Systems analysis and design – data driven and procedural techniques

2.1 Introduction

2.1.1 Introduction to modelling

Computing is a practical subject and any study of it should include both education and training. We have tried to follow this philosophy in this unit. It is not really possible to fully understand the techniques used in analysis and design without some practical experience, and we hope the exercises and explanations included in this unit go some way to providing that experience. We hope that you will undertake the exercises in collaboration with the other students and discuss your views of the course with them and your tutor. Systems development is a collaborative activity, and discussions with one's peers always improve the quality of the analysis and design. Constructive review and discussion about what you produce is one of the most important development techniques you can learn, and we encourage you do work together throughout the course, and require you to do so in the assignment for the final unit.

System analysis and design

We need to define what we mean by the terms 'system', 'analysis' and 'design'. On this course, by **system** we mean a computing system. (You may prefer to think of it the other way round – as a system that uses computer technology.) The purpose of the system is not the computer technology but the function it supports, for example a sales system or a system to control aircraft in flight. In both examples the function is managed by a mixture of human and computer control. We are considering what can be defined as a human activity system. To analyse and design such a system we have to take a holistic (all encompassing) approach and not just concentrate on the technical analysis and design but address the whole system including the human components. On occasion, it may be more effective to address work practices rather than to develop an expensive technical solution. Overall, the system provides the context of the analysis and design.

The purpose of **analysis** is to identify *what* the system has to do, which is achieved by investigation and modelling. Investigation generally focuses on two main areas – the system requirements and the way the current system works – in order to gain perspective on what is done and what is needed. Analysis seeks to validate the requirements so that they are properly established and defined. It must ensure that the requirements are defined to a level suitable for design to take place and ultimately to permit user acceptance testing. The main deliverable of analysis is the requirements specification, which will contain a full definition of the requirements and logical models of the required system.

The purpose of **design** is to specify *how* the system will work, which is achieved through research and investigation into possible solutions and the eventual design of an agreed solution that includes both logical and physical designs. Design will use modelling techniques to develop the final specification. Some of these techniques will involve drawn models whilst others will use physical development such as prototyping. The main deliverable of design is the system specification which will include detailed definition and models. It must be sufficiently detailed and scoped to form the basis of implementation. It is essentially a blueprint for the system.

At the macro level, design follows analysis as a stage in the typical development cycle of analysis, design and implementation. The dividing line between them is not always clear, however, and they shade into one another at various levels. As activities and techniques, they are intimately associated, and they support and complement each other in all areas of the development. For example, in requirements gathering, the *analysis* of the requirements often considers possible solutions and their design, in order to elicit its full meaning and definition. At the same time, the development of a solution at the design stage often identifies the need for further analysis in order to scope fully the options available to the designer.

The logical view and conceptualisation

In terms of system definition, there is essentially an arch going from the physical, to the logical and back to the physical, spanning the analysis and design stages. A new system replaces an existing system and at the beginning of analysis the existing system is analysed and a physical model developed. This physical system model is then conceptualised into a logical model. The reason for doing this is that the physical reality of the system can often obscure and corrupt the way the system should logically work. In abstracting the way the system should work, logically, we are able to understand the principles underlying the system and its true workings. This logical model is then used, together with the requirements, to produce the requirements specification. Based on the requirements specification, a logical design is developed. In developing a logical design, a model of how the system should ideally work is developed. This model forms the basis of the physical design, which accommodates any limitations of the software and hardware the system uses. Finally the physical design is specified. The conceptualisation of the system and its logical purpose are therefore central to the analysis and design.

Evolution and complexity

Very much like the evolution of life systems analysis and design should develop from the simple to the complex. It is difficult to produce a detailed specification if the broad outline and scope of the problem has not already been defined. The broad principle is always to go from higher level modelling to lower level modelling. This enables the reassessment of previous work as at every level more detail is developed. It also emphasises the iterative nature of analysis and development at all levels. Constant revisiting while moving forward is the key. Many of the processing techniques we will look at allow for decomposition. That is the breaking down of a process into lower level processes, and then the breaking of those processes down still further. Data techniques also build up their models through many iterations.

Why model?

A model is a representation of the real world: either as it is or the as it is intended to be. During its construction it should act as a focus for ideas and prompt the asking of relevant questions, research and creativity. A good analysis and design modelling technique is a powerful tool with which to promote correct actions and produce deliverables of high quality. As noted above, the analysis and design evolve and a good modelling technique prompts the asking of the right questions at the right time and level.

