CHAPTER 2

Literate modelling

2.1 *Acknowledgements*

We'd like to acknowledge Dr. Wolfgang Emmerich of University College London, Mr. John Quinn of British Airways for their help in the work we present in this chapter. We initially presented the core of ideas of Literate Modelling in a paper entitled "Literate Modeling - Capturing Business Knowledge with the UML" [Arlow2] at the «UML»'98 conference.

2.2 *Introduction*

One problem you will find with UML models (and in visual models in general) is that the valuable information captured in the model is only accessible to those who know the visual syntax of the modelling language. In a sense, valuable information about the business becomes encrypted in a concise, elegant modelling language that is only accessible to an elite who are "in the know".

But we think there is a way around this problem, and this is the topic of this chapter.

In fact, it is not just the visual syntax of UML models that creates problems. If you need to access the information embedded in a model, you may also need to know

how to work a CASE tool to navigate that information effectively. All CASE tools can generate reports, often in HTML format, but you may find that these reports are hard to read and navigate, and are (at least in our experience) of little practical use.

Also, unless you already know the general "shape" of a model, knowing precisely where to start with either the model in a CASE tool, or with a generated report can be difficult.

Even though you may know UML syntax and also know how to work the CASE tool, you may still find it difficult, and often impossible, to uncover the important business requirements and imperatives that shaped the model and give it it's business value.

When key information is taken out of its business context and expressed in UML or some other abstract visual notation, it often becomes invisible. In our 1998 paper [Arlow2] we call this the *trivialisation of business requirements*.

A result of trivialisation is that it is quite common for an Analyst or Designer to be unable to describe the forces, business context and requirements that shaped a UML model that they constructed just a few months previously.

Trivialisation of business requirements it is not unique to the UML but seems to be a universal feature of all visual modelling languages. This is because one of the key strengths of visual modelling, its conciseness and terseness, is also in some cases a weakness.

When you represent important business requirements a class, relationship, method, constraint, multiplicity or some other element on a UML diagram, the requirement lost amidst other similar modelling elements that may have much less business significance.

Our solution to this problem is to extend the UML model by providing a narrative description that is accessible to many different readers, not just those "in the know". This is what we call a "literate model".

We got the idea for literate modeling from our own experiences in trying to explain complex UML Enterprise Object Models to a wide spectrum of stakeholder from those with detailed knowledge of UML modelling to those with no knowledge at all. We called the technique "literate modeling" as it is in some ways similar to "literate programming" as discussed in [Knuth1]. Literate programming tried to make

programs more comprehensible by embedding them in an explanatory narrative, and literate modelling essentially tries to do the same thing, but for UML models.

In practice, literate programming, although a very good idea, didn't really take off too well. This was partly because of it's reliance on special text processing tools that were not widely available, and partly because programmers generally prefer to write code, rather than narrative!

In contrast to this, literate modelling has proven to be *very* popular with those who have used it. This is because a literate model not only provides a context for a UML model that is otherwise lacking, it also helps the modeler to do his or her work.

Creating a literate model as part the process of creating a UML model *will* improve the quality of your thought processes and of your modeling. You will also achieve enhanced communication with both technical and non-technical stakeholders.

2.3 *Comprehensibility and accessibility of UML models*

In this section we show you our assessment of how well different groups of people involved in a software development project are able to access and comprehend the various types of UML model.

We'd like to point out that this assessment is based on *experiential* evidence that we have accumulated over many man-years of using the UML in various substantial and mission critical projects. We first discussed these ideas in 1998 [Arlow2] and since then we have had many letters supporting these results. We are confident that, although subjective, this assessment is pretty accurate. However, it's a big world, and if you've had other experiences, we'd like to hear from you!

We're going to consider two aspects of UML models, their comprehensibility and their accessibility.

- Comprehensibility - the ability to understand the business semantics of the model. Comprehensibility is the key to obtaining business value from UML models, and it is often contingent on accessibility.
- Accessability - the ability to access the information contained in a UML model. There are two components to access ability:

1. Ability to understand UMLs visual syntax.
2. Ability to drive the CASE tool to navigate around the model.

In order to make an assessment of comprehensibility and accessibility, we also need to define *who* we are considering, and *what* it is that they are trying to comprehend.

If you consider stakeholders that could benefit from access to UML models, then for the purposes of this consideration, you can divide them up into six broad categories as shown in Table 1:

TABLE 1.

Role	Semantics	Type
Non-technical manager	A project management role. Needs only a very high-level understanding of the technical aspects of the project. In particular, they do not need to have UML knowledge.	non-technical
User	Someone who uses the delivered system. Users do not need to be technical, but may have some knowledge of analysis or requirements capture.	
Domain expert	An expert in the problem domain of the project. Domain experts do not need to have any UML knowledge.	
Analyst	Creator of analysis level UML models.	technical
Designer	Creator of design level UML models.	
Programmer	Creator of source code.	

These categories of people can be thought of as very broad roles within the project, and any individual project participant may play more than one of these roles at any point in time, or over time.

Finally, you need to consider the various types of UML diagrams (class diagram, sequence diagram etc.), and how accessible and comprehensible these are to the various roles.

