

6

When Models Don't Match: The Interoperability Challenge

| | | |
|-----|--|-----|
| 6.0 | INTRODUCTION | 172 |
| 6.1 | THE INTEROPERABILITY CHALLENGE | 175 |
| 6.2 | CONTENT CONFLICTS | 181 |
| 6.3 | ENCODING CONFLICTS | 184 |
| 6.4 | STRUCTURAL CONFLICTS | 187 |
| 6.5 | SEMANTIC CONFLICTS | 193 |
| 6.6 | MOTIVATING THE DOCUMENT ENGINEERING APPROACH | 200 |
| 6.7 | KEY POINTS IN CHAPTER SIX | 203 |

6.0

INTRODUCTION

Web services and other technologies for service oriented architectures promise a future in which businesses will be able to discover each other, exchange electronic documents, and conduct transactions with or without prior agreement. This is the vision of extended or “virtual” enterprises composed from a variety of business services, including many provided by small and medium-sized enterprises or those from developing countries who were previously excluded from automated business relationships due to cost or technical barriers. New and more cost-effective and capable technologies will enable a seamless or “plug-and-play” business Internet in which loosely coupled document exchanges are the foundation for flexible, adaptive, and on-demand business models.

But not quite yet.



The most basic requirement for two businesses to conduct business is that their business systems interoperate

The most basic requirement for two businesses to conduct business is that their business systems interoperate. Interoperability doesn't require that two systems be identical in design or implementation, only that they can exchange information and use the information they exchange. Interoperability requires that the information being exchanged is conceptually equivalent; once this equivalence is established, transforming different implementations to a common exchange format is a necessary but often trivial thing to do. Interoperability is an easy goal to express but hard to achieve, especially if you want to avoid the overhead of expensive customized or hand-crafted integration solutions.

In this chapter we will explain why interoperability is a challenging goal by studying the types of conflicts that can arise when two enterprises try to exchange information. Of course, before enterprises can exchange information they must also understand and agree on the appropriateness of the exchange and on their respective responsibilities, roles, and commercial processes in the exchange. We'll return to

these contextual issues in Chapter 8, but for now let's assume two enterprises have come to these agreements and are beginning to exchange information.

The information exchanged might not match because of syntactic or encoding conflicts, because of structural or assembly conflicts, or because of conflicts in meaning or semantics. Some of these conflicts can be remedied or worked around, but others reflect basic incompatibilities in how the businesses understand their information and prevent interoperability from being achieved.

We can identify four different ways in which exchanges of information can be misunderstood. These are based on various combinations of content, syntax, structure, and semantic conflicts that can occur in any single document exchange. These categories are best explained by the following examples. As a simple case Figure 6-1 describes the ways in which we might communicate a value of 100 U.S. dollars:

Differences in Content:

- option a. <A>USD 100
- option b. <A>One Hundred US Dollars
- option c. <A>\$US100

Differences in Encoding:

- option a. <Amount>USD 100</Amount>
- option b. USD,100
- option c. CUR:USD|AMT:100

Differences in Structure:

- option a. <Amount>USD 100</Amount>
- option b. <Currency>USD</Currency><Amount>100</Amount>
- option c. <Amount>100<Currency>USD</Currency></Amount>

Differences in Semantics:

- option a. <Amount>USD 100</Amount>
- option b. <PreTaxAmount>USD90</PreTaxAmount><Tax>USD10</Tax>
- option c. <Price>USD 100</Price>

Figure 6-1. Four Ways to Misunderstand a Document Component

Each of these categories contains alternative ways to express the value of 100 U.S. dollars; the options in each case illustrate the meaning of the category. In most cases what is expressed might mean something to a person, but that's not what is at issue here. What matters is whether a business system can understand these different expressions to mean the same thing.

To better understand the conflict categories we will work backward from the physical to the conceptual view of our document models. We start with the information value itself, the content carried in a document instance. Every information component in the document follows some form of constraints on its possible values that defines its data type. For example, a value might be alphabetic text, integers, decimal, a specific pattern like those for dates and times, or one of a set of possible values, such as a coded list of countries or airports. The business system of the party receiving the document must know how to interpret these values, and it uses the explicit or implicit information about data types to do that.

In the first category of examples, the recipient's business system is more likely to be able to process "USD 100" than "One Hundred US Dollars" because the former follows a more prescribed data format than the latter, which appears to be informal words of text. We call this a problem in content.

We next consider the language used to describe the information. When two businesses make different choices in the implementation phase of their project, they might introduce conflicts in encoding.

XML, EDI, and structured text offer different languages for implementing document components. So we need to understand how these different implementation models influence interoperability.

We then need to recognize that similar syntax does not guarantee equivalent document or component structures. For example, all the components may be present but not in the expected arrangement. These are called structural conflicts.

Finally we need to examine the most serious conflicts, the ones that occur when component models diverge semantically because they are not defining the same things in the same context. These may reflect different requirements or choices made in the earliest phases of analysis and modeling.

We began this book with a comparison between buying a book in a bookstore and buying one online at GMBooks.com. Now let's imagine that GMBooks.com wants to accept electronic orders from affiliated booksellers that come via documents rather than from people browsing its website.¹

6.1

THE INTEROPERABILITY CHALLENGE

Making this happen seems simple: GMBooks.com publishes its requirements for the information that electronic orders must contain and the protocols it uses to receive messages.

Some of its partners can easily program or configure their business systems to send electronic orders that conform to the GMBooks.com specification. But others might not be able or willing to do so, and those are the situations that we'll discuss in this chapter.

