Chapter 3. Models, Patterns, and Reuse

Robert J. Glushko and Tim McGrath

3.0 Introduction 1
3.1 Models 2
3.2 Adapting the Classical Modeling Approach to Documents 3
3.2.1 External Views: Instances of Document Implementations 5
3.2.2 Physical Views: Document Implementation Models (or Schemas) 5
3.2.3 Conceptual Views: Document Component and Assembly Models 8
3.3 The Model Matrix 13
3.3.1 Metadata and Metamodels 15
3.3.2 Metamodels for Processes 16
3.4 Patterns 17
3.4.1 Patterns in Business 18
3.4.2 Why Businesses Follow Patterns 19
3.4.3 Finding Patterns in the Model Matrix 20
3.4.4 Using the Model Matrix as a Framework 22
3.4.5 Process and Documents: Yin and Yang 23
3.5 Key Points in Chapter 3 25
3.6 Notes 26

3.0 Introduction

The GMBooks.com hypothetical Internet bookstore in Chapter 1 illustrates two important themes of Document Engineering, the idea of document exchange and the reuse of patterns of document exchange to implement or adapt a business model. In Chapter 4, "Describing What Businesses Do," we begin to make a systematic survey of the kinds of models and patterns that are reused in Document Engineering. Just like every other engineering discipline, Document Engineering emphasizes the reuse of existing specifications or standards that work. Doing so reduces costs and risks while increasing the reliability and interoperability of the deployed solution.

Document Engineering emphasizes the reuse of existing specifications or standards

We'll begin this chapter with our own definitions of *model, pattern,* and other words that are important in Document Engineering but are overused because they are important to lots of other domains as well. In spite of their overuse, we need these terms to describe what businesses do from a variety of perspectives.

We follow the classical modeling approach in distinguishing three levels of abstraction. The least abstract models, called *external models,* describe specific implementations of

business documents, processes, or other artifacts. *Physical models* are more general because they describe a set or class of instances, but they still capture the technology in which the instances were implemented. *Conceptual models* remove the implementation technology to emphasize the semantic relationships that define some class of instances.

We can also distinguish what businesses do according to the depth or granularity with which we describe the business relationships in each model. From the organizational or business-to-business perspective, patterns are coarse with just the most important roles and relationships visible. At the process level, more details of the relationship are visible, and we begin to see the documents that are exchanged to carry out each process. The information level is the most granular perspective, and we can see specific information components within the document models.

These two dimensions of model abstraction and model granularity let us define a model matrix that shows in a single diagram the relationships among business model, business process, and business information models at both conceptual and physical levels. This gives us a framework for discussing the most important and reusable patterns and for explaining how the most granular patterns for business information and business processes are composed and choreographed to create more complex patterns of greater scope.

3.1 Models

The business, organizational, and technological structures and relationships within and between enterprises can be extremely complex, which is why we need models to describe them. Models are simplified descriptions of a subject that remove some of its complexity to emphasize certain features or characteristics and deemphasize others.

Models are simplified descriptions of a subject

Of course there are always differences between the subject being modeled and the model, or else the model serves no purpose. Thus much of the skill of modeling involves knowing what to ignore—if you look at every single tree you never see the forest.

When there is a problem to be solved within a subject, analysts study the subject and ask experts questions about it. The information they gather is embodied in models that record and communicate the issues and constraints of the subject. These models help the

analysts, domain experts, and designers understand the existing situation and devise appropriate changes.

In Document Engineering we develop models that emphasize document requirements and patterns of information exchange. We use these models to analyze, communicate, and design the formal definitions of business processes and the documents that are exchanged to carry them out.

In Document Engineering we develop models that emphasize document requirements and patterns of information exchange

We can express a model with many different notations, each of which is effective for some purpose or audience. Simple line and box drawings on whiteboards or the back of an envelope can depict the most important constructs in a model and their relationships to each other. At other times more verbose and narrative descriptions or expressions in formal language may be necessary to represent the important details of a model.

In this book we depict our models in various ways, often using some of the conventional notations of the Unified Modeling Language (UML).[1] But using the UML is not in itself modeling. Nor do you need to use the UML to do Document Engineering. We can use any modeling notation. What matters is that the notation must capture the necessary metadata needed to define components and promote their reuse. In this text we will use the UML to describe business processes, collaborations and transactions with UML Activity and Sequence Diagrams (chapters 9 and 10) and document components and structures using UML Class and Dependency Diagrams (chapters 13 and 14).

3.2 Adapting the Classical Modeling Approach to Documents

The approach and terminology we use for modeling in Document Engineering is a document-centric adaptation of the classical three-level modeling approaches and architecture[2] depicted in Figure 3-1. This approach distinguishes between external representations that describe specific things, artifacts, or instances in the world, physical (or internal) views that present different models of instances in some technology, and conceptual views or models that abstract those descriptions from any particular implementation.

Whenever we analyze documents we can't avoid dealing with the processes that create and use them, but it is easier to introduce Document Engineering concepts and methods if we discuss documents and processes separately whenever we can. So we will also be analyzing business processes and creating models that describe them, but for the remainder of section 3.2 we will focus on document models and defer business process models until section 3.3.2.

