1
Introduction

This chapter is from the new book Patterns of Data Modeling *by Michael Blaha, CRC Press, 2010. All rights reserved. This chapter is posted on odbms.org with the permission of the author and publisher. This is the first of three chapters that will be posted over the upcoming months.*

Models provide the means for building quality software in a predictable manner. Models let developers think deeply about software and cope with large size and complexity. Developers can think abstractly before becoming enmeshed in the details of writing code. Although models are beneficial, they can be difficult to construct. That is where patterns come in. Patterns provide building blocks that help developers construct models faster and better.

This chapter starts with a discussion of models and then introduces the topic of patterns.

1.1 What Is a Model?

A ***model*** is an abstraction of some aspect of a problem. Most software models are expressed as graphical diagrams and by their form appeal to human intuition. Developers must understand a problem before attempting a solution. Models let developers express their understanding of a problem and communicate it to others — technologists, business experts, users, managers, and other project stakeholders. Developers can focus on the essence of an application and defer implementation details. Chapter 8 of [Blaha-2001] lists additional reasons for building models.

There are various kinds of models (such as data models, state-transition models, and data-flow models) that are used for databases, programming, and other purposes. This book concerns data models and the focus is on databases.

1.2 Modeling Notation

Data modeling has no widely-accepted, dominant notation. To appeal to a broad audience, this book uses two notations—UML (Unified Modeling Language) and IDEF1X—and presents most patterns and models with both notations. These notations largely express the same content, so you can readily understand this book as long as you are fluent with either one. The Appendix summarizes both notations and gives references for further details.

1.2.1 UML

The UML's data structure notation specifies entities and their relationships. This sets the scope and level of abstraction for subsequent development. The UML encompasses about a dozen notations of which one (the *class model*) concerns data structure.

The UML data structure notation derives from the original Chen notation [Chen-1976]. The Chen notation and its derivatives have been influential in the database community, but there are many dialects and a lack of consensus. This UML data structure model is just another Chen dialect, but one that has the backing of a standard. The Object Management Group has been actively working towards standardizing all of the UML notations.

The UML is normally used in conjunction with object-oriented jargon which I avoid. Object-oriented jargon connotes programming which I do not intend. This book's focus is on data modeling.

1.2.2 IDEF1X

The IDEF1X notation [Bruce-1992] specifies tables, keys, and indexes. IDEF1X also is a standard language and has been in use for many years. IDEF1X is closer to database design than the UML and more clearly shows the details of patterns.

1.2.3 Using Both Notations

In my consulting practice, I use both notations. I start with the UML to conceive the abstractions of an application. Then I translate the ideas into IDEF1X and add database details. From IDEF1X I generate database code. The UML is good for abstract modeling and not for database design. IDEF1X is good for database design and not for abstract modeling. Both notations are useful, but each has its place.

1.3 What Is a Pattern?

This book defines a ***pattern*** as a model fragment that is profound and recurring. A pattern is a proven solution to a specified problem that has stood the test of time. Here are some other definitions from the literature.

- A pattern solves a problem in a context. [Alexander-1979]
- "A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution." [Buschmann-1996]
- "A pattern is a template. It's a template that is an example worthy of emulation, and something observed from things in actuality. It's a template to a solution, not a solution. It's a template that has supporting guidelines (not so much that it keeps one from seeing how it might be used in novel ways)." [Coad-1994]

©Michael Blaha and CRC Press

- “A pattern is a template of interacting objects, one that may be used again and again by analogy.” [Coad-1995]
- “A pattern ... provides a proven solution to a common problem individually documented in a consistent format and usually as part of a larger collection.” [Erl-2009]
- “A pattern is an idea that has been useful in one practical context and will probably be useful in others.” [Fowler-1997]
- “A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem... It describes the problem, the solution, when to apply the solution, and its consequences.” [Gamma-1995]
- “A pattern describes a problem to be solved, a solution, and the context in which that solution works. It names a technique and describes its costs and benefits. Developers who share a set of patterns have a common vocabulary for describing their designs, and also a way of making design trade-offs explicit. Patterns are supposed to describe recurring solutions that have stood the test of time.” [Johnson-1997]
- “A pattern is a recurring solution to a standard problem... Patterns have a context in which they apply.” [Schmidt-1996]
- A pattern is “a form or model proposed for imitation.” [Webster’s dictionary]
- “In general, a pattern is a problem-solution pair in a given context. A pattern does not only document ‘how’ a solution solves a problem but also ‘why’ it is solved, i.e. the rationale behind this particular solution.” [Zdun-2005]

Since this book is about data modeling, the patterns focus on data structure (entities and relationships). I de-emphasize attributes as they provide fine details that can vary for applications.

1.4 Why Are Patterns Important?

Patterns have many benefits.