Figure 6-2 illustrates this idea using the document exchanges among the various participants in the GMBooks.com virtual enterprise. The <BuyersID> in the message sent by the Affiliate Bookseller identifies the buyer, but when this information about the buyer appears in the Purchase Order, Shipping Note, and Transaction Advice documents it has a different meaning, name, or data format. These different documents must work together to carry out the drop shipment process using the overlapping information that flows between them, but unless these differences are resolved, the messages can't interoperate.

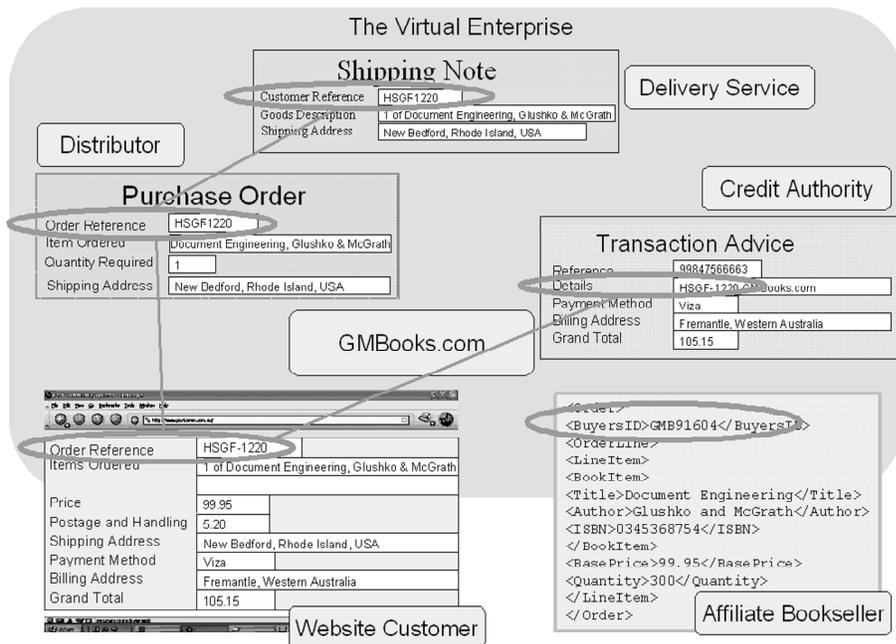


Figure 6-2. The Interoperability Challenge

6.1.1

THE INTEROPERABILITY TARGET

Let us assume that the information GMBooks.com needs for its order system is sensible, the buyer's name and address along with details about the ordered books. The conceptual model for the required order is shown in Figure 6-3 as a class diagram.

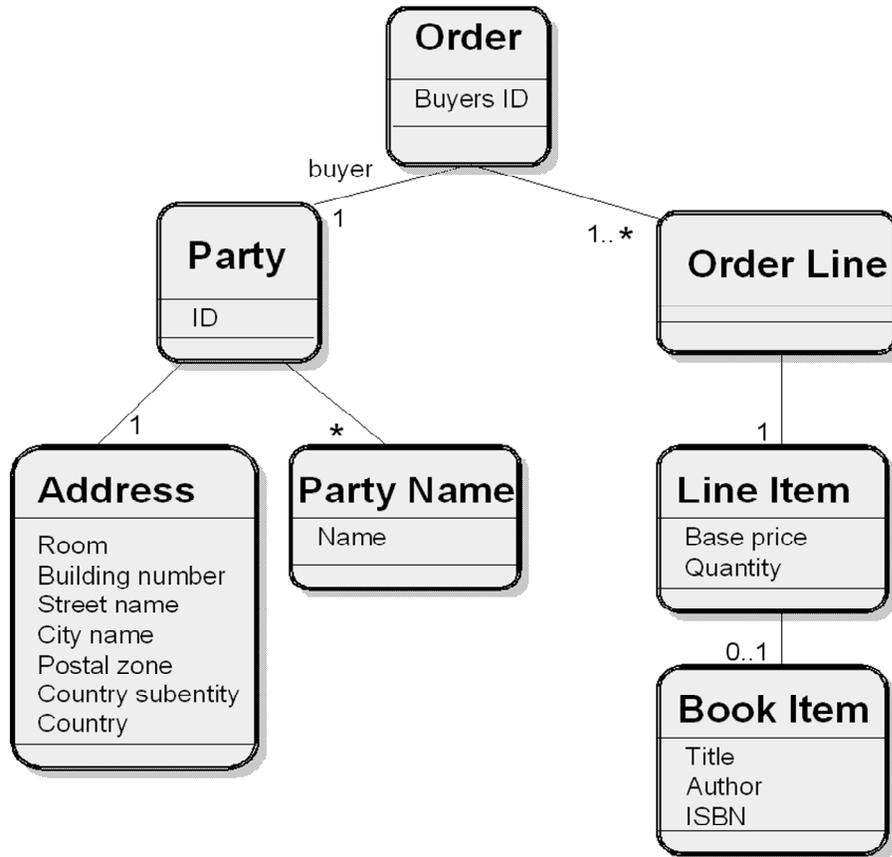


Figure 6-3. Conceptual Model for Orders

GMBooks.com may publish its specification for accepting electronic orders as the document implementation model expressed by the XML schema in Figure 6-4.²