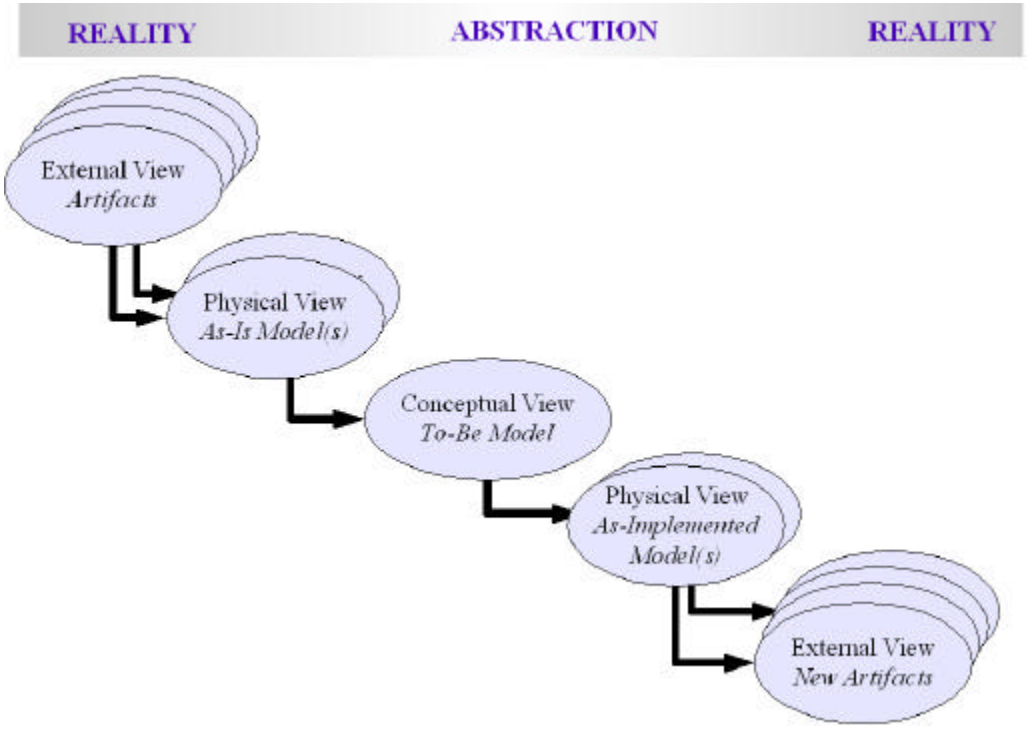


Figure 3-1. The Classical Modeling Approach

The first phase of applying the classical modeling approach to documents is to find and analyze real-world artifacts and represent the results in a model that describes their physical implementation. We then analyze these artifacts to create what are often called the As-Is models.

For business processes, the As-Is models are transformed into To-Be *business process models* by selecting and adapting patterns appropriate for the required context of use.

For documents we call the As-Is model a *document component model* and the To-Be models are called *document assembly models.*

Finally we bring the conceptual view back to a physical view by expressing it in technology appropriate for the contexts in which it will be used. These new *document*

implementation model and *process implementation models* are the As-Implemented models.

3.2.1 External Views: Instances of Document Implementations

When documents exist in printed or tangible form, like Halfat's tax receipt on a pottery fragment (see chapter 1), it is easy to think of them as artifacts in the world that could be described by external models. But what is most important about a document is the information or intangible content it contains, so the document should be considered an external model of the thing it describes. This is especially true in the domain of Document Engineering where our emphasis is on the documents exchanged by business processes. We analyze inventory reports, not the actual goods stored in a warehouse, for example, or purchase orders rather than the specific goods being ordered.

A document can be considered an external model of the thing it describes

If we are designing a new business process and no documents currently exist, we must identify information and process requirements by talking to people, sources that are even less directly coupled to the things in the world than documents are. So rather than treat descriptions of specific things in the world (or other information sources like printed or web forms) as models, we will treat them as the primary instances or artifacts that we analyze to create models. For example, Figure 3-2 is an XML document instance, which we'll refer to as Book.xml in the sections that follow. Figure 3-2 is not the book, "Moby Dick" by Herman Melville; it is an external view of it.

```
<?xml version="1.0" encoding="UTF-8"?>
<Book>
 <Title>Moby Dick</Title>
 <Author>Herman Melville</Author>
 <ISBN>0804900337</ISBN>
 <Publisher>Airmont</Publisher>
 </Book>
```

Figure 3-2. XML Document Instance (Book.xml)

3.2.2 Physical Views: Document Implementation Models (or Schemas)

After we analyze a number of document instances like Book.xml, we can represent our results in a model that describes them (the As-Implemented model in the classical

approach, or a document implementation model or document schema in Document Engineering).

In Chapter 2 we described the role of XML schemas in representing a document type as some bounded set of possible or desired XML documents. Figures 3-3a and 3-3b are XML schemas that validate Book.xml and other XML instances of the Book document type. The first is expressed as a DTD and the second is expressed using XSD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Book (Title, Author, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Figure 3-3a. Book.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
        <xs:element name="Book">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="Title" type="xs:string"/>
                                <xs:element name="Author" type="xs:string"/>
                                <xs:element name="ISBN" type="xs:string"/>
 <xs:element name="Publisher" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure 3-3b. Book.xsd

Figure 3-3. XML Schemas for Instances Like Book.xml

Document implementation models such as XML schemas can be very narrow, describing only the exact set of instances that were analyzed, or they can be more general, describing a wider variety of instances that are similar, but not identical, to those that were analyzed. In either case, the more instances we consider when we develop a document implementation model, the more likely we are to identify and capture the set of rules that govern the possible instances of the document type being modeled.

But document implementation models—defined as expressions of structure and integrity constraints on some set of information—are not limited to schemas for XML documents or even to markup languages. They can be expressed using many formal languages, including ISO 9735 for EDI and SQL/DDL for relational databases. There are also less formal ways of expressing As-Implementation models, such as the message implementation guides in narrative form that are often used to explain the structures of EDI document types. Figure 3-4 is a document implementation model that describes a database table in which to store information about book instances.

```
Create Table Book
{
Identifier CHAR(14) PRIMARY KEY,
BookTitle CHAR(50),
AuthorName CHAR(50),
Publisher      CHAR(20)
}
```

Figure 3-4. Database Schema for Instances Like Book.xml

As these three examples show, document implementation models are tightly bound to the technology of implementation, using constructs like XML elements or database fields so that computer programs can interpret the model. But this tight binding can prevent us from thinking beyond the specific implementation, especially if the models were created to describe or validate a limited set of examples.

For example, if we study the specific documents used by a business process and discover they have a variety of implementations, each with their own schema definition or notation, how can we compare them? Book.dtd, Book.xsd, and the Book database schema are expressed in widely different syntaxes that constrain our ability to understand the role of “Book” in our business applications.

Implementation models limit design and reuse capabilities

Document implementation models also limit design and reuse capabilities. If a bookstore wants to share information between its database and documents, as it does when responding to a customer query, neither the database schema nor the document schema is sufficient. We need to understand the relationship between them, which might be expressed as the *mapping* of one model to the other. But this is often not

straightforward. In our examples here, we might not know that *PCDATA* and string are synonyms and we can't be sure that the <ISBN> element in Book.xsd plays the same role as *Identifier* in the Book database schema.

Document implementation models will only tell us how their components are expressed in a particular technology. To understand how they relate to each other we also need to know what the components mean; that is, we need to understand their semantics.

3.2.3 Conceptual Views: Document Component and Assembly Models

We can best describe the semantics of documents using models of the concepts they contain. This conceptual view lets us distinguish one class of document from another. Conceptual views are independent of the physical implementation and are not tied to any particular technology.

Prose definitions are often adequate conceptual models for classifying documents. We might say that a typical dictionary is organized as a set of word entries, each of which consists of a main word, a pronunciation guide, and one or more definitions or senses. The entry may also have a derivation showing its roots in some classical language, other forms of the word, a list of synonyms or antonyms, quotations, or an illustration.

Likewise, we might say that a typical invoice contains information about goods or services provided by the seller and the amount and date of expected payment. It may also describe methods of delivery or other terms governing the transactions that occur. Dates, amounts, and postal codes are among the kinds of content in invoices for which expected values or ranges of values can be precisely specified.