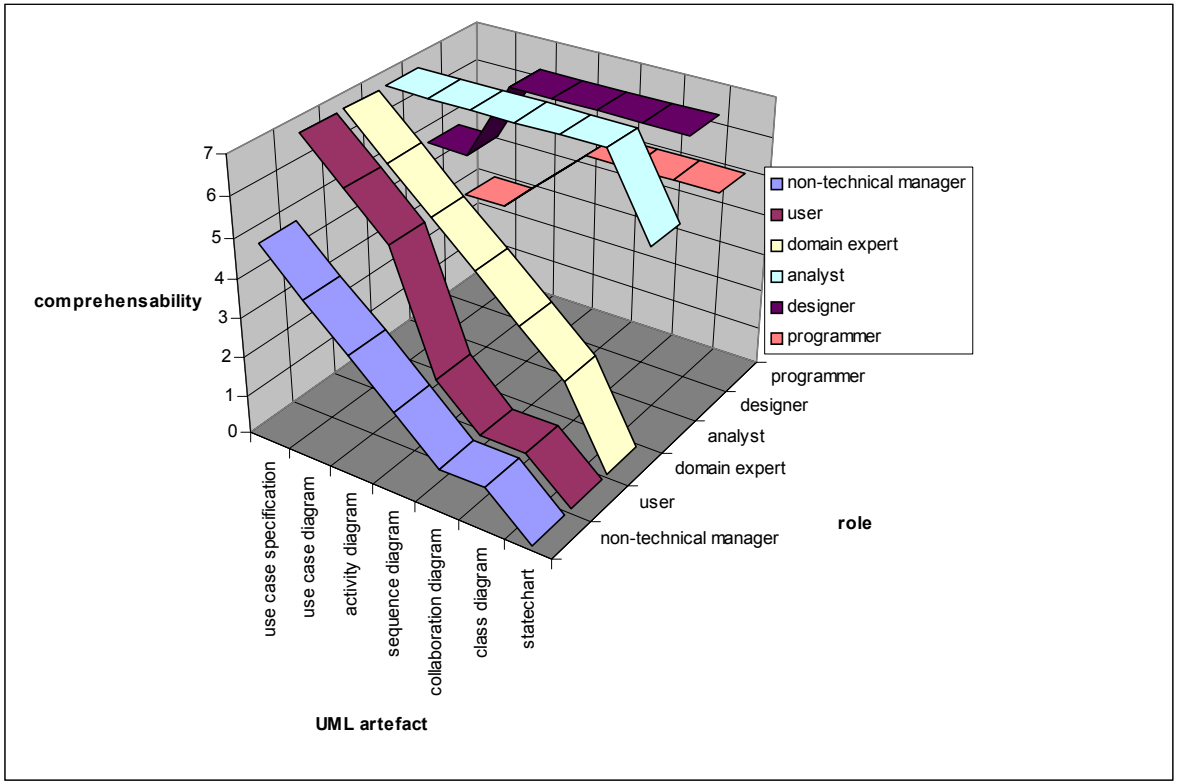
In fact, each of the various types of UML diagram targets, and is comprehensible to, a limited audience.

Figure 2.1 shows our estimates for the comprehensibility of the main UML artefacts in an analysis level model. We do *not* consider design models, or physical

models (deployment and implementation diagrams). This is because our focus in this book is on using literate models to convey business information. It is certainly possible to use literate modelling at more concrete levels of abstraction, but we feel that it's main benefit is at the analysis level.

For each of our roles, we have rated their comprehension of the various UML artefacts on a scale of zero to seven. On this scale zero means virtually no comprehension seven denotes virtually complete comprehension. We arrived at these estimates based on our private communications with many individuals performing roughly these roles over the course of many different UML projects in many different businesses. The chart is not (and we're not sure it ever could be) quantitative, but we hope that you can agree that it illustrates the trends in comprehension correctly.

FIGURE 2.1 **P001**



In the next few sections, we'll look at each of the UML artefacts and their comprehensibility in a bit more detail.

2.3.1 Use case specifications

Considering all roles, use case specifications have the highest overall comprehensibility. This is because:

- They are usually written in plain English and so there is no comprehensibility problem due to a need to know a visual syntax.
- They are often written in a word processor, as support for use case specifications in the current crop of CASE tools is generally limited to plain text. There is no access ability problem as no CASE tool is involved.
- They are often somewhat familiar to non-OO practitioners, as they are just descriptions of business processes from the point of view of the actor. It is quite easy to "step inside" the use case specification and role play in order to enhance comprehension.

Despite this high level of comprehensibility, there can still be some problems with use case specifications:

- A use case is a description of a specific business process from the perspective of a particular actor. As such they typically don't give a clear picture of the *overall* business context and imperatives that generate the need for the business process in the first place.
- Use cases are often written using domain specific jargon. This means that they can sometimes be quite incomprehensible to non-domain experts. As we discuss in [Arlow1], there are specific ways to get around this issue. Because of this need for a certain amount of domain knowledge, the more technical Designer and Programmer roles may have problems understanding the real business meaning of some use case specifications.
- The business context that gives rise to a set of business requirements is not well captured or explained by use cases or by *any* UML construct.

Unfortunately UML provides no *formal* mechanism to capture and present important contextual information. You can't easily embed this information in the use case specifications themselves as the business context is generally orthogonal to any particular use case. You can always use notes, free-text annotations to diagrams, and constraints. This may help you to capture the contextual information in the

model, but in terms of the accessibility and comprehensibility of the model it doesn't really help.