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

```
<xs:element name="BuyerParty" type="PartyType"/>
<xs:element name="OrderLine" type="OrderLineType"
  MaxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="PartyType">
  <xs:sequence>
    <xs:element name="ID" type="xs:string"/>
    <xs:element name="PartyName" type="PartyNameType"/>
    <xs:element name="Address" type="AddressType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PartyNameType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element name="Room" type="xs:string"/>
    <xs:element name="BuildingNumber" type="xs:string"/>
    <xs:element name="StreetName" type="xs:string"/>
    <xs:element name="CityName" type="xs:string"/>
    <xs:element name="PostalZone" type="xs:string"/>
    <xs:element name="CountrySubentity" type="xs:string"/>
    <xs:element name="Country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="OrderLineType">
  <xs:sequence>
    <xs:element name="LineItem" type="LineItemType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemType">
  <xs:sequence>
    <xs:element name="BookItem" type="BookItemType"/>
    <xs:element name="BasePrice" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:element name="Quantity" type="xs:int"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="BookItemType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Author" type="xs:string"/>
    <xs:element name="ISBN" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Figure 6-4. Document Implementation Model (XML Schema) for Orders

A typical instance of an order that conforms to this schema might look like Figure 6-5.

```
<?xml version="1.0" encoding="UTF-8"?>
<Order>
  <BuyersID>GMB91604</BuyersID>
  <BuyerParty>
    <ID>KEEN</ID>
    <PartyName>
      <Name>Maynard James Keenan</Name>
    </PartyName>
    <Address>
      <Room>505</Room>
      <BuildingNumber>11271</BuildingNumber>
      <StreetName>Ventura Blvd.</StreetName>
      <CityName>Studio City</CityName>
      <PostalZone>91604</PostalZone>
      <CountrySubentity>California</CountrySubentity>
      <Country>USA</Country>
    </Address>
  </BuyerParty>
  <OrderLine>
    <LineItem>
      <BookItem>
```

```
<Title>Document Engineering</Title>
<Author>Glushko and McGrath</Author>
<ISBN>0262072610</ISBN>
</BookItem>
<BasePrice>99.95</BasePrice>
<Quantity>300</Quantity>
</LineItem>
</OrderLine>
</Order>
```

Figure 6-5. Instance of an Order Document

6.1.2

RECOGNIZING EQUIVALENCE

We might assume that any affiliate would be able to send this simple document. But experience has taught us that this is not the case. Variations in strategies, technology platforms, legacy applications, business processes, and terminology make it difficult or impossible for some firms to satisfy the requirements.



Variations in strategies, technology platforms, legacy applications, business processes, and terminology make it difficult to use compatible documents

If GMBooks.com were the dominant bookseller on the Web, it might try to compel its affiliates to comply with its process and information requirements as a condition of doing business, but this strategy is rarely successful because it does not encourage loyalty. In most cases, GMBooks.com would adopt the practical approach of trying to accept orders in whatever form they are sent. In this situation, GMBooks.com needs to assess whether it can extract the information it needs from what it receives. So the challenge GMBooks.com faces when it reviews order documents is determining whether they conform to their information requirements; that is, to recognize equivalence.

Initially GMBooks.com might test every incoming order document against its schema in Figure 6-4 and simply reject any order that isn't well-formed XML or a valid

instance of their document schema (see Section 2.6). But this assessment is more difficult than it might seem. If an incoming invalid message contains elements or attributes whose names match those that are expected, it would be tempting to extract them and rearrange them to conform to the target schema. However, the same names don't necessarily imply that the meanings are the same. Or the names might not match but the underlying concepts might be identical. To establish semantic equivalence, we need to compare conceptual representations and determine whether the different physical models (such as schemas) relate to the required conceptual ones.

Unfortunately most documents don't arrive with an associated conceptual representation that unambiguously defines the meaning encoded in the physical model. We need to apply Document Engineering techniques to determine whether the required information can be extracted and transformed from the incoming message.

In this case, we might think, "What's so hard to understand about names, addresses, and books?" We hope the examples in this chapter will explain that things are not always as obvious as they seem. In Chapters 11 and 12, "Analyzing Documents" and "Analyzing Document Components," we will introduce some techniques for encouraging semantic clarity in conceptual models.

6.2

CONTENT CONFLICTS

Certain conflicts can arise if two business systems use different data types for the content of the same component. Take, for instance, the following snippet of an order in Figure 6-6:

```
<LineItem>
  <BookItem>
    <Title>Document Engineering</Title>
    <Author>Glushko and McGrath</Author>
    <ISBN>0262072610</ISBN>
  </BookItem>
  <BasePrice>$99.95</BasePrice>
  <Quantity>300</Quantity>
</LineItem>
```

Figure 6-6. Order Fragment with Base Price Content Conflict

In this order example the base price for the book contains a \$ symbol. This creates a data type conflict in the content of the component. In Figure 6-4 we can see that GBooks.com has defined BasePrice in its XML schema as a decimal (meaning a positive or negative number with a decimal point) and this does not allow a currency code or symbol.³ The \$ symbol in the base price value sent by the affiliate may cause it to be rejected by the GBooks.com order system, even though the meaning, structure, and syntax of the value provided by the affiliate are correct. The content value provided does not satisfy the possible character set specified in the GBooks.com definition.



Content conflicts occur when two parties use different sets of values for the same components

Content conflicts often happen when the possible values for instances of a component must be conformant to a specified pattern or to a set of otherwise arbitrary codes or identifiers. The latter are commonplace in business documents where using a fixed set of possible values allows for precise identification. For example, many enterprises use an identification scheme for countries, usually the ISO 3166-1 alpha-2 codes that have values like US for United States of America and CN for China. When possible values are taken from external standards agencies, such as the ISO, they are called external codes to emphasize that they are not under the enterprise's control.

Precise identification by each side of an exchange using some set of legal values isn't sufficient in itself. The trading parties need to agree on the set of values or the patterns that define acceptable values.