But similar types of content occur in many documents, and the distinctions between these can be subtle, especially when there is overlap in information and structural patterns. Precisely what is it that differentiates a catalog from a brochure, a newspaper from a magazine, a dictionary from an encyclopedia, a calendar from a schedule, or a purchase order from an invoice? Obviously there are distinctions between them, or we wouldn't have different words to describe them and we couldn't reliably classify them as one type of document or another. How can we identify and communicate what distinguishes them?

In Document Engineering we introduce use two types of conceptual models for documents that are more formal and precise than prose definitions. The first is the document component model, which describes the complete set of semantic components in a domain, including their structure and their potential relationships. The document component model portrays the network of associations between the components, so rather than describing a single type of document, it implicitly describes many different types of documents. Such a conceptual model of information about books is shown in Figure 3-5 in the notation of a class diagram.

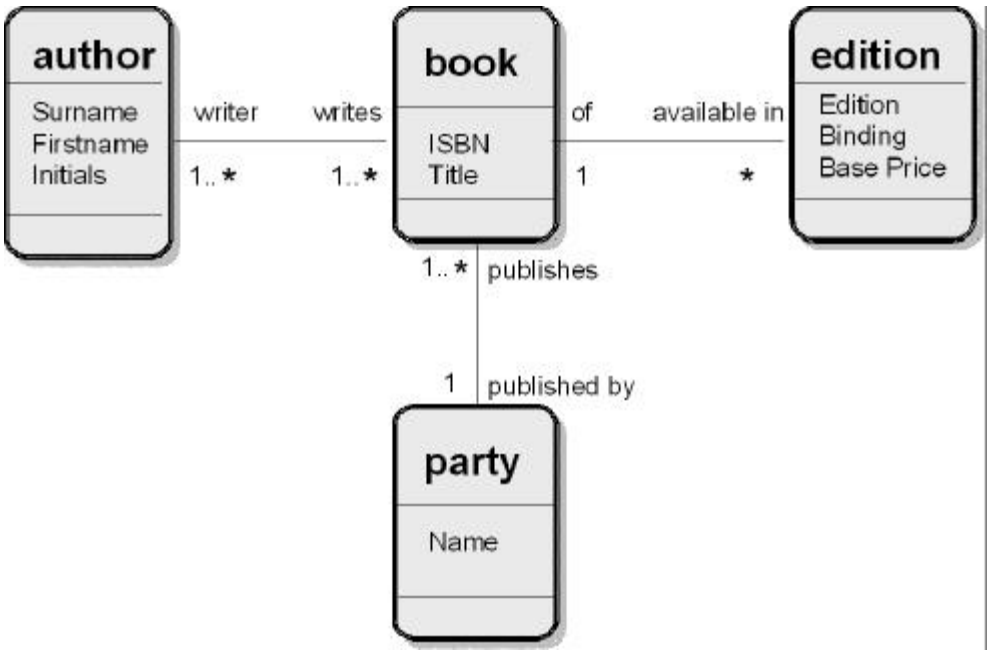


Figure 3-5. UML Class Diagram for Document Component Model of "Book"

Document component models can be represented as UML class diagrams. This notation graphically depicts object classes, their attributes, and their associations. The model also captures important rules about the relationships between classes.

For example, a Publisher can be considered a reuse of the object class called Party with a special association to Book that we label as Publishes in Figure 3-5. So the Publisher is modeled as a "Published by" Party. However, an Author is not a reuse of Party because the former has attributes that do not apply to the latter.

The model in Figure 3-5 also depicts the business rules that an author can write more than one book and that books can have more than one author. It also tells us that even if the book has more than one edition, it has only one ISBN.

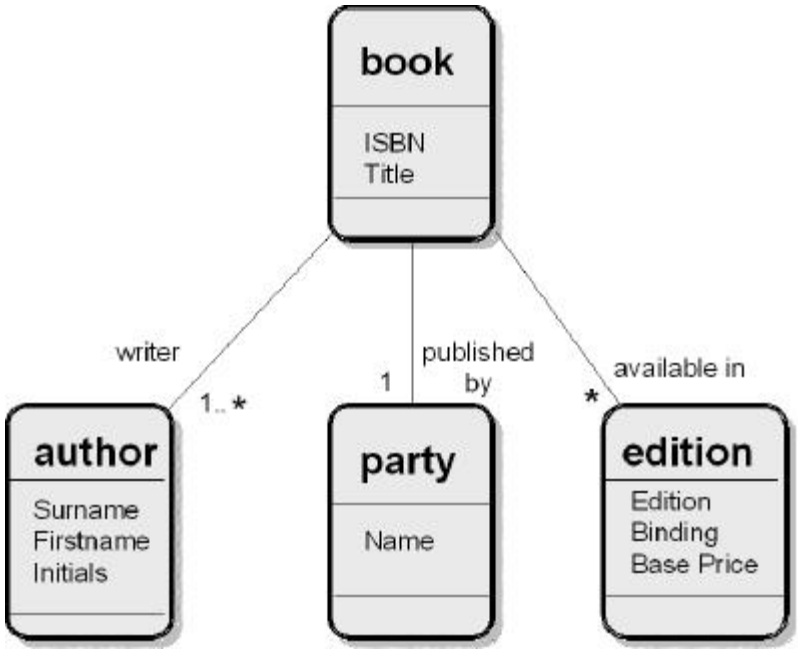
A document component model describes the complete set of semantic components in a domain

The Book concept could be implemented as an XML schema, database table, EDI message definition, or paper form. Each of these can represent concepts like Author and Title even though the implementation models may vary. But there is a crucial activity we must carry out before we can implement anything.

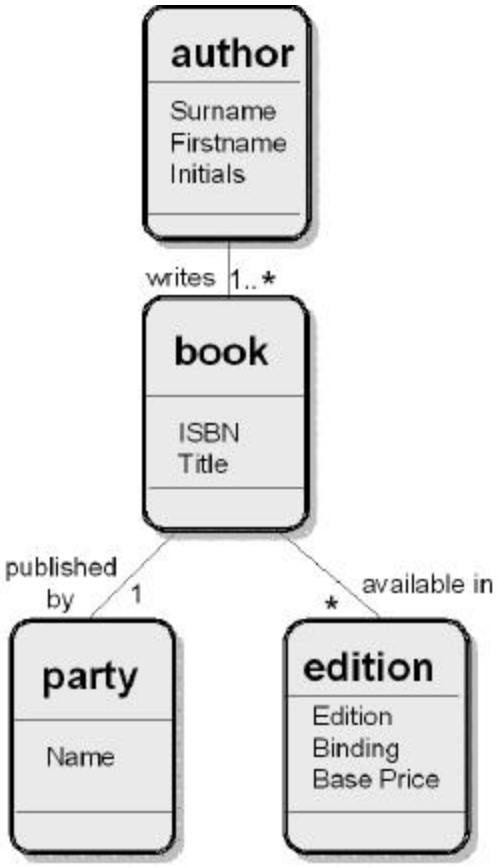
If we follow a path through the network of associations represented in a document component model, we are selecting a subset or arrangement of components to meet the information requirements of a specific context. A document assembly model describes the way in which the selected components are assembled into a hierarchical structure.