- **Enriched modeling language**. Patterns extend a modeling language—you need not think only in terms of primitives; you can also think in terms of frequent combinations. Patterns provide a higher level of building blocks than modeling primitives. Patterns are prototypical model fragments that distill the knowledge of experts.
- **Improved documentation**. Patterns offer standard forms that improve modeling uniformity. When you use patterns, you tap into a language that is familiar to other developers. Patterns pull concepts out of the heads of experts and explicitly represent them. Development decisions and rationale are made apparent.
- **Reduce modeling difficulty**. Many developers find modeling difficult because of the intrinsic abstraction. Patterns are all about abstraction and give developers a better place to start. Patterns identify common problems and present alternative solutions along with their trade-offs.

©Michael Blaha and CRC Press

- **Faster modeling**. With patterns developers do not have to create everything from scratch. Developers can build on the accomplishments of others.
- **Better models**. Patterns reduce mistakes and rework. Each pattern has been carefully considered and has already been applied to problems. Consequently, a pattern is more likely to be correct and robust than an untested, custom solution.
- **Reuse**. You can achieve reuse by using existing patterns, rather than reinventing solutions. Patterns provide a means for capturing and transmitting development insights so that they can be improved on and used again.

1.5 Drawbacks of Patterns

Even though patterns have much to offer, they are not a panacea for the difficulties of software development.

- **Sporadic coverage**. You cannot build a model by merely combining patterns. Typically you will use only a few patterns, but they often embody core insights.
- **Pattern discovery**. It can be difficult to find a pertinent pattern, especially if the idea in your mind is ill-formed. Nevertheless, this difficulty does not detract from the benefits when you find a suitable pattern.
- **Complexity**. Patterns are an advanced topic and can be difficult to fully understand.
- **Inconsistencies**. There has been a real effort in the literature to cross reference other work and build on it. However, inconsistencies still happen.
- **Immature technology**. The patterns literature is active but the field is still evolving.

1.6 Pattern vs. Seed Model

It is important to differentiate between patterns and seed models. The programming pattern books, such as [Gamma-1995], have true patterns that are abstract and stand apart from any particular application. In contrast most of the database literature ([Arlow-2004], [Fowler-1997], [Hay-1996], [Silverston-2001a,b]) confuses patterns with seed models.

A ***seed model*** is a model that is specific to a problem domain. A seed model provides a starting point for applications from its problem domain. Seed models are valuable in that they can save work, reduce errors, contribute deep insights, and accelerate development.

Thus seed models are truly useful. But if you are working in a different problem domain, you must first find the relevant seed models, understand the seed models, extract the implicit patterns, and then apply the patterns to your own application. In contrast, this book makes patterns explicit so that they are ready to go for any problem domain. Table 1.1 contrasts patterns with seed models.

©Michael Blaha and CRC Press

Table 1.1 Pattern vs. Seed Model

Characteristic	Data modeling pattern	Seed model
Applicability	Application independent	Application dependent
Scope	An excerpt of an application	Intended to be the starting point for an application
Model size	Typically few concepts and relationships (< 10)	Typically 10-50 concepts and relationships
Abstraction	More abstract	Less abstract
Model type	Can be described with a data model	Can be described with a data model

Note: A pattern is abstract and stands apart from any particular application, unlike a seed model.

1.7 Aspects of Pattern Technology

My usage of the term pattern is different than the literature but consistent with the spirit of past work. I treat pattern as an overarching term encompassing mathematical templates, antipatterns, archetypes, identity, and canonical models.

- ***Mathematical template***: an abstract model fragment that is useful for a variety of applications. A mathematical template is devoid of application content. Mathematical templates are driven by deep data structures that often arise in database models. Most templates have a basis in topology and graph theory, both of which are branches of mathematics.

  Mathematical templates have parameters that are placeholders. The parameters must be instantiated for each use. I use the notation of angle brackets to denote template parameters. You incorporate a mathematical template into a model by substituting application concepts for the parameters.
- ***Antipattern***: a characterization of a common software flaw. An antipattern shows what not to do and how to fix it. The literature emphasizes antipatterns for programming but they also apply to databases.
- ***Archetype***: a deep concept that is prominent and cuts across problem domains. This book's archetype models are small and focus on core concepts. [Arlow-2004] nicely articulates the idea of an archetype, but their models are so large that they are really more like seed models. A small model is more likely to be application independent and widely reusable than a large model.
- ***Identity***: the means for denoting individual entities, so that they can be found. There are different aspects of identity that deeply affect application models.

©Michael Blaha and CRC Press

- ***Canonical model***: a submodel that provides a useful service for many applications. A canonical model is an abstract service that is not bound to a particular problem domain in contrast to a seed model.

1.8 Chapter Summary

Models are the key to successful software development. Models help you think deeply, focus on the important issues, reconcile your thoughts with others, and ultimately build the software. Models enable you to achieve the conceptual integrity that is essential to quality development.

A pattern is a model fragment that is profound and recurring. A pattern is a proven solution to a specified problem that has stood the test of time. Patterns are important in that they help developers build models better and faster, which leads to building software better and faster. The fundamental treatment of patterns in this book contrasts with the application-specific seed models that dominate the database literature. In this book pattern is an overarching term that encompasses mathematical templates, antipatterns, archetypes, identity, and canonical models.