```
<Address>
  <Room>505</Room>
  <BuildingNumber>11271</BuildingNumber>
  <StreetName>Ventura Blvd.</StreetName>
  <CityName>Studio City</CityName>
  <PostalZone>91604-3136</PostalZone>
  <CountrySubentity>California</CountrySubentity>
  <Country>USA</Country>
</Address>
```

Figure 6-7. Order Fragment with Postal Zone Content Conflict

In Figure 6-7 the GMBooks.com affiliate provides a value for the postal zone using the U.S. Postal Services nine-digit Zip+4 coding scheme. However the example we saw in Figure 6-5 uses the less specific five-digit Zip code. Both of these external coding schemes are acceptable to postal services, but the former may be a content conflict for GMBooks.com.

At other times possible values are internally defined by a single enterprise for its own use. Some examples of internal sets of possible values can be seen in our GMBooks.com document models (Figures 6-3 and 6-4). First, a BuyerPartyID, the value that uniquely identifies each buyer, would probably be assigned by GMBooks.com to each customer when the parties establish a business arrangement. Secondly, a BuyersID may be issued by affiliates to identify their orders. The rules or sets of the values for each of these will be controlled by the originating party.



Content conflicts occur when two parties use the same sets of values for different components

These internal sets of values can often be the cause of content conflicts because both parties may be using the same or overlapping sets for different components. We further discuss the analysis and encoding of sets of possible values in Section 12.1.8.

6.3

ENCODING CONFLICTS

A more obvious way in which information exchanges can conflict is at the level of encoding—that is, the language chosen for implementing the exchange or the way information is represented by the language’s syntax.

6.3.1

LANGUAGE CONFLICTS

The most apparent differences in encoding occur when two different languages are chosen. For example, Figure 6-8 denotes an order document using the UN/EDIFACT (ISO 9735) standard.

```
UNH+0GMB91604004600001+ORDERS:1:911:UN+362910 04061815???:15'  
BGM+120+362910+9'  
DTM+4:040618:101'  
NAD+BY+KEEN::91++MAYNARD JAMES KEENAN'  
NAD+VN+GMBOOKS.COM::92++GM BOOKS LTD'  
UNS+D'  
LIN+1'  
PIA+1+0262072610:IS'  
IMD+F+2+:::DOCUMENT ENGINEERING BY GLUSHKO AND MCGRATH'  
QTY+21:300.0000:EA'  
PRI+CON:99.95'  
UNS+S'  
CNT+2:2'  
UNT+23+000091604004600001'
```

Figure 6-8. Order Encoded in UN/EDIFACT

Clearly this is not immediately compatible with the order example in Figure 6-4. But as UN/EDIFACT is the only internationally recognized standard for electronic order documents, the affiliate might be annoyed to be told by GMBooks.com that it is using an unacceptable format.

There is also a popular EDI language developed by the American National Standards Institute known as ANSI ASC X12. During the 1990s this standard was increasingly adopted by U.S. publishers and booksellers and built into their order processing systems. In such cases an affiliate's order document might look like that in Figure 6-9.

```
ST*850*000820
BEG*00*SA*820**040605
N1*ST*KEEN*92*GMB91604
PO1*1*1*EA***EN*0262072610
PID*F****Document Engineering GLUSHKO MCGRATH
PO4**300*EA
CTT*2
SE*56*000820
```

Figure 6-9. Order Encoded in ANSI ASC X12

About 20 years before the development of standard EDI languages in the 1980s, the Book Industry Study Group developed a format known as BISAC for ordering books. Many small and medium-sized booksellers might still use specialized sales management software that can produce only BISAC formatted orders. In such cases the order document might look like Figure 6-10.

```
00000018800868 GMB91604 946305INTERNET.BSC F039000178
1000002820 8800868 9230178 040605000000Y 000000001N00020000000
4000003820 Y0000000002620726100000300000000000000000000041000000
4200004820 Document Engineering GLUSHKO and McGRATH
5000037820 0000200000000170000000029
900003800000000000170000100000000290000100001000000000000170000100000000000000
```

Figure 6-10. Order Encoded in BISAC

We can also imagine a small and technologically unsophisticated affiliate bookseller who keeps records in a spreadsheet. An XML application interface might seem daunting to this partner, and the only way they can export and import orders is in comma-separated files. In such cases the order document might look like Figure 6-11.

```
KEEN91604,Dr.,Maynard,James,Keenan,11271 Ventura Blvd. #505,Studio  
City,California,91604  
Document Engineering,Glushko & McGrath,0262072610,99.95,300
```

Figure 6-11. Order Encoded in Comma-separated Syntax

A comparison of the documents in Figures 6-8 to 6-11 with the conceptual model of the order depicted in Figure 6-3 reveals that each is based on similar concepts and each appears to convey information suitable for GMBooks.com requirements.

However, the components of all of them require mapping or transforming into their GMBooks.com counterpart. For example, we would need to know that in the UN/EDIFACT order, NAD+BY+ indicates the GMBooks.com Order/Buyer/ID, or that for any rows starting with the code number 42 in the BISAC document, columns 21 to 50 contain the OrderLine/LineItem/BookItem/Title.



A one-to-one mapping of document components
is not always achievable

As you can imagine from the above examples, one-to-one mapping is not always achievable. Numerous mapping or translation tools exist to convert EDI and other formats to XML (and vice versa), but most of them work near the surface of the message to relate parts of one message to the other and don't provide much support for understanding or reusing the models below the surface.⁴

6.3.2

GRAMMATICAL CONFLICTS

Even if both parties encode their models using the same language, differences in applying the grammar of the language can prevent their documents from interoperating.

Many XML encoding conflicts result from different uses of the element and attribute constructs.⁵ For example, GMBooks.com might have an affiliate whose order system generates XML instances that look like Figure 6-12.