The modelling techniques we will look at on this course are widely used in analysis and design and are well proven, having been tuned and improved continually over time. They are also based on a broader set of underlying analysis and design principles that have evolved from hard-won experience, gained through many generations of systems developers. The fact that these techniques embody such experience is another reason for using them. Models are good vehicles for communication. They encourage focused thought about the things they model and prompt feedback and guidance from users and colleagues. You have to be careful which models you present to users. They may be too technical for some users who may find them daunting. You need to pick the correct model for a particular set of users.

Analysis, design and development methods

Development methods are based on analysis, design and implementation techniques. They take a particular approach to the development cycle, for example a waterfall or iterative approach, and then include a defined set of modelling techniques. Some methods, such as the Structured Systems Analysis and Design Method (SSADM), are relatively prescriptive in the techniques that can be used and the way they are used. Other methods, such as the Dynamic Systems Development Method (DSDM), are frameworks that outline how development should proceed, whilst suggesting a range of techniques and allowing the developer to decide which is the most appropriate. Other methods focus on particular approaches to design and implementation, such as the object-oriented Unified Process using the Unified Modelling Language (UML), which we consider in Unit 4. These object oriented approaches to analysis and design are in fact the methods you are most likely to use to model new systems. It is, however, important to have an understanding of the more traditional approaches to analysis and design. This unit explores some of the methodologies associated with the Structured Systems Analysis and Design Methodology (SSADM). This methodology is associated with the waterfall approach, and like that approach it has lost favour in recent years. There are, however, good reasons why a software engineer should be familiar with SSADM, including:

- Much software engineering involves adapting or extending existing systems, the documentation for which will often be presented as SSADM models.
- Even where a system is fully object oriented, it is often necessary to provide storage using a relational database, which needs to be based on a traditional entity relationship model.
- The proponents of cutting edge development systems - Rapid Application Development (RAD), Agile Development and Extreme Programming - sometimes produce ephemeral data flow models in the early stages of analysis.
- It is still seen by employers as a necessary or desirable skill. You can get some idea of this by searching pages added during the past week (use advanced search) using the keywords *SSADM Job*, and then the keywords *UML Job*. The result varies from week to but the 62,000 hits for the former and 39,000 for the latter I just turned up is typical.

CASE and drawing tools

The methodology covered in this unit involves producing structured diagrams using a defined set of symbols and conventions. We do not specify any particular tool for these tasks, indeed we expect many people will use the drawing facilities of Word or PowerPoint. Diagrams produced by these applications are entirely acceptable. If, however, you already have a drawing package like Visio or SmartDraw, you will find the diagrams take less time and trouble to produce.

Professionals use CASE (Computer Aided Software Engineering) applications that not only make drawing easier, but also manage the drawings and link then to other parts of the analysis and design. They are generally expensive and certainly not worth contemplating for this course. One possible exception is a tool called Ascent which is a lightweight, but inexpensive, tool designed for educational use.

But we stress: there is no need to use a CASE tool or a specialist drawing application. The facilities provided by Word and PowerPoint (or their Open Office equivalents) are entirely adequate.

2.2 Data Flow Models

2.2.1 Introduction to data flow diagrams

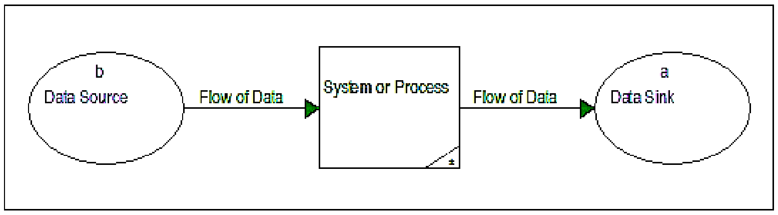
The data flow model plays a central role in legacy analysis and design methodologies. Each methodology has its own symbols and rules, and some are more formal than others, but in essence they model the same things and the diagrams are very similar.

Data flow diagrams were traditionally used to document the analysis of the system and provide an overview of the proposed solution. In the interests of avoiding ambiguity such diagrams were drawn up in accordance with strict rules and conventions and were presented as formal documents. Computer Aided Software Engineering (CASE) packages were used to create and manage the diagrams. Such formal diagrams are far from out of date. Much software engineering involves the updating and adaptation of existing systems and the practitioner needs to be able to understand the legacy system documentation.

When designing new products, however, it is unlikely that data flow diagrams would be used to define the system. The requirements are more likely to be expressed in terms of use cases and the data flows in terms of messages between objects. The data flow diagram has not, however, disappeared from the analyst's toolkit. It is still a useful tool and one used even in paradigms such as Agile and Extreme Programming. The difference is that, instead of being part of the permanent documentation, it is seen as something ephemeral, to be drawn quickly on a white board, to provide insights into the system.