2.3.2 Use case diagrams

You will find that find these to have similar comprehensibility to use case specifications. UML use case diagram syntax is very simple and this leads to high levels of comprehensibility. However, the following features may cause some comprehensibility problems:

- use case generalization
- «extend»
- «include»

Use case generalization is not that widely used (unless the parent use case is abstract), largely because it's effect on use case specifications can be very complex. See [Arlow1] for a discussion of this.

Of the other two relationships, «include» is easy to understand for anyone who has some background in programming, and «extend» can be explained even to non-technical users if the explanation is clear enough. Again, we refer you to our previous book [Arlow1] for simple explanations of these relationships.

Use case diagrams are semantically very weak. You may find that the real business meaning of a use case diagram is not apparent without detailed explanation, or reference to the use case specification itself. We have therefore given them a lower comprehensibility than Use Case Specification for the non-technical roles, Non-Technical Manager, User and Domain Expert.

You may find that comprehensibility is lower for the technical roles, such as Designer and Programmer, as they may not have sufficient business domain knowledge to grasp the true business meaning of the use case diagram.

2.3.3 Activity diagrams

One of the nice features of activity diagrams is that you can use them for almost anything! They can model use case flows, business processes or even the detailed specification of a method. Because we are focusing on the analysis domain, we will only consider two uses for activity diagrams here - modeling use case flows and modeling business processes.

Essentially, activity diagrams are just “OO flowcharts”. Most people are familiar with flowcharts, and so they tend to have high levels of comprehensibility. We have positioned them on our chart as having slightly lower comprehensibility than use case specifications and use case diagrams because there is significantly more visual syntax to learn for activity diagrams.

In most other respects, activity diagrams exhibit similar comprehensibility across the roles as use case diagrams and specifications. The key difference is that activity diagrams tend to be slightly more comprehensible to the technical roles and less comprehensible to the non-technical roles.

2.3.4 Sequence diagrams

You are now in the realm of object orientation, and comprehensibility falls sharply for non-OO literate participants. We have found that Non Technical Managers and Users find raw sequence diagrams very difficult to follow because they don't really understand the details of object interaction.

Comprehension may be slightly higher for Domain Experts, as these roles often have some exposure to object-orientation through working with Analysts.

If you adorn sequence diagrams with scripts, this will increase comprehensibility markedly for the non-technical group. But comprehensibility is now of the script, rather than of the visual syntax, which remains largely obscure.

Because sequence diagrams show the interaction between objects, Designers and Programmers tend to naturally understand them. However, they might not be so sure about the underlying business processes that drive the interactions!

2.3.5 Collaboration diagrams

Non-technical roles typically find these confusing, and unlike sequence diagrams, there is no reasonable possibility for you to adorn them with a script to increase their comprehensibility. We give these a very low comprehensibility for this audience although again, comprehension may be higher for the Domain Expert.

The technical roles find these diagrams both useful and comprehensible.

2.3.6 Class diagrams

For comprehensibility these require:

1. Some basic OO training
2. Knowledge of UML syntax
3. Ability to use the CASE tool to uncover class and relationship semantics

We have found that comprehensibility of these diagrams is typically very low for Non Technical Managers and Users. It may be slightly higher for Domain Experts, as these roles often have some exposure to object-orientation through working with Analysts.

For the technical group, Analysts, Designers and Programmers, comprehensibility is very high, although we have noticed that many programmers do not have sufficient understanding of UML syntax and object oriented analysis to fully appreciate them. Hence, you may find that the Programmers' comprehension is lower than that of Analysts and Designers.

Analysts tend to understand the class diagram from the business perspective.

Designers often know less about the business, and their comprehension may be more in terms of object-oriented design issues such as patterns, idioms, APIs and technical infrastructure.

In many organizations Programmers tend to be more junior than Analysts and Designers. As such they may know little about the business and little about good object-oriented design principles. This leads to a lack of comprehension of many of the key aspects of the UML model.

2.3.7 Statecharts

Statecharts are quite specialized and have a very elegant yet terse syntax that is rarely understood by the non-technical group. On our scale, comprehensibility is effectively zero for this group.

Generally, we have found that it is Designers, and not all Designers at that, who have a good grasp of statecharts.

The problem is that statecharts attempt to capture a dynamic system in a static notation. This obviously makes them quite hard to understand as it is left up to the reader to imagine the dynamic flow between states. This can only happen if the reader understands object interactions.

Statecharts increase in comprehensibility if they can be executed and animated in the CASE tool. But this is still quite rare.

2.4 *The problem of comprehensibility*

You can see that several important issues arise from the above discussion:

1. Moving through our seven diagrams along the UML artefact axis of Figure 2.1 from use cases to statecharts, the non technical group is gradually left behind. They lose comprehension as the diagrams become more technical and the emphasis shifts from a focus on business requirements to a focus on the intricacies of implementation
2. There is a traceability issue. The non-technical group understand the business requirements best, but they have little comprehension of UML Sequence, Collaboration, Class and State diagrams. Traceability of high level requirements to these diagrams therefore relies mainly on the fidelity of the modelling transformations, and lacks essential feedback from the non-technical group.
3. We have found that Designers and Programmers may have little understanding of the actual business and its needs. You cannot rely on them to capture business requirements correctly in their models and code.
4. Key business requirements are expressed as elements in UML diagrams. But there are so many elements in a UML diagram that the important requirements become lost. We call this process *trivialisation*, because key requirements are translated into a context in which their importance is no longer apparent.