A document assembly model describes the way in which selected components are assembled into a hierarchical structure

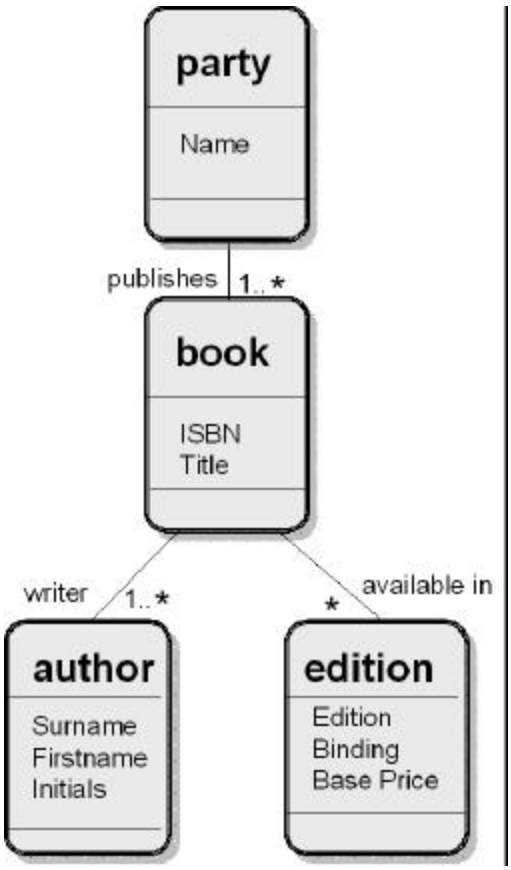
Document assembly models are best visualized as tree diagrams of hierarchical structures. For example, from the document component model in Figure 3-5, we could construct three different document assembly models by traversing the associations in different orders. The resulting hierarchical document types would organize the same semantic components in different structures to impose different interpretations or contexts that emphasize the book, the author, or the publisher. Three such document assembly models are shown in Figure 3-6.



3-6a. Document Assembly Model for Book Context



3-6b. Document Assembly Model for Author Context



3-6c. Document Assembly Model for Publisher Context

Figure 3-6. Alternative Document Assembly Models

Conceptual views of models like those shown in figures 3-5 and 3-6 are a better way to represent and communicate the results of analysis and design than the physical views in figures 3-3 and 3-4 because they are not constrained by any specific technology. This technology independence also makes them easier to manipulate or revise. Similarly, at the conceptual level it is easier to generalize a model to make it describe a larger set of possible or desired artifacts than the ones we happened to observe when we first created an implementation model. It is also easier to specialize at the conceptual level, for example, by deriving a related model that incorporates additional characteristics or relationships; in this case, we could express a conceptual view of a model for chemistry books based on the model for a book.

When technologies change, the optimal implementation model will also change even though the underlying conceptual models don't

When technologies change, the optimal implementation model will also change even though the underlying conceptual models don't. A fascinating example goes back 10,000 years to the last part of the Stone Age, when farmers in the Near East began to use clay pegs or tokens to keep track of farm products, often storing them in hollow clay balls.[3] The Neolithic accountants later realized that instead of enclosing the tokens inside a clay ball, they could simply make marks in the clay to represent the one-to-one correspondence between the clay tokens and the goods, ultimately leading to the invention of Cuneiform writing in the fourth millennium BCE. Today we use more modern technologies for tracking inventory, but the underlying conceptual model of counting is essentially the same. We'll further discuss the role of technology in the relationship between conceptual and physical models in Section 4.4 and Chapter 5.

3.3 The Model Matrix

Later in this book we will discuss in more detail how to analyze documents by creating physical and conceptual document models. We will also apply similar approaches to analyzing contexts and creating models for business relationships, processes, and other kinds of reusable patterns.

These two dimensions of model abstraction and model granularity form a matrix for organizing the analysis and modeling approaches in Document Engineering,[4] as shown in Figure 3-7.

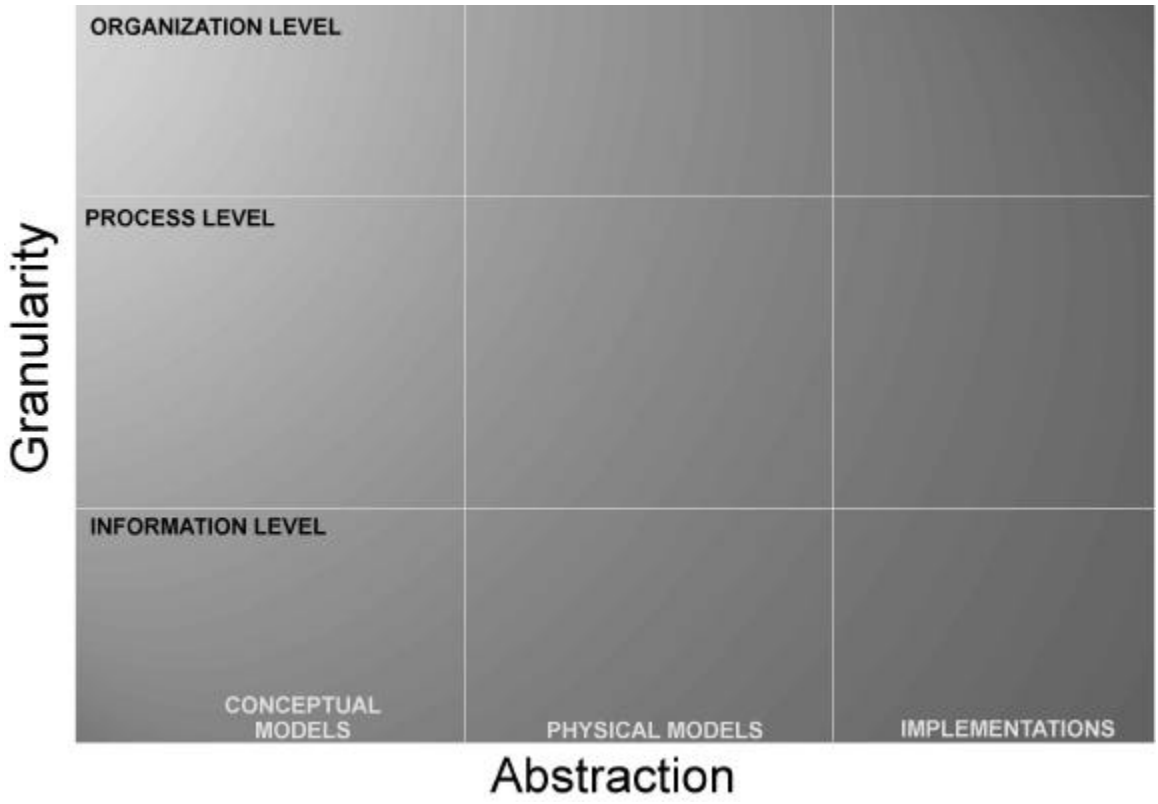


Figure 3-7. The Model Matrix

Let's begin by describing the matrix dimensions. From left to right is the abstraction dimension. The most abstract or context-free conceptual models are arranged on the left. Moving to the right implies more physical models, and finally specific implementations of actual documents or processes are the external models.