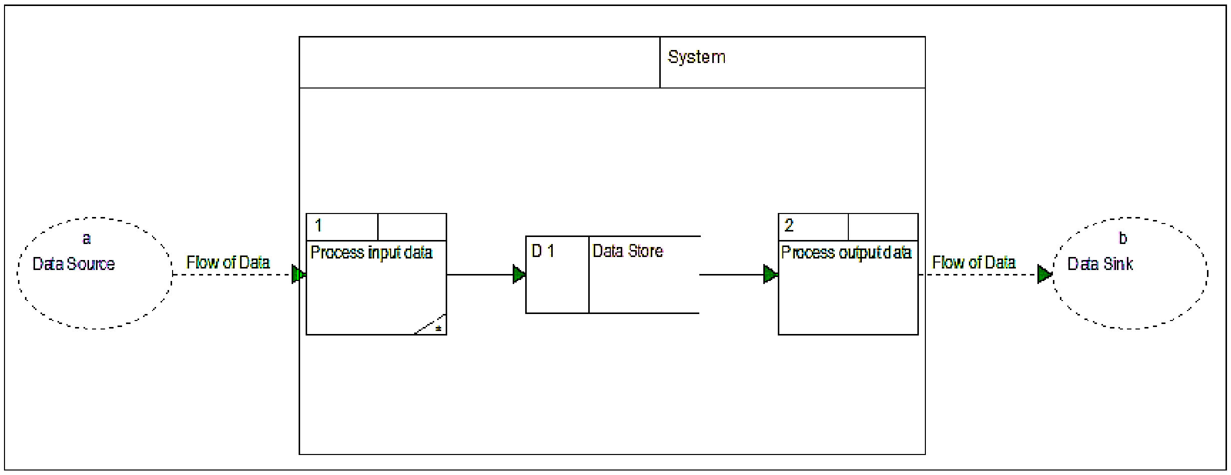
We will look at the legacy formal systems later, but first we look at the essence of the methodology, as it is used in contemporary software engineering. In this context the objective is to produce something helpful and easy to read. There is always a problem balancing completeness and clarity, and where the diagram is the definitive model on which designs are based, completeness has to take precedence. When used as a tool for preliminary analysis, however, clarity is the primary objective.

The concept of a data flow diagram is simple. You show a system or a process as a rectangular box, an external data source or sink as an ellipse, and data flows as directional lines.

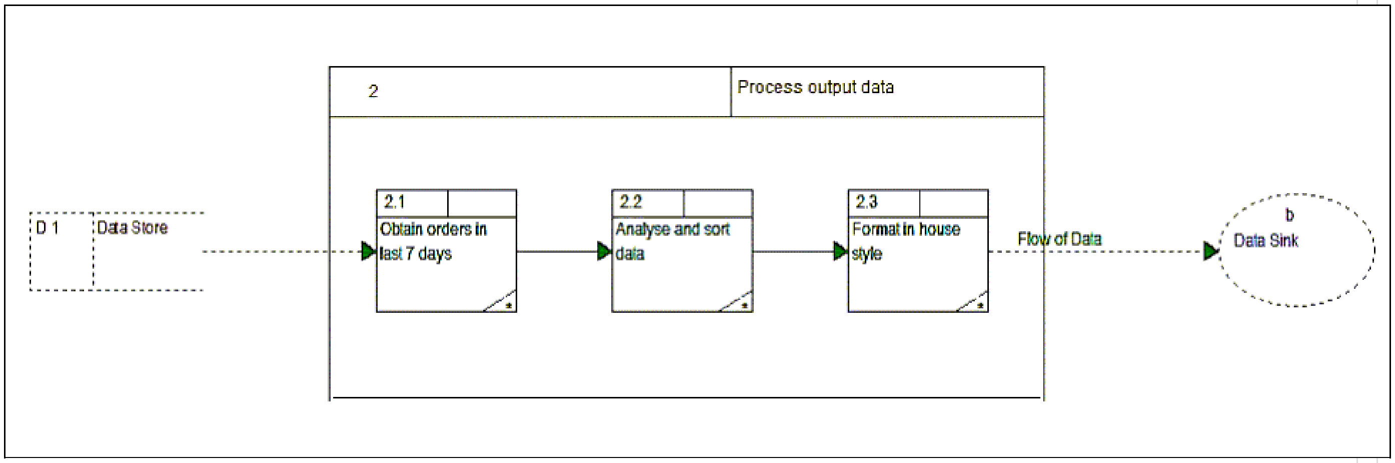


This is all you would have in a top level, or context, diagram (though of course you would have more data sources and sinks, and more data flows.) The idea at this stage is to identify all the data flows into and out of the system, without considering what happens to them within the system. The rectangle contains nothing but the name of the system.