The last point, about trivialisation of requirements is very important, and we discuss it in more detail in the next section.

2.5 *The trivialisation of business requirements in visual modelling*

We all know that some business requirements are more important than others. But often, you can't tell from a blunt statement of the requirement just how important it is to the overall operation of the business. In order to appreciate the true importance of a requirement, you need to see it in it's business context, but it is precisely this business context that is lacking in conventional UML models.

In the real world, you may notice that important business requirements are often highlighted by a certain amount of activity and ceremony - there may be papers, working groups investigating the requirement and discussion at managerial level. This activity is a key indicator that something is perceived to be important to the business by those in charge.

But all of this valuable contextual information is absent from the UML model. Although you may have a statement of a particular business requirement as part of a UML use case, you have no formal mechanism to highlight the importance of this requirement or to set it in its true business context.

Worse, when the requirement is expressed in a class diagram, it becomes merely a cluster of modelling elements much like any other.

Rather than being *highlighted* in the UML model, essential business requirements tend to fade into the background. This is what we mean by trivialisation.

In our paper [Arlow1] we present the following example from British Airways that illustrates trivialisation.

The last decade has been the decade of the global airline. Globalization often involves forming alliances so that one partner may sell seating capacity on another partner's flight. This practice is known as codeshare.

Codeshare is good for the airline, as it extends it’s network, and it is good for passengers, as they can complete a complex journey using a set of co-operating companies. It can also improve customer service. In fact, codeshare can generate new business worth millions of pounds.

For example, a single operating flight from London, Heathrow to EuroAirport (Basle, Mulhouse, Freiburg) on 23 July 2002 at 21:00 may have a BA flight number (BA6670), a Swiss flight number (LX371) and be operated by Swiss. It will also have an *operational* flight number that is the “real” flight number as far as air traffic control is concerned.

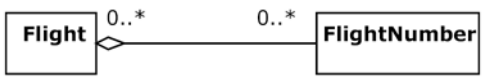
You can see that it is an essential business requirement for alliance partners to be able to support codeshare in their systems. The key to this support is that each flight must be able to have *many* flight numbers.

But how do you represent this key business requirement in a UML model?

From the use case perspective, it's not entirely clear where the requirement gets captured. There will be a use case involving a BA customer flying on a BA flight. But it is unlikely that there will be any mention of codeshare in this use case as the principle of codeshare is that it is meant to be *transparent* to the BA customer.

In the class diagram codeshare is represented as a many to many relationship between Flight and FlightNumber as shown in Figure 2.2.

FIGURE 2.2 001



So a multimillion-pound business requirement, effecting an alliance of companies together worth billions, is represented as a multiplicity on a UML Analysis class diagram!

This sort of trivialisation is surprisingly common when you begin to recognize it.

2.6 *Literate modelling*

As you saw in the last few sections, although UML models can have a high degree of precision and conciseness, they may be difficult to access and comprehend - especially by non-technical people. Literate modeling provides one solution to this problem.

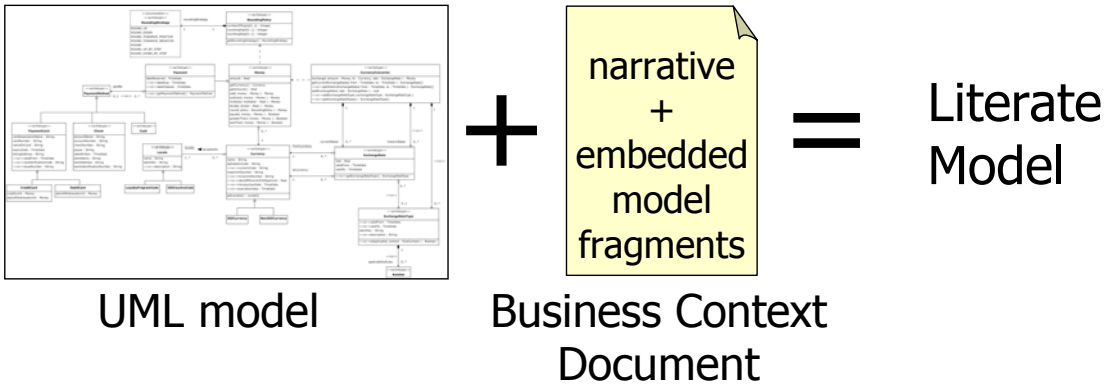
Literate modelling applies of Knuth's idea of Literate Programming [Knu84] to UML models. The approach is very simple - you interleave UML models with a narrative text that explains the model to both the author of the model and to all the roles discussed above.

Literate Modelling addresses all of the issues we have raised: the accessibility and comprehensibility of the UML models and the trivialisation of business requirements. It does this by providing the missing business context in the form of a document that anyone can read.

So the core idea of literate modeling is very simple - you simply extend your UML modeling by adding new documents that we call Business Context documents that

explain the model in light of the business context and forces that have shaped it. This is illustrated in Figure 2.3.