The two dimensions of model abstraction and granularity form the Document Engineering matrix

From top to bottom is the dimension of model granularity, on which we can depict the amount of detail with which we describe the business relationships in each model. From the organizational or business-to-business perspective, patterns show only the most important roles and relationships. At the process level more details about the relationship are visible, and we begin to see the documents that are exchanged to carry out each process. The information level is the most granular perspective, and we can see specific information components within the document models.

3.3.1 Metadata and Metamodels

Metadata might be the hardest term to define in the field of Document Engineering because its usual definition of "data about data" isn't much of a starting point. We are using the prefix meta to convey concern with the concepts and results of the discipline named in the suffix.[5] For example, a metalanguage is a language or system of symbols used to discuss another language or system, and a metatheory is a formal system that describes the structure of some other system. So metadata consists of data structures used to discuss other data structures.

Metadata augments the values of information (or data) with additional properties that explain its meaning, organization, cardinality, and other characteristics of interest in our models. What constitutes metadata is relative. Data may be metadata depending on your perspective. For example, statistics are data to some people and metadata to others.

What constitutes metadata is relative

In Document Engineering we use various pieces of metadata in our models so that we can define richer models and also compare and align different types of models. Remember, we're not talking about the content of the models; we're talking about the constructs by which the model content is organized or structured.

In XML the metadata are things like elements and attributes. In SQL the metadata includes tables and columns, and in the UML metadata includes features like classes, associations, and attributes.

To help us use metadata, we need a model, which we call the *metamodel*. A metamodel is a higher abstraction of a model, used to describe the type of information in a model. If we were to define a model of a document, the document model's metamodel might specify that the content of a document can be described using separate data objects, each of which has properties such as cardinality, definitions, conditional rules, and sets of legitimate values.[6] For example, the metamodel for Book.xsd is the specification for W3C Schema (XSD), which explains how these schemas (or implementation models) are constructed. By explaining what metadata is required and how they relate to each other, metamodels enable us to build consistent and robust models.

While the overall purpose of metadata may be similar in various types of models, because of their terminology, syntaxes, or different notations, it isn't always easy to

recognize the correspondences. For example, is an XML element equivalent to an SQL table? What is the relationship between elements and classes? So metamodels are also useful if we want to exchange or compare different models. If two models share the same metamodel, it is easier to compare and align the two.

A common metamodel helps align different models

With physical views of models, this is self-evident: two XML XSD schemas are easier to compare than an XML XSD schema and an XML DTD, or any XML schema and a database schema.

Models of conceptual views also have metadata and metamodels to describe them. For example, the ebXML Core Component Technical Specification[7] defines a metamodel for defining conceptual information models for document components. This means that even though there are many different types of business documents or document components, they can be described by using a common conceptual skeleton or scaffold.

3.3.2 Metamodels for Processes

Metamodels for business processes are especially important in Document Engineering because processes are inherently more abstract than documents, which readily exist as highly tangible implementations with a conventional notion of a document as a container or message with information components. In contrast, business processes can be described at many levels, and the lack of a predictable amount of detail for their constituents would make it less likely that any two process models could be meaningfully compared.

Business processes are inherently more abstract than documents

To deal with this fundamental modeling challenge, metamodels for describing business processes have evolved that distinguish multiple levels of abstraction along with the semantic properties that are necessary to define each level. We prefer the ebXML Business Process metamodel, which specifies three levels of abstraction: processes are defined in terms of *collaborations*, which are in turn described using *transactions.*[8] In addition, the ebXML Business Process Specification Schema[9] (BPSS) is designed to express a rich repertoire of patterns in a standard way, making it much easier to understand and compare business processes when they are described using the BPSS. We

use this metamodel for processes in Chapter 9, "Analyzing Business Processes," and Chapter 10, "Designing Business Processes with Patterns."

3.4 Patterns

Patterns are models that are sufficiently general, adaptable, and worthy of imitation that we can reuse them. A pattern must be general enough to apply to a meaningfully large set of possible instances or contexts. It must be adaptable because the instances or contexts to which it might apply will differ in details. And it must be worthy, that is the instances or contexts to which the pattern might apply should benefit from following it. Patterns are an important idea in many fields.

Patterns are models that are sufficiently general, adaptable, and worthy of imitation that we can reuse them

For example, the system of government called a parliament is a pattern used by numerous countries and states. In a parliamentary system, people democratically elect others to represent their interests, and the government is headed by a member of the political party with a majority of the elected representatives.

The Parliament model is an abstract, conceptual pattern because a country that adopts this model does not adopt any specific politicians and bureaucrats, just the pattern describing the ways in which its elected representatives are organized to govern. For a country to adopt a physical pattern for "parliament" it would have to invade another country and occupy its government buildings or kidnap its politicians (not that such events have never happened).

When patterns are implemented, they are often adjusted or customized to suit their particular context. Thus the parliamentary systems of government in the United Kingdom, Australia, and Japan are not identical, but they have many common features because each follows the same basic pattern.

Patterns are useful in every activity, from constructing houses to building software applications[10] to describing human behavior. Using patterns saves effort and yields more consistent, compatible, and successful designs. Indeed, sometimes a pattern is so consistently adopted it becomes an official or de facto standard (see section 5.7, "From Proprietary to Standard Models.")

Using patterns saves effort and yields more consistent, compatible, and successful designs

Document Engineering is mostly concerned with patterns of information exchange within and between enterprises and the patterns of semantic components in the documents being exchanged. But is it also useful to take even broader perspectives on what businesses do and the relationships between them because patterns at higher levels of abstraction set the context for more granular ones in which documents are specified.

3.4.1 Patterns in Business

Businesses exhibit a remarkable variety of behavior. Every business is different because they have different owners, employees, managers, and customers and because they operate in different industry, geopolitical, and regulatory contexts. The diversity of businesses can be seen easily in the yellow pages of a telephone directory or, more systematically, in the business classification codes designed to facilitate uniform collection and analysis of data about businesses. Examples of these formal categorizations include the 6-digit North American Industry Classification System (NAICS)[11] code and the UN/SPSC[12] coding system for products.