At the next level, here called Level 1, the system rectangle is expanded to show the processes within the system, and any stores of data they use. When using a CASE (Computer Aided Systems Engineering) tool, you would double click (or right click and choose from a menu) to 'explode' the context diagram, and show within it the processes and data stores used.



Finally, each of the process boxes in the level 1 diagram is in turn exploded to produce a detailed summary of the functions that need to be carried out.

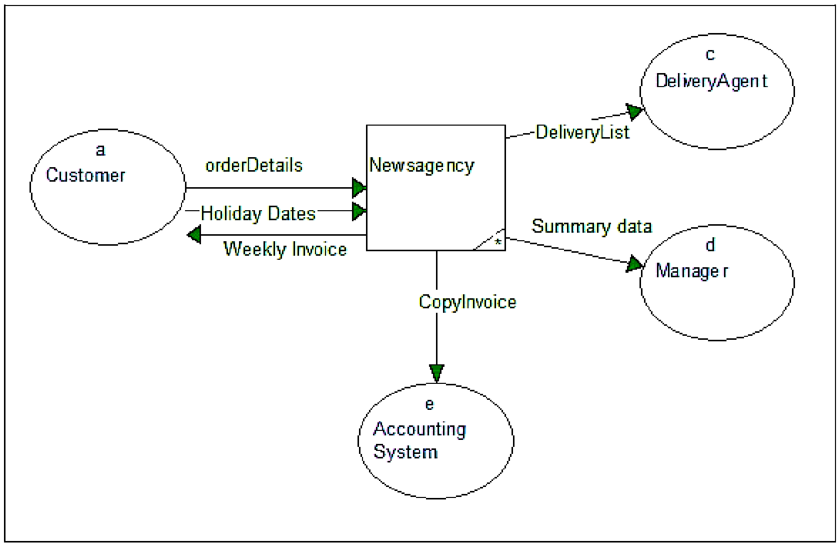


The best way to understand data flow analysis is to look at a simple example, and then produce your own. This is what we will do in the next three sections.

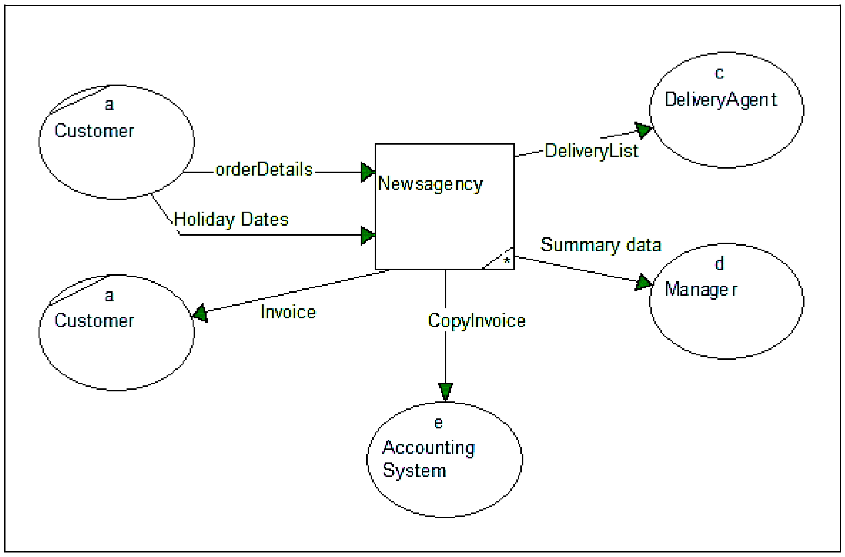
2.2.2 The context diagram

The context diagram identifies the system boundary, the data flows to and from the system, and the external entities that provide or receive the data flows. Here is a context diagram of part of a system of a newspaper home delivery business. (In this illustration we ignore the part of the system that deals with ordering newspapers from the wholesaler.)

There is a more detailed description of the system on which the illustrations in this topic are based in the Newspaper home delivery example [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13621].



This diagram is very simple but there is already some difficulty in fitting in all the data flow lines between the system and customer. In a more detailed system the congestion might make the diagram unreadable. In such cases, we duplicate external entities (using a segment symbol to show duplication), like this:



This is so simple; you might wonder why we do it. It is, however, an important starting point of the analysis of a system. It identifies (or helps you identify):

- the system boundary;
- the external data sources and sinks;
- the data flows to and from the system.