```
<BuyerParty ID="KEEN" Name="Maynard James Keenan" Room="505"
BuildingNumber="11271" StreetName="Ventura Blvd." City="Studio City"
State="California" PostalCode="91604">
  <Item Title="Document Engineering" Author="Glushko & McGrath"
ISBN="0262072610" BasePrice="99.95" Quantity="300"/>
```

Figure 6-12. Order Encoded Using XML "Attribute-Value" Style

In contrast to the XML document that GMBooks.com expects, this partner's XML instance encodes almost everything as attributes to minimize the size of the document. The GMBooks.com order system will not immediately be able to accept this instance. However, a comparison of the XML document in Figure 6-12 with the conceptual model of the order depicted in Figure 6-3 reveals that the conceptual models are essentially the same. Only the XML naming and design rules are different.

As a result, these sorts of encoding conflicts between XML documents are quite easy to resolve using XSLT and XPath (Section 2.7.2).



Encoding conflicts can be resolved if the underlying semantics and structures are compatible

Encoding conflicts are generally resolvable if the underlying semantics and structures are compatible because encoding a conceptual model as a physical one is the last phase before implementation (see Figure 3-1). If two parties have been creating models for the same business context, they will have similar conceptual models and assemblies of structures, any different choices at the encoding phase should be easy to diagnose and reconcile. We revisit issues of encoding rules in Chapters 7 and 15.

6.4

STRUCTURAL CONFLICTS

Another category of conflicts arises when the models of documents or their components have different structures. Even when both parties use the same encoding rules, structural conflicts can cause interoperability problems.

6.4.1

DOCUMENT ASSEMBLY CONFLICTS

In the GMBooks.com business process, an affiliate might model customer information and order information the same way that GMBooks.com does but might produce two documents like those in figures 6-13a and 6-13b.

```
<?xml version="1.0" encoding="UTF-8"?>
  <BuyerParty>
    <ID>KEEN</ID>
    <PartyName>
      <Name>Maynard James Keenan</Name>
    </PartyName>
    <Address>
      <Room>505</Room>
      <BuildingNumber>11271</BuildingNumber>
      <StreetName>Ventura Blvd.</StreetName>
      <CityName>Studio City</CityName>
      <PostalZone>91604</PostalZone>
      <CountrySubentity>California</CountrySubentity>
      <Country>USA</Country>
    </Address>
  </BuyerParty>
```

Figure 6-13a. Buyer Information Document

```
<?xml version="1.0" encoding="UTF-8"?>
<Order>
  <BuyersID>GMB91604</BuyersID>
  <BuyerParty>
    <ID>KEEN</ID>
  </BuyerParty>
  <OrderLine>
    <LineItem>
      <BookItem>
        <Title>Document Engineering</Title>
```

```

    <Author>Glushko & McGrath</Author>
    <ISBN>0262072610</ISBN>
  </BookItem>
  <BasePrice>99.95</BasePrice>
  <Quantity>300</Quantity>
</LineItem>
</OrderLine>
</Order>

```

Figure 6-13b. Order Information Document

Because the document instances can be linked by the Buyer's ID number, these two documents can easily be merged to create the order needed by GMBooks.com, and because each document conforms to the expected structure for its part, no additional transformation is required.

6.4.2

COMPONENT ASSEMBLY CONFLICTS

A more serious problem occurs when the two parties have assembled components into structures in incompatible ways. This may happen when they view some of the components in a different context. For example, GMBooks.com might have an affiliate who consolidates orders for smaller retailers and submits one order on behalf of several buyers. This business model naturally results in an item-centric view of the information rather than the customer-centric view expected by GMBooks.com. Such an item-centric order might look like Figure 6-14.

```

<?xml version="1.0" encoding="UTF-8"?>
<Order>
  <OrderLine>
    <LineItem>
      <BookItem>
        <Title>Document Engineering</Title>
        <Author>Glushko & McGrath</Author>
        <ISBN>0262072610</ISBN>
      </BookItem>
    </LineItem>
  </OrderLine>
</Order>

```