But just as they exhibit great variety, businesses exhibit great regularity in what they do and how they do it. At first glance, there doesn't seem to be much in common between "Computer Systems Design Services" (NAICS code 541512) and "Potato Farming" (NAICS code 111211) or "Bare Printed Circuit Board Manufacturing" (NAICS code 334412). But this diversity in classification belies the fact that most businesses do many things in similar ways.

Businesses exhibit both great variety and great regularity in what they do and how they do it

We call something a business or enterprise because it demonstrates some purposeful and organized activity to provide products or services, usually with a profit motive. But beyond this we can also agree that computer systems designers, potato farmers, and circuit board manufacturers all need to rent or buy, furnish, and insure their business locations, hire employees, procure and pay for supplies, market and sell their goods and services, fulfill orders, issue invoices, finance their operations, provide customer service, and so on.

Indeed, the fact that we have words like procure, pay, order, and invoice to describe common business processes and documents in an industry-neutral way confirms that there are general patterns in how business gets done.

3.4.2 Why Businesses Follow Patterns

Businesses in different industries also adopt patterns specific to their activities for a number of reasons:

- They may be affected by common laws or regulations. Local or national governments might require businesses to obtain permits, to ensure that their products and services meet health or safety standards, to pay taxes, and so on.
- They may follow similar trade practices and be affected by the same microeconomic factors, such as common suppliers or customers and similar opportunities or threats related to the introduction of new technologies or methods.
- They may be affected by common external forces imposed by the overall economic and financial environment such as tax and interest rates, levels of employment and education, and consumer confidence and other macroeconomic factors.
- They want to minimize the cost of hiring and training workers.

All of these influences encourage the adoption and use of patterns. Yet businesses don't follow patterns just because they are forced to do so by external influences. "Good business practice" is a dominant pattern and businesses also consciously strive to become more efficient and effective at what they do. Obvious examples of intentional patterns in business are those followed by franchises, where every business uses the same detailed operating methods and technology to get the benefits of aggregated purchasing, mass advertising, and data mining of composite transaction information to identify sales trends.

Businesses also need to operate in ways that are intelligible and acceptable to their trading partners or customers or else explain why they don't. Running a business according to the usual patterns and practices in an industry makes it easier for suppliers and customers to interact with it. A retailer that accepts only cash and doesn't allow

purchases to be returned will probably have to post warning signs at its checkout counter or website equivalent and will certainly have difficulty competing with more customer-friendly firms.

Reusing well-understood patterns makes businesses easier to start, manage, and improve. Adopting common patterns can reduce development and maintenance costs, improve performance, and enhance relationships with suppliers and customers. A business can more easily learn from others in its industry if it contributes to and follows industry best practices or *reference models*. The more systematic the practices in an industry, the more a business benefits from following them because of the network effects of standardization.

So businesses must balance two conflicting goals–to differentiate itself from its competitors and to run their business according to principles and methods used throughout their industry. Of course a business might decide not to follow the standard patterns in its industry. Perhaps it has the market dominance to impose its will on suppliers or customers or it hopes to create a competitive breakthrough by using a radically different technology or process. If it succeeds, the business will be creating a new pattern that others will soon try to adopt.

A business must balance how to differentiate itself and how to run according to industry practice

But for every business that invents a truly new business model or process there are many more who aspire only to get better at doing the things they are already doing. They do this by recognizing and adopting good business patterns.

3.4.3 Finding Patterns in the Model Matrix

Generic or abstract conceptual patterns become more specific or concrete by adding context. We will define *context* more completely in chapters 7 and 8. For now it is sufficient to say that contextualization means moving from left to right in the model matrix. Similarly, moving up the *granularity* axis in the Model Matrix gives a coarser granularity can suggest patterns that hides details and so might encourage new innovations.

So it follows that the best place to find reusable patterns in our models will be where they are generalized enough to be applicable across different implementation technologies but have enough context to be meaningful. And at the same time they must give a comprehensive view that is not so detailed it limits adaptations.

So to find useful patterns we navigate along the abstraction and granularity dimensions to confirm our analysis and understanding of the context of use. We present this metaphor graphically in Figure 3-8 using the "Pattern Compass." We'll more fully develop these ideas in Chapter 10, "Designing Business Processes with Patterns."

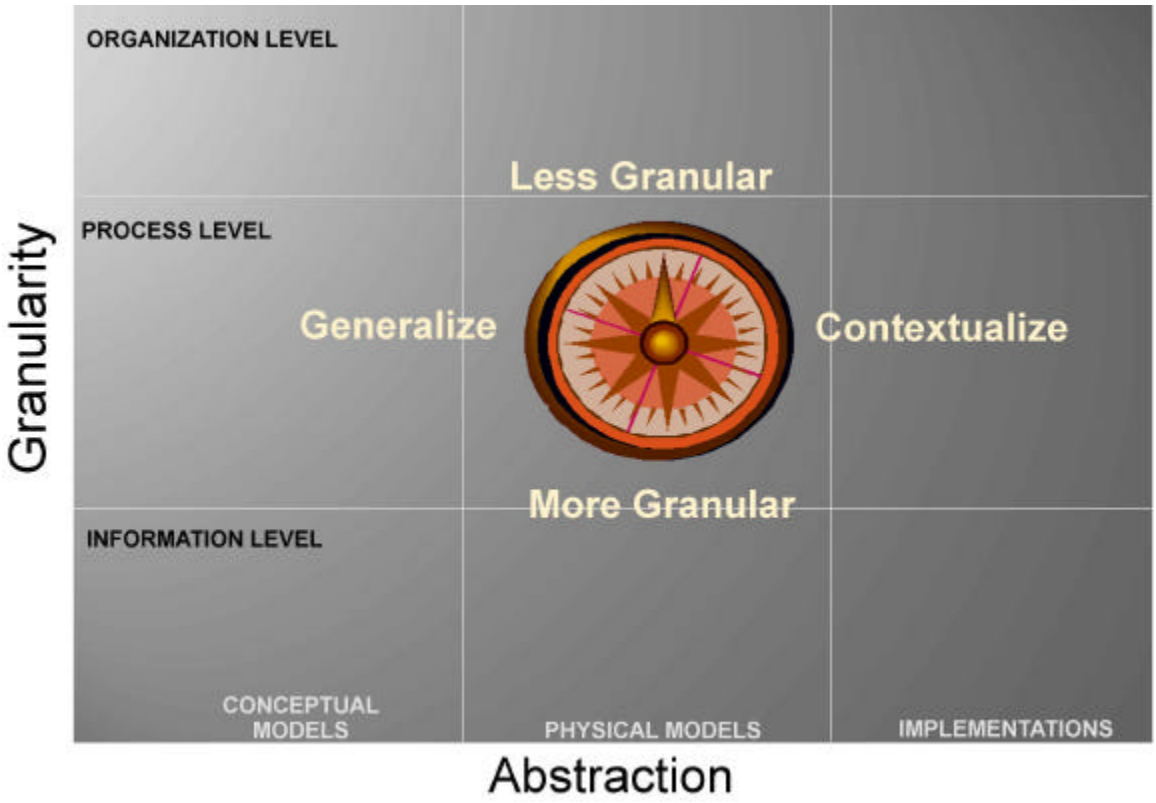


Figure 3-8. The Pattern Compass in the Model Matrix

The organization of patterns and models in the model matrix makes them easier to learn and reuse. As we examine other business relationships, processes and information from a Document Engineering perspective, the model matrix can provide a convenient framework.