FIGURE 2.3 P002



You can use business context documents to:

1. Explain the rationale behind the UML model in terms that business users can understand
2. Highlight important business requirements
3. Map important business requirements to specific features of the model
4. Explain how the business requirements and context caused particular modelling choices to be made

In our experience the literate model increases the business value of a UML model by making it accessible and comprehensible to a very wide audience.

2.7 *The business context document*

In this section, we discuss how to write a business context document.

Business context documents discuss the background, general principles and concepts, essential requirements and the forces that shape a specific part of the busi-

ness. They consist of a narrative with embedded UML diagrams. Any description of any part of these diagrams is *always* from the perspective of the business.

A good Business Context can be quite difficult to write. As the author, you need to have quite a range of knowledge and skills. You need:

- A very sound and broad overview of the business
- Good UML modelling skills
- Good writing and communication skills

We have had very good experiences applying the techniques of Nauru-Linguistic Programming (NLP) when writing business context documents. NLP provides a model of communication and a set of specific communication techniques to improve the quality of communication. A full discussion of NLP is outside the scope of this book, and we refer you to "The Structure of Magic" [Bandler1] for more information.

You have two options for structuring your business context document:

1. Around the things in your business
2. Around the processes in your business

Our experience is that structuring the business context document around the things in your business (Customer, Product, Order etc.) is the best approach. This is for the following reasons:

- Things, and their relationships to each other tend to change quite slowly. This is *particularly* true if the things in question are business archetypes!
- Things tend to naturally form cohesive clusters (e.g. Customer, Product, Order) that provide a very good focus for the business context document.
- Things support processes (and processes require things) and so things are in some sense more fundamental than processes.

On the other hand, buisness processes tend to change quite rapidly and cut across clusters of things. You can document important business processes as a narrative, but you should use the more stable business context documents as the building blocks for this narrative.

The first step in creating a business context document is to identify a suitable focus for the document. This focus is a cohesive cluster of things that deliver value to your business. We generally name the business context document after the key

thing in that cluster. For example, if you consider the Money Archetype Pattern presented in Chapter 6, you can see that the cluster of things (archetypes in this case) are:

- Money
- Currency
- ExchangeRate
- MoneyCalculator
- Locale
- etc.

The key thing is Money, and so in this case we call the business context document "The Money archetype pattern".

Each business context document has the following minimal structure:

- Business context
    - A general discussion of the business context that this document describes
- Compliance to standards
    - Existing standards that anyone working in this area needs to know about
- Overview
    - Overview UML model showing all of the main things and relationships
- Thing
    - Narrative
        - Model fragment
    - ...
- Thing
    - Narrative
        - Model fragment
    - ...
- ...

We find that the Class Diagram, Use Case Diagram and Sequence Diagram are quoted most often in the Business Context document. In rare cases, you may find it useful to include a few State Diagrams for the more technical readers.

You can use *informal* diagrams wherever they enhance the text, but they should never be a substitute for a UML diagram.

The structure shown above is not fixed, and you can add things to it or remove things from it as you see fit. However, the core semantics of the business context document - that it explains the model in light of the business context and forces that have shaped it - must always be preserved.

One key advantage of the business context document is that it can begin to regularize the language used in a particular business domain. To achieve this you should highlight definitions of things and terms, as we do in the literate models presented later in this book.

You will find that most businesses use terms quite loosely. For example, airlines often use the term "flight" to mean four distinctly different things:

- A specific aircraft flying between an origin and a destination at a particular point in time
- A group of aircraft flying between an origin and a destination over a period of time
- A marketing entity that describes travel that may be realised by one or more physical flights between an origin and a destination beginning and ending at particular points in time
- A marketing entity that describes travel that may be realised by one or more physical flights over a period of time

This is an example of a homonym - a single word that has a cluster of different meanings.

The other case you will encounter is the synonym. This is where two different terms have the same meaning. For example, many business with web sites use the terms 'customer' and 'user' to mean the same thing.

Synonyms and homonyms are a reality of business life, yet you can (and must) resolve them in your business context documents. You can mention synonyms and homonyms in your narrative, explaining why you have chosen one term in preference to another. You can also create a glossary to go with your business context documents where there is a single entry for each preferred term with all synonyms and homonyms listed underneath.

If business context documents had no other benefits, they would be still be worth creating just because they introduce the possibility of regularizing business terminology!

2.8 *Clusters of things*

The UML grouping mechanism is the package, and in a well-constructed UML model you will find that packages contain cohesive clusters of things. This tells you that there should be a simple relationship between the your business context documents and the package structure of your UML model. In the simplest case, there is one business context document per package. However, it is also quite common for a business context to describe a cluster of closely related packages such as a package and its nested packages.

Just as there are dependencies between packages, there are corresponding dependencies between business context documents. A client document will often have to refer to a thing in a server document. You can resolve this as follows:

- You should always include the definition of the thing you are referring to in the client document
- You should always reference the server document where the thing is covered in detail

Replicating the same definition in different documents is clearly a bad idea from a maintenance perspective. However, from a readability perspective it is essential that your reader has all the information *at their fingertips*. Only make your reader go off to another document if it is absolutely necessary. Word processors such as Word or Framemaker allow you to put commonly used text, such as definitions, in a library where you can reuse them. This reduces your maintenance overhead.

If you find that your business context documents imply a *different* package structure to that of your UML model, then you need to resolve this. Go back to the business and find out what the *true* clustering of things is. You will find that the truth *is* out there.