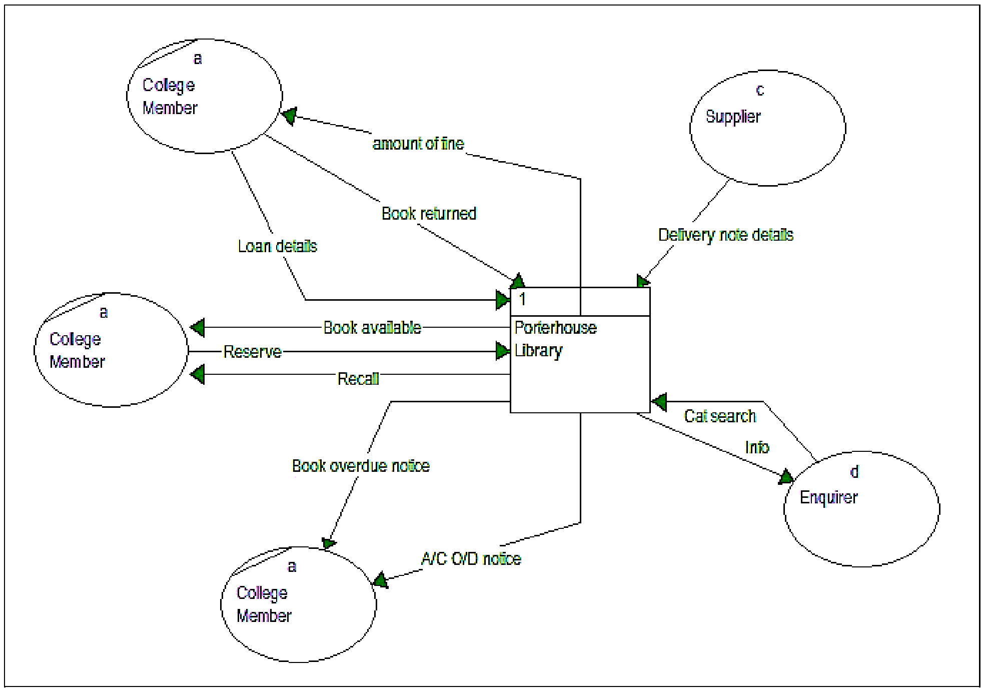
A data flow diagram is concerned with data, rather than with physical objects, and strictly the flows should be labelled in terms of data rather than physical objects. Analysts sometimes go to a lot of trouble to name data flows accordingly. If a customer orders a product, and the product is sent to him, the data flow diagram would show 'details of product sold' rather than 'delivery note' (which is a physical object) and certainly would not show 'Widget' or 'Shoe' or whatever product was delivered. This is fine in theory, but if you are to fit all the data flows in to a single diagram you have to be very concise in your descriptions of them. In the above example we have contracted 'details of amounts owed' to 'invoice' and 'details of deliveries to be made' to 'delivery list'. In some instances, especially where the interaction with the external entity is immediate (a customer in a shop, or a borrower in a library) it is often clearer to refer to the physical entity (since it is reasonable to assume the reader will know you mean the flow is of data about the physical entity). You may need to use such contractions in the following example.

Exercise 1

Draw a context diagram for the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622].