Generic conceptual patterns become more specific by adding context

3.4.4 Using the Model Matrix as a Framework

A complete understanding of an enterprise's business relationships, the processes that carry them out, and information exchanged by those processes requires compatible and interconnected models of all three. This understanding is achieved when the strategic concerns embodied in organizational patterns that describe what a business wants to do can be linked by process patterns to information models that describe how to do it. In graphical terms, this convergence takes place when organizational and information models "meet in the middle" of the model matrix in process models.

So another use of the model matrix is as a roadmap to the analysis and design activities and methods that get us to its middle. Here the systematic differences in abstraction and granularity of these kinds of models in the matrix suggest that different kinds of modeling approaches are needed to create them.

A business analyst, document analyst, data analyst, and a task analyst will create different models

Different models emerge from the skills and tools of the business analyst, document analyst, data analyst, and task analyst. Each of these approaches looks at documents and processes differently, and while each of them is highly effective in some areas, they all have blind spots where their methods do not work well. We can overlay these different modeling perspectives on the model matrix in Figure 3-9.

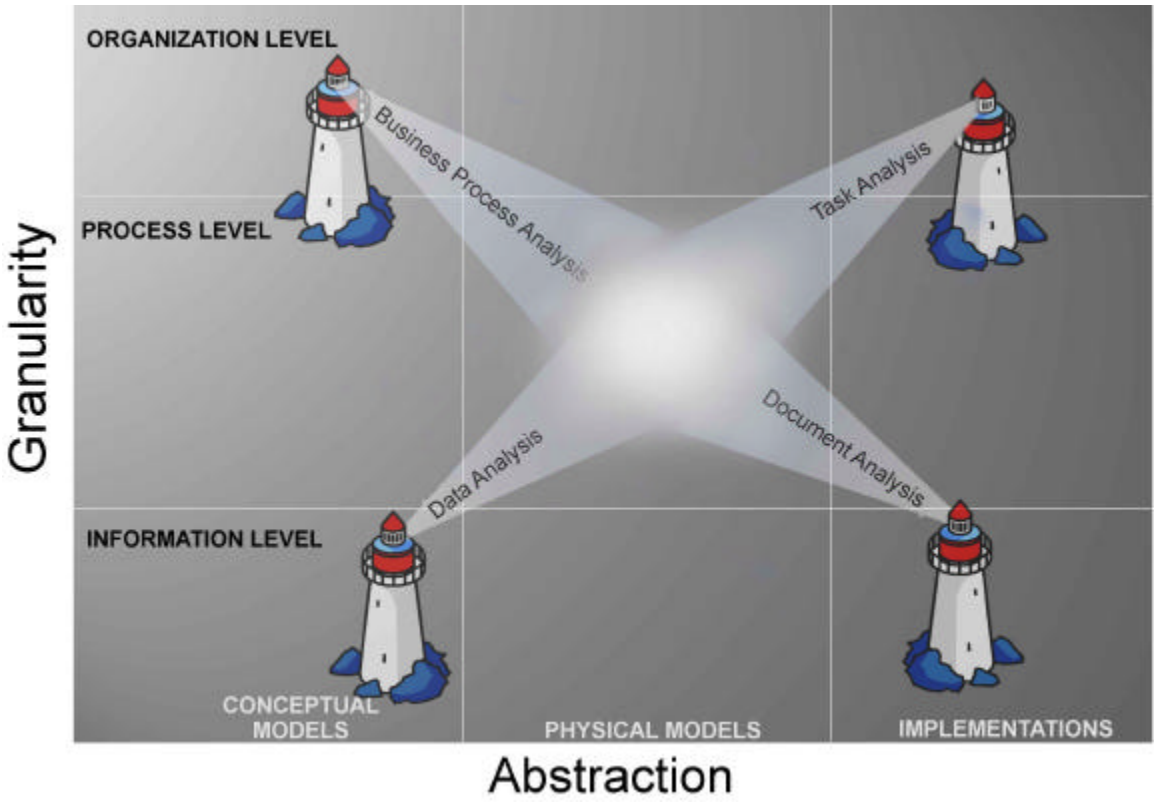


Figure 3-9. Converging Modeling Approaches in the Model Matrix

This depiction shows *business process analysis* focusing on the top left corner of the model matrix. This captures the conventional practice of process analysis in following a top-down approach to progressively refine abstract descriptions of what a business does. In contrast, *document analysis* techniques emphasize the study of instances of document artifacts, which are found in the lower right corner of the model matrix. Likewise, *data analysis* focuses on logical models of objects and associations, and *task analysis* focuses on the specific steps and information that users need to carry out a task. In Chapter 7 we introduce the Document Engineering Approach as a set of activities that follow a path through the model matrix, employing each of these modeling approaches in turn to yield models that "meet in the middle."

3.4.5 Process and Documents: Yin and Yang

Another important idea embodied in the model matrix is the essential and inescapable relationship between models of processes and models of documents. At the center of the model matrix, where processes are described as transactions and information exchange patterns, processes and documents are two perspectives on the same thing. Are process

patterns just combinations of information exchange patterns, or are information exchange patterns just the payload patterns in processes? The answer is yes to both questions. Business process and information patterns are the *yin and yang* of Document Engineering.