2.9 *Business context document conventions*

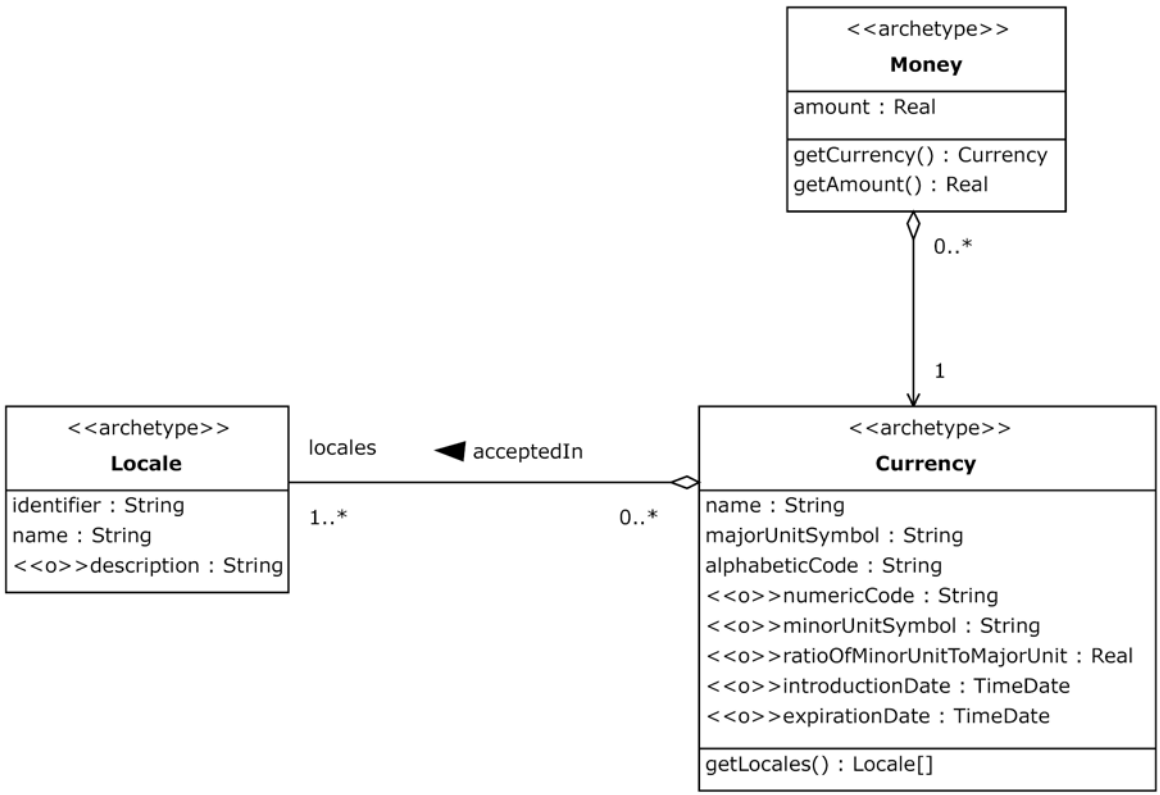
Always write Business context documents using the names of the things in the UML model. In this way you tie the narrative to the UML model. These names may (as in this book) be highlighed in a specific typeface.

Using model element names directly in the text provides a very stringent test for the quality and comprehensibility of your model. Parts of the narrative that are domain experts find hard to understand may indicate where you have named something poorly or even where you might be using the wrong abstraction.

Consider the following extract from the Money archetype pattern (section 6.4):

"The Money and Currency archetypes are shown in Figure 8.2.

Figure 8.2002



Money is an amount of a specific Currency. This Currency is accepted in one or more Locales. We'll come back to Locale later."

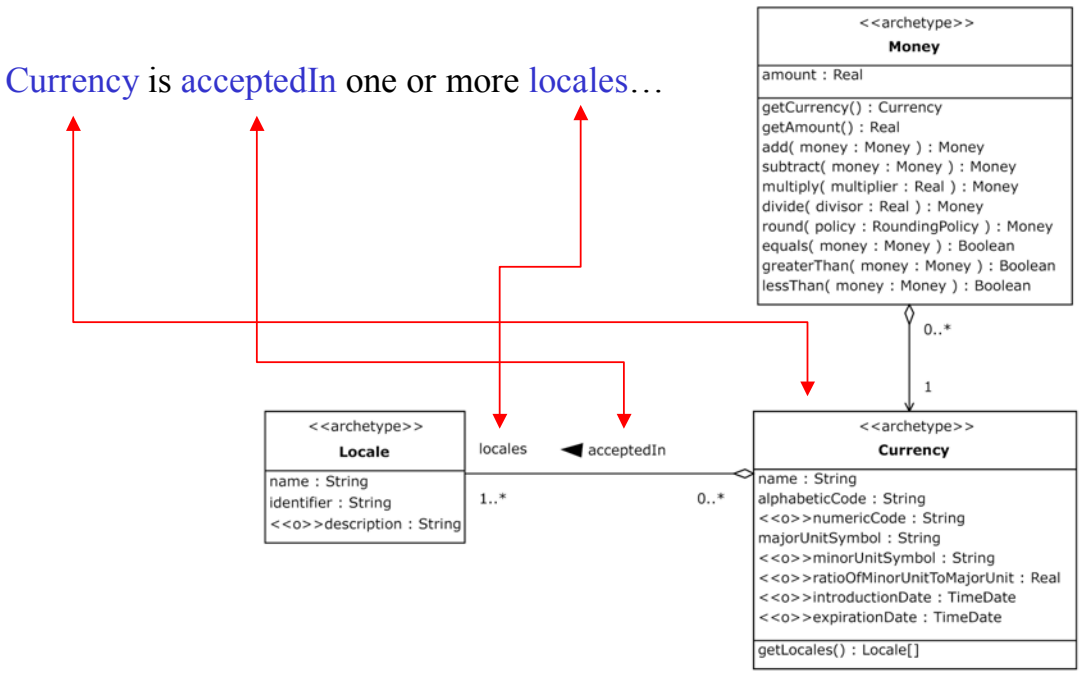
Notice the following points:

- The text refers to one or more specific UML model elements (see Figure 2.4).
- The names of all model elements are written in a special font. You can refer to any model element on the diagram that has a name.
- Use plurals where necessary. So if you need to talk about more than one Locale, you use the term Locales.
- Use 's appropriately. So, for example, you can talk about a Country's Locale.

- The text reads well and is comprehensible whether the UML model is present or not.

The last point is very important. The text should be comprehensible whether you can read the UML model or not. In fact, a good test of a business context document is to cover up all the UML models and see if it is still readable. It should be!

FIGURE 2.4 P003



2.10 *Readability*

We encourage you to believe that boredom is always optional!

Ever wondered why some texts put you to sleep? It's probably down to the use of passive voice.

Passive voice disassociates the reader from the story line of the document and leads to boredom and low comprehension. Dr. Richard Bandler (one of the world's greatest hynotists) pointed out in [Bandler1] that using the passive voice is one of the best ways of inducing trance in a reader.

Combining passive voice with a long, rambling and ambiguous sentences can be truly devastating! In fact, if you *want* to write something that no-one will ever really read, then this is a sure-fire way to achieve that.

Our observation is that a surprising number of corporate documents are:

- Written in passive voice
- Have long, rambling ambiguous sentences
- Are never read

Our advice to you is to write the business context document as though you are talking to someone - much as we have written this book. Try to make the document as engaging as possible. Given the subject matter you are unlikely to ever make the New York Times best-seller list, but you will make your documents more readable and therefore useful.

Perhaps the most important tip we could give you is simply to "tell a story".

Your UML diagrams should always tell a story and so should your business context documents. This story explains to the reader, in simple language, how part of the business operates and why it operates in that way. The literate model should explain and highlight important business things, processes and requirements.

We find that we get the best results when the Business Context document is lively, involving, direct, provocative, precise, concise and, if possible, humorous. However, it can be difficult to incorporate humour well and you should avoid it if in doubt.

This isn't really the place to discuss good writing style in depth, so we refer you to "Bugs in Writing" [Dupré1], for more information.

2.11 *Use concrete examples*

It's important to make the business context documents as concrete and "real world" as possible.

One of the best ways to do this is to include real-world examples throughout the text. For example, if you are discussing product specification, then give a real business example in your text.

If you are working in an alliance situation, where your literate models may also be used by business partners, then consider giving some examples from your partners business. We have done this several times and it is always very well-received. It shows your partners that you have been taking their needs into consideration and it helps to bypass the "not invented here syndrome".

Another tactic that we find useful in business context documents is to include more than one example in areas that are difficult. You might start with a simple example to give the reader the general idea and then go on to a more complex example. In one literate model we created for a global transportation company we chose the most complex journey in their timetable and demonstrated how our UML model could accommodate that. This gave our literate model a *lot* of credibility within that business.

Apart from the credibility issue, another very good reason for choosing a difficult example is to stress-test your UML model. We have often found that something that works for a simple case breaks down for more complex cases. In fact, many of the UML models you find in UML text books will only work for very simple cases. You can greatly improve your modelling skills by providing worked examples that illustrate how your model support complex real-world business situations.

2.12 *Precision and correctness*

Combining a narrative text with a UML model gives you something that is more than the sum of it's parts.

The reason for this is that the narrative is free to explore the entire business context in which the UML model must operate, whilst the UML model enforces precision

on the narrative. We often find that a good literate model is much more detailed and precise than either the model or the narrative would be if they stood alone.

Also, both the UML model and the narrative tend to be correct when combined together in a literate model. This is because the UML model highlights errors in the narrative and the narrative highlights errors in the literate model. You may be surprised at how often a UML model that looks OK on paper begins to seem not so OK when you begin to write about it!

2.13 *The future of literate modeling*

In the future, the most significant enhancement we can make to literate modelling will be to define an XML schema for the business context document. This schema will incorporate special tags containing information that links the tag contents directly to elements in a UML model. In fact, you can do most of what you need to do in literate modeling with only three new XML tags:

<modelElement>
<modelElementDefinition>
<umlDiagram>

For example, the tag <modelElement> might look something like this:

<modelElement pathName = nameOfElement, identifier = identifierOfElement>
someText
</modelElement>

This tag contains the path name of a model element and (optionally) its unique identifier in the UML CASE tool. The content of the tag should be the name of the element as used in the narrative. For example, if the path name of the element is clearviewtraining.com::Money Archetype Pattern::Currency, then at one point in the narrative, someText might be “Currency” and in another part, “Currencies”.