```
    <BasePrice>99.95</BasePrice>
    <Quantity>300</Quantity>
  </LineItem>
  <BuyersID>91604</BuyersID>
  <BuyerParty>
    <ID>KEEN</ID>
    <PartyName>
      <Name>Maynard James Keenan</Name>
    </PartyName>
    <Address>
      <Room>505</Room>
      <BuildingNumber>11271</BuildingNumber>
      <StreetName>Ventura Blvd.</StreetName>
      <CityName>Studio City</CityName>
      <PostalZone>91604</PostalZone>
      <CountrySubentity>California</CountrySubentity>
      <Country>USA</Country>
    </Address>
  </BuyerParty>
</OrderLine>
</Order>
```

Figure 6-14. Item-Centric Order Document

Even though they have the same models for names, addresses, and other components in isolation, the differences in how they are put together results in different hierarchies and different documents.

More significantly, the position of components in the hierarchy affects their meaning. For example, the component `BuyersID` in Figure 6-14 is not necessarily the same component used in the schema in Figure 6-4. We cannot ascertain whether it means the identifier used by the buyer for the item or for the whole order.

In another example we could consider two types of event calendars: One is date-centric, listing for each date the events that take place; another, which might be more appropriate when most dates don't have events scheduled, is event-centric, listing

events and for each the date or dates on which they take place. In the former type of event calendar, every scheduled occurrence of an event is explicit. In the latter type, it would be reasonable to specify a start date and end date for events that span multiple dates, making the list of occurrences implicit.

In this situation it might be technically possible to transform event-centric calendars into date-centric calendars by interpolating the implicit occurrence of dates in the former, and most of the time the transformation would be semantically correct. But in some cases the transformation might require other information, like whether the event is something that takes place only during weekdays and whether holidays are excluded, information that is explicit in the date-centric calendar.⁶ We will expand on this event calendar modeling project as our case study in Chapters 8 through 15.



The earlier in the modeling process that two parties make different decisions, the greater the possibilities for their models to be incompatible

Such assembly conflicts represent a more serious set of interoperability problems than the simpler encoding conflicts, because assembly occurs before encoding in the modeling process. Put another way, the earlier in the modeling process that two parties make different decisions, the greater the possibilities for their models to be incompatible.

6.4.3

COMPONENT GRANULARITY CONFLICTS

We might also encounter conflicts that derive from identifying components in different levels of detail—these are issues about the granularity of structure in a component. These kinds of interoperability challenges are illustrated in the order fragments shown in figures 6-15a and 6-15b.

```
<BuyerParty>
  <ID>KEEN</ID>
  <PartyName>
    <Name>Maynard James Keenan</Name>
  </PartyName>
  <Address>
    <StreetAddress>11271 Ventura Blvd. #505</StreetAddress>
    <City>Studio City 91604</City>
    <CountrySubentity>California</CountrySubentity>
    <Country>USA</Country>
  </Address>
</BuyerParty>
```

Figure 6-15a. BuyerParty Fragment with Underspecified Granularity

```
<BuyerParty>
  <ID>KEEN</ID>
  <PartyName>
    <FamilyName>Keenan</FamilyName>
    <MiddleName>James</MiddleName>
    <FirstName>Maynard</FirstName>
  </PartyName>
  <Address>
    <Room>505</Room>
    <BuildingNumber>11271</BuildingNumber>
    <StreetName>Ventura Blvd.</StreetName>
    <CityName>Studio City</CityName>
    <PostalZone>91604</PostalZone>
    <CountrySubentity>California</CountrySubentity>
    <Country>USA</Country>
  </Address>
</BuyerParty>
```

Figure 6-15b. BuyerParty Fragment with Overspecified Granularity

In Figure 6-15a, the information components for Room, BuildingNumber, and StreetName from the GMBooks.com conceptual model have been combined into a StreetAddress component.

In Figure 6-15b, the components that make up the Name of the Party in the GMBooks.com model have been more precisely expressed as separate components for FamilyName, MiddleName, and FirstName.

These granularity differences result in one-way interoperability; a more granular model can be transformed into a less granular model, but not vice versa. A transformation could be written that would take the values of Room, BuildingNumber, and StreetName from Figure 6-15b and combine them into StreetAddress to produce the desired instance. But we would not reliably be able to decompose the StreetAddress from Figure 6-15a into the Room, BuildingNumber, and StreetName components required for the GMBooks.com target document.⁷



A more granular model can be transformed into a less granular model, but not vice versa

We won't belabor this argument by showing that the granularity transformation challenge is equally severe when it comes to personal names, dates, telephone numbers and many common document components. We can all imagine how scrambled computer-addressed mail might be for Dr. Jean-Pierre Paul van Gogh-Shakespeare III, Esq.

We will explain our approach for aggregating components and creating document assembly models from a component model in Chapters 12, 13, and 14.

6.5

SEMANTIC CONFLICTS

By far the most complex issues affecting interoperability in document exchange are the result of semantic conflicts. Even if we resolve the encoding and structural conflicts, we have a long way to go to ensure meaningful communication of information.



Even if we resolve the encoding and structural conflicts, we have a long way to go to ensure meaningful communication of information

6.5.1

FUNCTIONAL DEPENDENCY CONFLICTS

Suppose an affiliate of GMBooks.com submits the order shown in Figure 6-16.

