrade-cepis.org.

[Sen75] M.E. Senko. Information systems records, relations, sets, entities, and things. *Information Systems*, 1(1):3–13, 1975.

[SG03] R. Soley and OMG Staff Strategy Group. Model driven architecture. http://www.omg.org/cgi-bin/doc?omg/00-11-05, 2003.

[SK97] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110–111, Apr 1997.

[SK03] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, Sept.–Oct. 2003.

[SK06] W. Schwinger and N. Koch. Modeling web applications. In G. Kappel, B. Prýýl, S. Reich, and W. Retschitzegger, editors, *Web Engineering: The Discipline of Systematic Development of Web Applications*, pages 39–64. John Wiley & Sons, Hoboken, New Jersey, 2006.

[SM88] S. Shlaer and S.J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice-Hall, Upper Saddle River, New Jersey, 1988.

[Spr04] J. Sprinkle. Model-integrated computing. *IEEE Potentials*, 23(1):28–30, Feb.–March 2004.

[SR95a] D. Schwabe and G. Rossi. Building hypermedia applications as navigational views of information models. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences (HICSS'95) Vol. III*, volume 3, pages 231–240, 1995.

[SR95b] D. Schwabe and G. Rossi. The object-oriented hypermedia design model. *Communications of the ACM*, 38(8):45–46, 1995.

[SRS+08] W. Schwinger, W. Retschitzegger, A. Schauerhuber, G. Kappel, M. Wimmer, B. Pröll, C. Cachero, S. Casteleyn, O. De Troyer, P. Fraternali, I. Garrigós, F. Garzotto, A. Ginige, G.-J. Houben, N. Koch, N. Moreno, O. Pastor, P. Paolini, V. Pelechano, G. Rossi, D. Schwabe, M. Tisi, A. Vallecillo,

K. van der Sluijs, and G. Zhang. A survey on web modeling approaches for ubiquitous web applications. *IJWIS*, 4(3):234–305, 2008.

[SS09] Y. Singh and M. Sood. Model driven architecture: A perspective. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1644–1652, June–July 2009.

[SVC06] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[Szt01] J. Sztipanovits. Advances in model-integrated computing. In *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*, volume 3, pages 1660–1664, 2001.

[TCP08] O. De Troyer, S. Casteleyn, and P. Plessers. WSDM: Web semantics design method. In Rossi et al. [RPSO08], pages 303–351.

[Tho04] D. Thomas. MDA: revenge of the modelers or UML utopia? *IEEE Software*, 21(3):15–17, May–June 2004.

[TL82] D. Tzichritzis and F.H. Lochovski. *Data Models*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[TL98] O. De Troyer and C.J. Leune. WSDM: A user centered design method for web sites. *Computer Networks*, 30:85–94, 1998.

[TYF86] T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, June 1986.

[Uhl03] A. Uhl. Model driven arcitecture is ready for prime time. *IEEE Software*, 20(5):70, 72, Sept.–Oct. 2003.

[Uhl08] A. Uhl. Model-driven development in the enterprise. *IEEE Software*, 25(1):46–49, 2008.

[UML09a] OMG Unified Modeling Language (OMG UML), infrastructure: Version 2.2. http://www.omg.org/spec/UML/2.2/, Feb. 2009. Object Management Group.

[UML09b] OMG Unified Modeling Language (OMG UML), superstructure: Version 2.2. http://www.omg.org/spec/UML/2.2/, Feb. 2009. Object Management Group.

[UWE] UWE web site. http://uwe.pst.ifi.lmu.de.

[VFHB03] R. Vdovjak, F. Frasincar, G.-J. Houben, and P. Barna. Engineering semantic web information systems in Hera. *Journal of Web Engineering*, 2(1–2):3–26, 2003.

[VKC+07] A. Vallecillo, N. Koch, C. Cachero, S. Comai, P. Fraternali, I. Garrigós, J. Gómez, G. Kappel, A. Knapp, M. Matera, S. Meliá, N. Moreno, B. Pröll, T. Reiter, W. Retschitzegger, J. Eduardo Rivera, A. Schauerhuber, W. Schwinger, M. Wimmer, and G. Zhang. MDWEnet: A practical approach to achieving interoperability of model-driven web engineering methods. In Koch et al. [KVH07].

[VnM04] B. Vela, C.J. Acu na, and E. Marcos. A model driven approach to XML database development. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)*, pages 273–285, Shanghai, China, November 2004.

[VW] Visualwade. http://www.visualwade.com.

[WKC+03] I. Wilkie, A. King, M. Clarke, C. Weaver, C. Raistrick, and P. Francis. *UML ASL Reference Guide: ASL Language Level 2.5*. Kennedy Carter Ltd., 2003.

[WR] Webratio. http://www.webratio.com.

[WSSK07] M. Wimmer, A. Schauerhuber, W. Schwinger, and H. Kargl. On the integration of web modeling languages: Preliminary results and future challenges. In Koch et al. [KVH07].