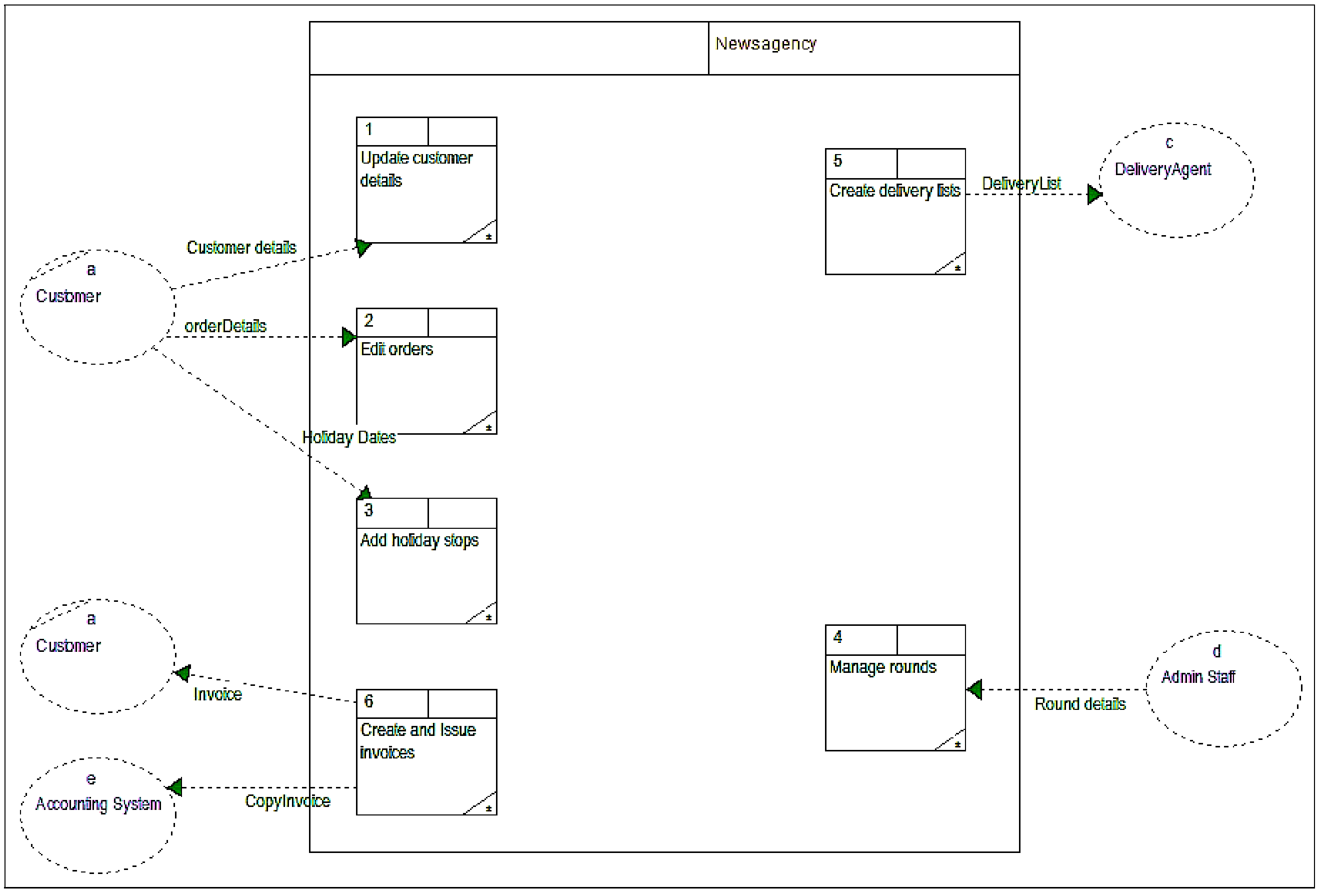
Tutor's comments

Here is my context diagram of the system. You may have produced a different diagram, and will almost certainly have different names for the data flows. Remember, this is a quick sketch of how you see the system. The most important characteristics are accuracy and clarity. If you have represented the data flows clearly and unambiguously, then that is enough. It may be that I have missed things you have seen (or I may have things you have missed). If so, the context diagram has done its job: it has provided an overview we can discuss and eventually agree, of the data flows the system is going to have to process.