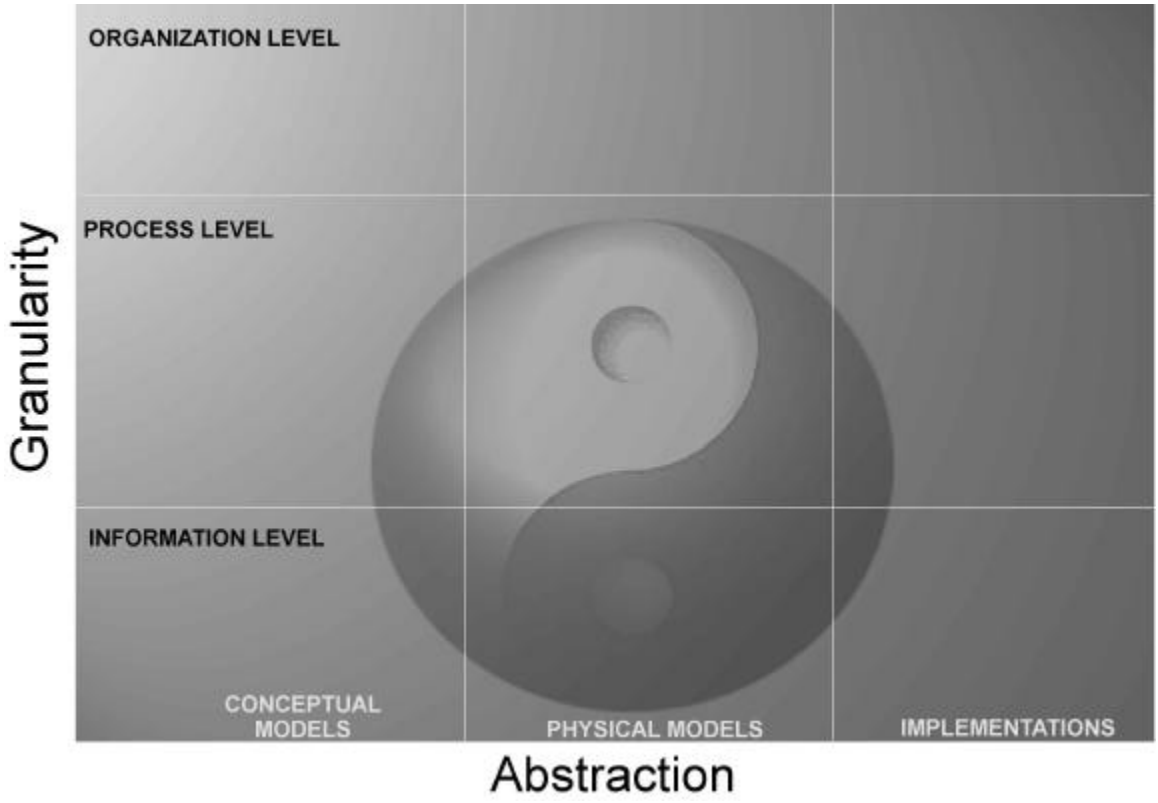


Figure 3-10. The Ying and Yang of Document Engineering

These central concepts of Chinese philosophy might seem out of place here, but they express perfectly the complementary and opposing relationships between business processes and documents. Processes produce and consume documents, which are a static snapshot or the tangible result of the process activity. Process descriptions emphasize business concerns and determine whether ways of doing business are compatible. Document descriptions emphasize technology concerns and determine whether business systems are compatible. We can separate processes and documents in our analysis, discussion, and models, but in the end they are always interconnected because both business and technical compatibility are necessary.

There are complementary and opposing relationships between processes and documents

In practical terms this means that models for processes and documents need to be developed with the same care and to compatible levels of detail. It explains why we need a Document Engineering approach that exploits complementary modeling approaches.

3.5 Key Points in Chapter 3

- Document Engineering emphasizes the reuse of existing specifications or standards.
- Models are simplified descriptions of a subject.
- In Document Engineering we develop models that emphasize document requirements and patterns of information exchange.
- A document can be considered an external model of the thing it describes.
- Implementation models limit design and reuse capabilities.
- A document component model describes the complete set of semantic components in a domain.
- A document assembly model describes the way in which selected components are assembled into a hierarchical structure.
- When technologies change, the optimal implementation model will also change even though the underlying conceptual models don't.
- The two dimensions of model abstraction and granularity form the Document Engineering matrix.
- What constitutes metadata is relative.
- A common metamodel helps align different models.
- Business processes are inherently more abstract than documents.
- Patterns are models that are sufficiently general, adaptable, and worthy of imitation that we can reuse them.
- Using patterns saves effort and yields more consistent, compatible, and successful designs.

- Businesses exhibit both great variety and great regularity in what they do and how they do it.
- A business must balance how to differentiate itself and how to run according to industry practice.
- Generic conceptual patterns become more specific by adding context.
- A business analyst, document analyst, data analyst, and a task analyst will create different models.
- There are complementary and opposing relationships between processes and documents.

3.6 Notes

1. Object Management Group, "Introduction to OMG's Unified Modeling Language (UML)," http://www.omg.org/gettingstarted/what_is_uml.htm (last visited 18 September 2004).
2. "Information processing systems—Concepts and terminology for the conceptual schema and the information base," ISO/TR 9007 (1987).
3. Denise Schmandt-Besserat, *How Writing Came About* (University of Texas Press, 1996). The discovery of noniconographic counting tokens overturned the conventional view that Cuneiform writing evolved as an abstraction and simplification of pictorial representations
4. The model matrix is itself a conceptual artifact because we haven't yet created a physical knowledge repository organized in this way. But we are inspired to do so by the online MIT Process Handbook, which resembles it in some respects. See Thomas Malone, Kevin Crowston, and George Herman (Editors), *Organizing Business Knowledge: The MIT Process Handbook* (MIT Press, 2003). An online version is at http://ccs.mit.edu/ph/. This is an ambitious effort to organize knowledge about business models and business processes. It doesn't take the document / information exchange perspective of the model matrix or describe processes at that level of granularity, but it has a huge repertoire of patterns at less granular levels.
5. This is distinct from the use of *meta* meaning change or alteration (as in *metamorphism*).
6. The standardization of metadata for information exchange is led by the ISO TC154 Subcommittee 32 (http://metadata-stds.org/).
7. A downloadable version of the ebXML CCTS is available at http://www.oasis-open.org/committees/download.php/4259/CEFACT CCTS Version 2 of 11 August.pdf.
8. UN Economic Commission for Europe, *UN/CEFACT Modeling Methodology, version 8.1* (CEFACT/TMWG/N090R8.1, 2001.

9. *UN/CEFACT ebXML Business Process Specification Schema*. Version 1.10. 18 October 2003. Using the BPSS as the conceptual model for business processes in Document Engineering means that we are not really creating a Business Process Model. Since all of our process descriptions follow the same metamodel schema, it would be more correct to say we are creating a *Business Process Model Instance*. But this sounds a bit unnatural and makes it harder to treat document models and process models as two related views of the same thing, so we will often not tack on "instance" when we talk about business process models.
10. The idea of patterns as standard solutions in software design goes back several decades but is most often associated with the landmark book *Design Patterns* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (Addison Wesley, 1995). See also Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos. *Patterns for e-business: A strategy for reuse*. (IBM Press, 2001) and Gregor Hoppe and Bobby Wolfe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Systems.* (Addison-Wesley, 2004).
11. U.S. Census Bureau. *North American Industry Classification System.* See http://www.census.gov/epcd/www/naics.html.
12. United Nations Standard Processes and Services Code. See http://www.unspsc.org/.