The <modelElementDefinition/> tag is used as follows:

<modelElementDefinition pathName = nameOfElement, identifier = identifierOfElement/>

This tag facilitates the automatic retrieval of documentation strings from the UML model and their insertion into the literate model.

Finally, we need a tag to identify embedded UML diagrams:

<umlDiagram pathName = nameOfDiagram, identifier = identifierOfDiagram/>

This tag just refers to a specific UML diagram in the model.

They key design point of all of these tags, is that the UML model is the source of information that is embedded into the business context document. This means that the business context can be considered to be a rich view of the UML model. We believe that this is the right approach.

As well as these literate modelling specific tags, you also need tags for things like paragraphs, tables etc. A commonly used XML schema for documents is the DocBook schema (www.docbook.org). This provides an excellent base for the literate modelling extensions.

This XML representation would allow a degree of automation in the creation of literate models. It would also allow for consistency checking between the business context documents and the UML models.

Another exciting possibility is that the literate models themselves could provide the user interfaces for the archetype automation technology that we describe in chapter 2. We may be doing some work on this in the near future.

2.14 *Summary*

Literate modelling has been *very* successful for us. It has improved the quality of our UML models and our ability to communicate with stakeholders. We think it will work well for you also. We encourage you to try it!

- Literate Modelling was invented by Jim Arlow (Clear View Training Limited), Dr. Wolfgang Emmeric (University College London) and John Quinn (British Airways) in 1998
    - Similar to Literate Programming [Knuth1]
    - Arose from our attempts to explain UML models to non-technical audiences
- Issues with visual models

- Valuable business information is encoded in a visual syntax only accessible to those "in the know"
  - Comprehensibility - those who can understand the visual syntax
  - Access ability - those who can use the CASE tool
- Trivialisation of business requirements
  - Important requirements often become invisible when expressed in UML
    - Lost amongst a host of visually similar modelling elements
  - The business context that generated the requirements is missing from UML models
  - Analyst/designers may not remember the business rationale for a model they created only a short time ago

- Comprehensibility and access ability of UML models
  - Comprehensibility - the ability to understand the business semantics of the model
  - Access ability - the ability to access the information contained in the model
    - Ability to understand UML
    - Ability to work the CASE tool
  - See figure Figure 2.1
  - The problem of comprehensibility
    - As diagrams become more technical (use cases to statecharts) the non-technical group are left behind
      - Traceability
    - Designers and programmers may have little understanding of the business
      - Trivialisation (see above)
- Literate modelling
  - Extend your UML model with a business context document that captures the missing business context of the model
    - Explain the rationale behind the UML model in business language
    - Highlight important business requirements
    - Map business requirements to specific features of the UML model

- Explain how the business context shaped the model
- Literate models are more valuable than UML models alone
  - High comprehensibility
  - High access ability

- The business context document
  - Discusses the background, general principles and concepts, essential requirements and forces that shape a UML model
  - Written from the business perspective
  - Structured around the things in the business
    - Things and their relationships change quite slowly (especially archetypes!)
    - Things naturally from cohesive clusters - ideal for structuring the business context document
    - Things support processes
  - How to create a buisness context document
    - Identify a suitable focus (a cluster of things)
    - Name the document after the central thing
    - Structure
      - Introduction
      - Compliance to standards
      - Overview
      - Thing
        - Narrative
        - Model fragment
      - ...
      - Summary
    - Use informal diagrams where they enhance the text
      - Never substitute an informal diagram for a UML diagram
    - Synonyms - two or more words mean the same thing
    - Homonyms - the same word has two or more different meanings
    - Always choose a single term and define it

- Deal with synonyms and hominess using a glossary

- UML packages may provide the organising principle for the business context document
  - A single package or cluster of closely related packages may provide the basis for a business context document
  - Interpackage dependencies become dependencies between business context documents
    - When a client document refers to a thing in a server document
      - Always include the definition of the thing in the client document
      - Always reference the server document
      - Don't make your readers look things up!
  - If the business context document implies a different structure than the UML model, go back to the business to resolve this
- Business context document conventions
  - Always use names of model elements in the narrative
  - Write model element names in a special font
  - Use plural where necessary e.g. Locale may become Locales
  - Use 's where appropriate e.g. Country's
  - Check the narrative reads well even if the UML diagrams are covered up
  - Readability
    - Don't use passive voice
    - Tell a story!
    - Style should be lively, involving, direct, provocative, precise, concise and (if possible) humorous.
  - Use concrete examples
    - Always give your reader real-world examples that are meaningful to them
    - Use simple examples to get an idea across
    - Use complex examples
      - Show that your model really does support the business requirements
        - Stress-test your model

        - Work business partners into the examples
    - Precision and correctness
        - A literate model will be more precise and correct than a UML model alone
        - A literate model will be more precise and correct than documents about the business area alone
- The future of literate modeling
    - An XML schema for literate modeling.
        - <modelElement pathName = nameOfElement, identifier = identifierOfElement>someText</modelElement>
        - <modelElementDefinition pathName = nameOfElement, identifier = identifierOfElement/>
        - <umlDiagram pathName = nameOfDiagram, identifier = identifierOfDiagram/>
    - Using the literate model as a user interface for archetype automation.