```
<OrderLine>
  <LineItem>
    <BookItem>
      <ISBN>0262072610</ISBN>
      <BasePrice>99.95</BasePrice>
    </BookItem>
    <Quantity>300</Quantity>
  </LineItem>
</OrderLine>
```

Figure 6-16. Fragment of an Order Document Lacking Book Titles and Authors

This fragment of an order document omits the book title and author from the item information.

We need to consider why the order might have been designed in such a way. We begin by referring to the GMBooks.com conceptual model for the LineItem in Figure 6-3, which consists of five information components. The modelers at GMBooks.com apparently concluded that title, author, and ISBN (the elements contained in BookItem) are a group of information components that together logically describe a book. We say they are functionally dependent.

By comparison, the components known as BasePrice and Quantity only modify the properties of a book when it appears on a line item within an order. Each order may have a different quantity or price for a book so they are not functionally dependent on the book itself.

However, the designers of the affiliate's documents appear to have decided that there will only be a single price for a book and this information component logically belongs to a BookItem.



Having different views of the dependency relationships mean business rules and semantics are interpreted differently

This is a design conflict based on different views of the dependency relationships between the information components. The business rules and therefore the semantics are interpreted differently. The two parties use different models for how information components associate with each other. In these situations, the resulting documents may be incompatible. We will explore the formation of assemblies of components based on their dependencies in Chapter 13.

6.5.2

VOCABULARY CONFLICTS

When we looked at encoding conflicts in Section 6.3, we discussed the language and grammar of the implementation. But when we talk about semantics we need to examine the vocabulary. We use this vocabulary to convey the semantics of components, including the names we give them. And the way we implement these names also involves syntax considerations, such as with naming tags in XML documents.

XML Tag Names

When encoding document models, the creators of an XML vocabulary sometimes seek to avoid problems and disputes by automatically generating tag names that have no equivalent in any natural language. For example, one proposal to create XML versions of the standard UN/EDIFACT messages suggested five-letter "UN-XML" tags like <HFKDR>, <BBBTS>, and <RTFDS>8 whose meaning would be specified by the mapping of the tag to items in the standard UN/EDIFACT data dictionary.

To some extent, we applaud the use of arbitrary tag names because it further rebuts the notion that XML is somehow “self-describing” and that schemas are optional (see Section 2.5.3). And we appreciate the desire to avoid bias. But we are convinced that generating meaningless tag names is a bad idea. It would be better to start with names that help business analysts, programmers, and other users of the vocabulary to do their jobs. It’s easy enough to then transform the names for anyone who doesn’t like them or who needs a different set.

Choosing the terms used for naming is often a difficult and contentious activity. Everyone naturally wants to create names in his or her native language, but even in the same language or family of dialects, the same concepts have multiple words or even different spellings for the same word (consider, for example, the bewildering differences among the numerous varieties of English). Even when describing exactly the same document component, chances are very good that two developers or two teams of data modelers will choose different names for it.⁹ In Section 7.5.2 we’ll mention two possible solutions: controlled vocabularies, a closed set of defining terms, and formal ontologies, which define the meaning of terms using a formal or logic-based language.



Two modelers will often choose different names for the same component

Given this difficulty, GMBooks.com might encounter an affiliate with the instance in Figure 6-17 that has the correct conceptual model but not the expected tag names:

```
<Customer>
  <Number>KEEN</Number>
  <Name>
    <BusinessName>Maynard James Keenan</BusinessName>
  </Name>
  <Location>
    <Unit>505</Unit>
    <StreetNumber>11271</StreetNumber>
    <Street>Ventura Blvd.</Street>
    <City>Studio City</City>
    <ZipCode>91604</ZipCode>
    <State>California</State>
```

```
<Country>USA</Country>
</Location>
</Customer>
```

Figure 6-17. Buyer Party with Different Tag Names

This also applies when using different languages. For example, GMBooks.com might receive an order with components such as those in Figure 6-18 from a French affiliate.

```
<Acheteur>
  <ID>KEEN</ID>
  <Nom>
    <NomCommercial>Maynard James Keenan</NomCommercial>
  </Nom>
  <Adresse>
    <Appartement>505</Appartement>
    <Bâtiment>11271</Bâtiment>
    <Rue>Ventura Blvd.</Rue>
    <Ville>Studio City</Ville>
    <CodePostal>91604</CodePostal>
    <Etat>California</Etat>
    <Pays>USA</Pays>
  </Adresse>
</Acheteur>
```

Figure 6-18. Buyer Party with Tag Names in French

Both these document's components conform to the conceptual model in Figure 6-3 and only the names are different. In other words, Buyer Party, Customer, and Acheteur all refer to the same concept, that is, the party purchasing the goods.



XML is not self-describing

This reemphasizes that XML is not self-describing and confirms that the names we assign to tags are only a small part of defining the meaning of the information they contain.

But even if two separate modelers are unlikely to employ an identical set of component names, each single modeling effort should have a system for keeping names logical and consistent. Guidelines for naming conventions are discussed further in Sections 7.6.2 and 12.1.11.



The names of components are only a small part of their semantic definition

6.5.3

CONTEXTUAL CONFLICTS

Imagine that when GMBooks.com developed their order system, they were taking orders only from the North America. Later they decided to branch out into international orders, and they received an order from Japan. Part of this order is shown in Figure 6-19:

```
<Buyer>
  <ID>KEEN91604</ID>
  <FullName>
    <Title>Dr.</Title>
    <FirstName>Maynard</FirstName>
    <MiddleName>James</MiddleName>
    <LastName>Keenan</LastName>
  </FullName>
  <Address>
    <PostalCode>170-3293</PostalCode>
    <Prefecture>Tokyo</Prefecture>
    <Ward>Chuo</Ward>
    <Subarea>Ginza</Subarea>
    <SubareaNumber>5</SubareaNumber>
    <BlockNumber>2</BlockNumber>
    <HouseNumber>1</HouseNumber>
  </Address>
</Buyer>
```

Figure 6-19. Order Fragment with Incompatible Postal Address

This is a perfectly reasonable postal address model in Japan, but it is conceptually incompatible with the postal address model expected by GMBooks.com. In Japan, streets are not all named and addresses are designated by ever-smaller subdivisions of a city. There is simply no way to transform a Japanese address into the desired instance. If GMBooks.com wants to fulfill this order and do business with Japanese customers, it needs to redesign its system to handle Japanese orders. This requires a separate model for Japanese addresses or changes to the existing model to accommodate it. The semantic conflict here resulted from a limited understanding of what constituted an address. The sample of documents or information sources that were analyzed to produce the original address model was too narrow. Perhaps GMBooks.com looked only at examples of American and Canadian addresses or decided that Japanese orders were unlikely. On the other hand, the affiliate in Japan was basing its model on a sample of addresses that seemed representative to it. Both parties thought they were designing for the same business context of online bookselling, but they chose only local sources to obtain their information requirements.



Different document samples can lead to incompatible models

The decision about what information sources to analyze when developing a model—the inventory and sampling phase—occurs early in the modeling process. So if different parties begin with different samples, their context of use can diverge at a very early stage and chances are that the resulting models will be incompatible. We discuss the techniques for collecting the inventory of sources and choosing an appropriate sample in Section 7.5 and Chapter 11. The inventory will include information sources that are not in the form of traditional documents, such as databases, spreadsheets, web pages, and the people who create and use them.

Finally, an even more extreme case of incompatibility can be seen in the order snippet in Figure 6-20:

```
<BuyerParty>
  <ID>KEEN</ID>
  <PartyName>
    <Name>Maynard James Keenan</Name>
  </PartyName>
  <Address>
    <Latitude direction="N">37.871</Latitude>
    <Longitude direction="W">-122.271</Longitude>
  </Address>
</BuyerParty>
```

Figure 6-20. Order Fragment with an Address That Isn't Postal

This (admittedly a bit contrived) document fragment might result if someone wanted to order books to be delivered to an offshore oil platform in the Yellow Sea.¹⁰ But in this case, the location designation is not a postal address in any sense. As a set of coordinates, it is wholly outside the context of the Address model designed by GMBooks.com. This is a case of semantic mismatch. While GMBooks.com defined Address in their model to accommodate locations recognizable to postal services, this example concerns locations in a different context. As such, the meaning of the two component models differs, and no interoperability can really be expected.



Semantic conflicts should be resolved when the context of use is being defined

This is a difficult conflict to resolve, and it can be dealt with only at the very first step in the modeling process when the context is being defined. In Section 7.3, we explore in more detail the activities involved in determining the modeling context, including the identification of patterns, requirements, and business processes that must be supported.

6.6 MOTIVATING THE DOCUMENT ENGINEERING APPROACH

Interoperability is a desirable goal but not one that is easily achieved. Using what may have seemed at the outset to be a simple order document, we have identified a

range of potential conflicts when two parties conceptualize, organize, or implement their models of names, addresses, and books differently. Some types of conflicts are minor and can be easily resolved, while others require case-by-case analysis or may be impossible to resolve (See SIDEBAR).



Many of the claims about web services and a “plug-and-play” business are exaggerated and naïve

The variety of ways in which two models might not match should make it obvious that many of the claims about web services and a seamless or “plug-and-play” business Internet are exaggerated and naïve. Likewise, even though extraction, mapping, and transformation technology continues to improve, fixing problems when two models don’t match is likely to remain a labor-intensive activity.

Automatic Resolution of Interoperability Conflicts

Applications can be written to have very precise expectations about the input they receive and to reject any input that doesn’t conform. This is generally a bad approach from both technical and business standpoints. While it is good for an application to be conservative in what it sends, it should try to be liberal in what it accepts and be programmed with the philosophy that unexpected inputs may represent new requirements that it should be able to handle.

Nevertheless, being open and extensible for new input requirements doesn’t mean that the application should automatically try to accept nonconforming messages. As we’ve seen in this chapter, a mismatch in physical implementations may or may not imply a mismatch in conceptual models, and most of the time it takes a person rather than a program to decide. Automated programs can propose classifications or diagnoses of the nonconforming messages, but ultimately the decision about whether to accept a message and try to make use of its contents should be made by someone who understands interoperability from a broad business and technical perspective in the specific context in which the message is received.

Once we understand the meaning of the input, some kinds of conflicts are relatively superficial, and we can write general-purpose data extraction and transformation programs to resolve them. From then on any messages with those conflicts can be automatically processed. Other kinds of conflicts are deeper, and while we might be able to work around them with custom programming, we need to be concerned that the differences in conceptual models might cause problems elsewhere in our business systems.

But it is worthwhile studying how things can go wrong only if we use what we learn to make things go right. Each way in which documents can fail to interoperate—each problem or conflict we encounter—can be turned around to motivate a Document Engineering activity to remedy or prevent such problems. Modeling is difficult, but we can and should study good modelers and good models to learn skills and principles to use when we do it. This makes a good model a significant intellectual achievement that deserves to be reused. Starting with Chapter 7, we introduce a Document Engineering approach that emphasizes careful modeling and the reuse of models whenever possible.

6.7

KEY POINTS IN CHAPTER SIX

- The most basic requirement for two businesses to conduct business is that their business systems interoperate.
- Many of the claims about web services and a “plug-and-play” business are exaggerated.
- Variations in strategies, technology platforms, legacy applications, business processes, and terminology make it difficult to use compatible documents.
- The earlier in the modeling process that two parties make different decisions, the greater the possibilities for their models to be incompatible.
- If projects begin with different samples, their models can diverge and the resulting models will be incompatible.
- A one-to-one mapping of document components is not always achievable.
- Content conflicts occur when two parties use different sets of values for the same components.
- Content conflicts also occur when two parties use the same sets of values for different components.
- Encoding conflicts can be resolved if the underlying semantics and structures are compatible.
- A more granular model can be transformed into a less granular model, but not vice versa.

- Even if we resolve the encoding and structural conflicts, we have a long way to go to ensure meaningful communication of information.
- Semantic conflicts should be resolved when the context of use is being defined.
- Having different views of dependency relationships creates different contexts of use.
- Two modelers will often choose different names for the same component.
- The names we assign to components are only a small part of defining their meaning.