Bibliographic Notes

[Hoberman-2009] has an especially lucid explanation of modeling. [Blaha-1998], [Elmasri-2006], [Hernandez-2003], and [Teorey-2006] present processes for data modeling and database design.

[Alexander-1979] discusses patterns for the architecture of buildings. Alexander's book has been highly influential in the software patterns community and helped to initiate the field.

Most of the software patterns literature concerns design and implementation patterns. [Gamma-1995] is a classic book that focuses on patterns for programming design. [Buschmann-1996] discusses architectural patterns, design patterns, and idioms.

[Arlow-2004], [Coad-1992], [Fowler-1997], [Hay-1996], and [Silverston-2001a,b] address data modeling patterns. I regard many of their "patterns"—with no criticism intended—to be seeds for applications rather than true patterns. Ironically, each of the authors use a different notation. [Silverston-2009] is more abstract and closer in spirit to this new book.

[Coad-1992] has an excellent discussion of patterns in other disciplines. [Coad-1995] is really an object-oriented methodology book. It does not discuss deep data modeling patterns.

[Appleton] is a helpful Web site about patterns. [Backlund-2001] has a nice overview of the pattern technology available at that time. [Brown-1998] and [Laplante-2006] are good references for antipatterns.

©Michael Blaha and CRC Press

References

[Alexander-1979] Christopher Alexander. *The Timeless Way of Building*. New York: Oxford University Press, 1979.

[Appleton] http://www.cmcrossroads.com/bradapp/docs/patterns-nutshell.html#Patterns_What

[Arlow-2004] Jim Arlow and Ila Neustadt. *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML*. Boston, Massachusetts: Addison-Wesley, 2004.

[Backlund-2001] Per Backlund. *The Use of Patterns in Information System Engineering*. M.Sc. dissertation, University of Skövde, Sweden, 2001.

[Blaha-1998] Michael Blaha and William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Upper Saddle River, New Jersey: Prentice-Hall, 1998.

[Blaha-2001] Michael Blaha. *A Manager's Guide to Database Technology*. Upper Saddle River, New Jersey: Prentice-Hall, 2001.

[Brown-1998] William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Ltd, 1998.

[Bruce-1992] Thomas A. Bruce. *Designing Quality Databases with IDEF1X Information Models*. New York: Dorset House, 1992.

[Buschmann-1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, United Kingdom: Wiley, 1996.

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

[Coad-1992] Peter Coad. Object-oriented patterns. *Communications ACM 35*, 9 (September 1992), 152–159.

[Coad-1994] Peter Coad and Mark Mayfield. Object Model Patterns Workshop Report. *ACM OOPSLA Conference*, October 23–27, 1994, Portland, Oregon, 102–104.

[Coad-1995] Peter Coad, David North, and Mark Mayfield. *Object Models: Strategies, Patterns, and Applications*. Upper Saddle River, New Jersey: Yourdon Press, 1995.

[Elmasri-2006] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems (5th Edition)*. Boston: Addison-Wesley, 2006.

[Erl-2009] Thomas Erl. *SOA Design Patterns*. Upper Saddle River, New Jersey: Prentice-Hall, 2009.

[Fowler-1997] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Boston, Massachusetts: Addison-Wesley, 1997.

[Gamma-1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, Massachusetts: Addison-Wesley, 1995.

[Hay-1996] David C. Hay. *Data Model Patterns: Conventions of Thought*. New York, New York: Dorset House, 1996.

[Hernandez-2003] Michael J. Hernandez. *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design (2nd Edition)*. Boston: Addison-Wesley, 2003.

[Hoberman-2009] Steve Hoberman. *Data Modeling Made Simple, 2nd edition*. Bradley Beach, New Jersey: Technics Publications, 2009.

[Johnson-1997] Ralph E. Johnson. Frameworks = (components+patterns). *Communications ACM 40*, 10 (October 1997), 39–42.

©Michael Blaha and CRC Press

[Laplante-2006] Phillip A. Laplante and Colin J. Neill. *Antipatterns: Identification, Refactoring, and Management*. Boca Raton, FL: Auerbach Publications, 2006.

[Schmidt-1996] Douglas C. Schmidt, Mohamed Fayad, and Ralph E. Johnson. Software patterns. *Communications ACM 39*, 10 (October 1996), 36–39.

[Silverston-2001a] Len Silverston. The *Data Model Resource Book, Volume 1*. New York, New York: Wiley, 2001.

[Silverston-2001b] Len Silverston. The *Data Model Resource Book, Volume 2*. New York, New York: Wiley, 2001.

[Silverston-2009] Len Silverston and Paul Agnew. *The Data Model Resource Book, Volume 3*. New York, New York: Wiley, 2009.

[Teorey-2006] Toby Teorey, Sam Lightstone, and Tom Nadeau. *Database Modeling and Design (4th Edition)*. New York: Morgan Kaufmann, 2006.

[Zdun-2005] Uwe Zdun and Paris Avgeriou. Modeling Architectural Patterns Using Architectural Primitives. *ACM OOPSLA Conference*, October 16–20, 2005, San Diego, California, 133–146.

©Michael Blaha and CRC Press