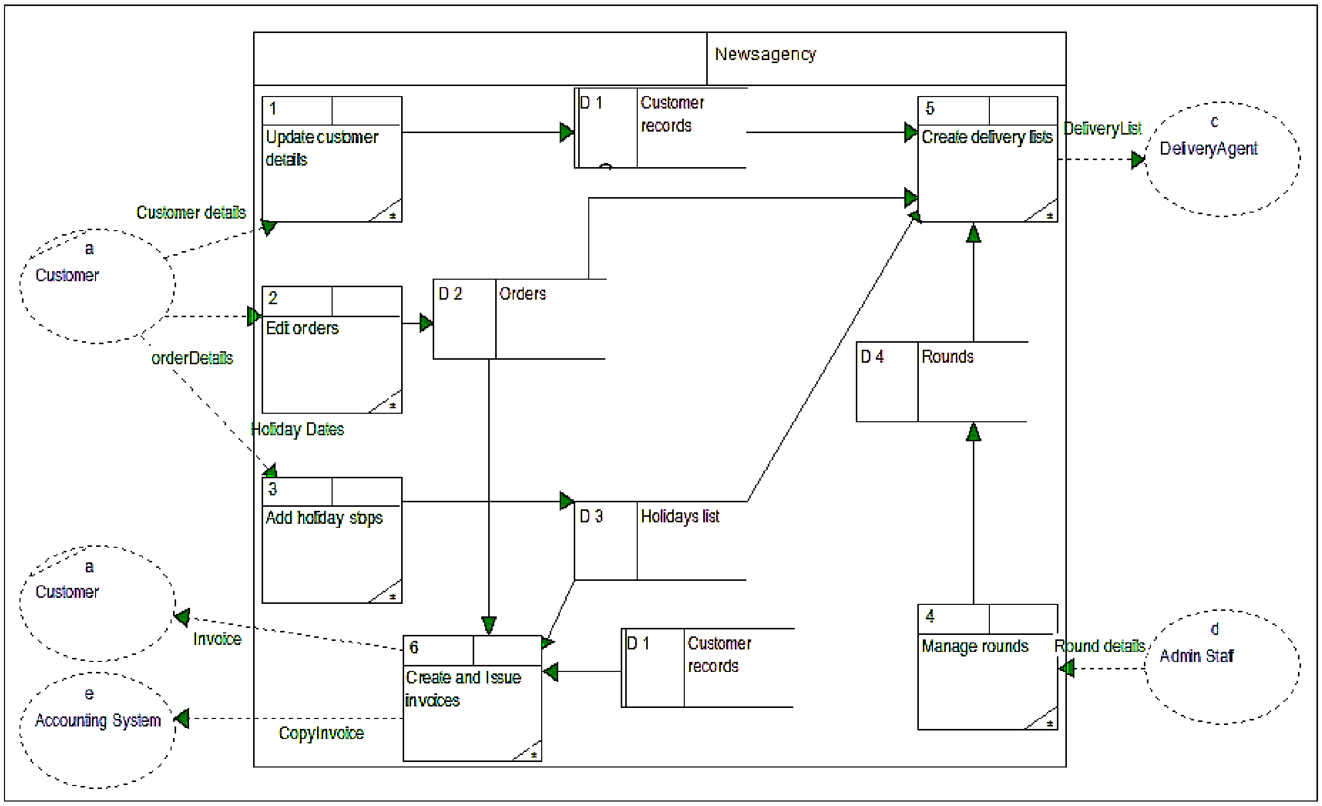
2.2.3 The level 1 diagram

The level 1 diagram shows the same data sources and sinks, and the same data flows, as the context diagram. The difference lies within the system rectangle where the internal processes of the system are identified and associated with the data flows. The first stage is to identify the processes that handle the input data and provide the output data. The processes should be given names that reflect their activity or function.



There is clearly something wrong with this diagram: the data received by the system disappears, and the data leaving the system appears from nowhere. It is an obvious point but one you need to check for in all data flow diagrams. If data arrives at a process, it must also leave it. If a process provides data, it must get the data from somewhere. (You may like to debate this sweeping statement in the Unit 2 forum [link: http://study.conted.ox.ac.uk/mod/forum/view.php?id=13615].) In some systems each process deals with its own input and output data and so there is no problem. More often, data needs to be stored in the system, and this is what happens here.

The data stores shown in a data flow diagram do not represent databases, or tables. They are simply data stores. Separate data stores may be actually parts of the same table, or what is shown in a data flow diagram as a single data store may be a complex joining of several tables. The data store need not be a database at all: it may be a text file, a paper diary or a post it note on the wall. In the data flow diagram we just identify that data is stored. We are not concerned about the medium. So here we add data stores to our level 1 analysis of the newspaper delivery business.

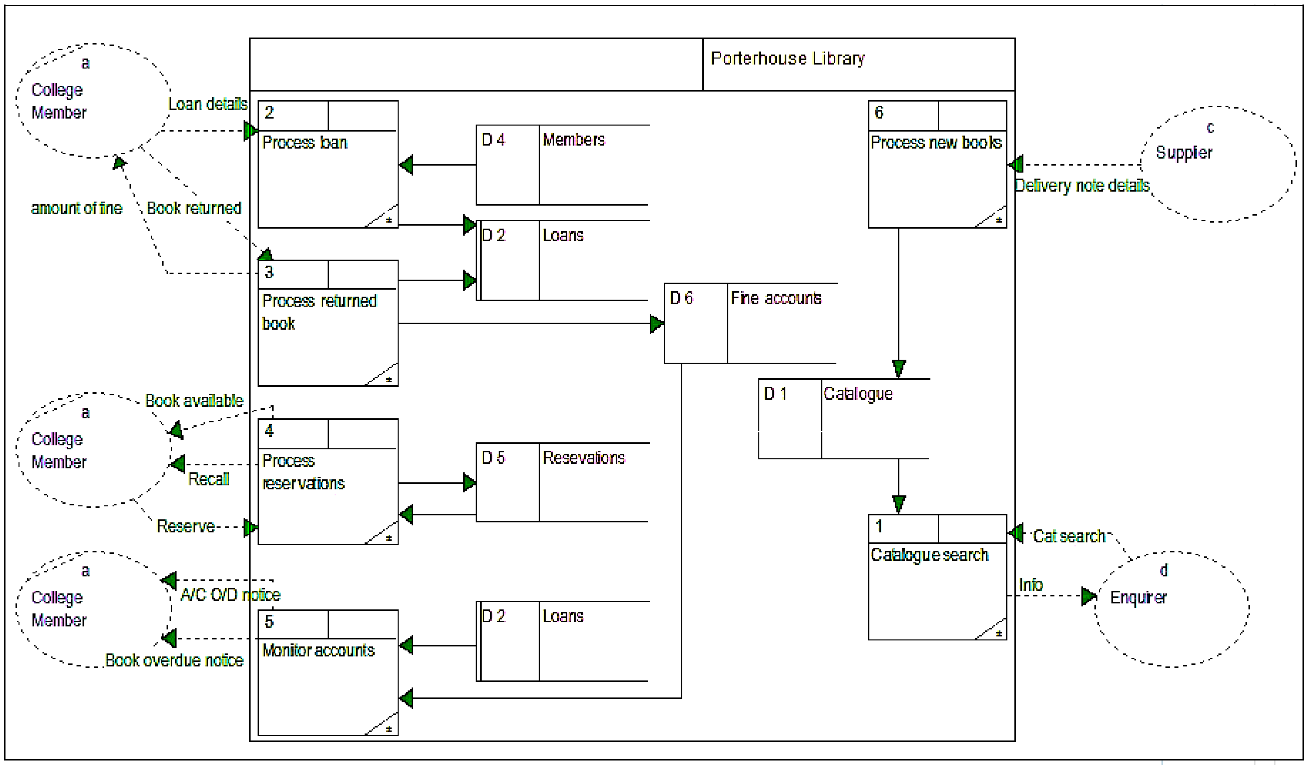


Exercise 2

Draw a level 1 diagram for the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622].

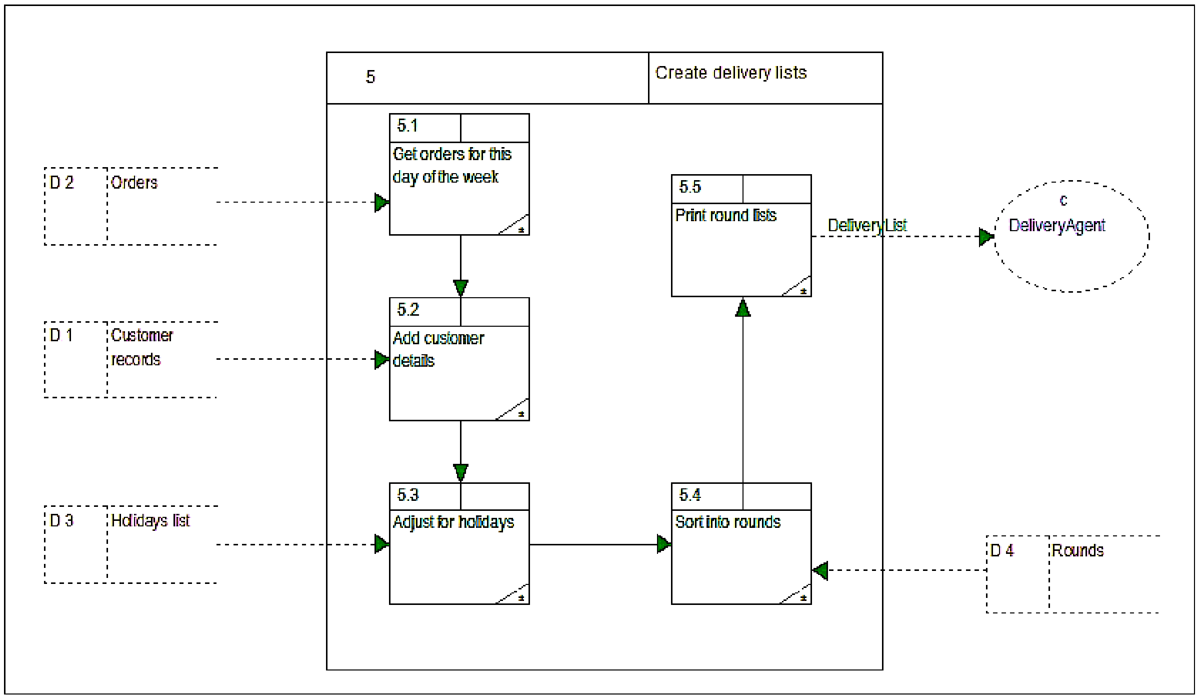
Tutor's comments

Once again do not worry if your diagram does not look like mine. In the interest of clarity and usefulness you may have organised the symbols differently and may have combined some functions. Keep in mind the purposes of the diagram: it helps us think about and talk about the system. If our diagrams differ, that is an excellent starting point.



2.2.4 The level 2 diagram

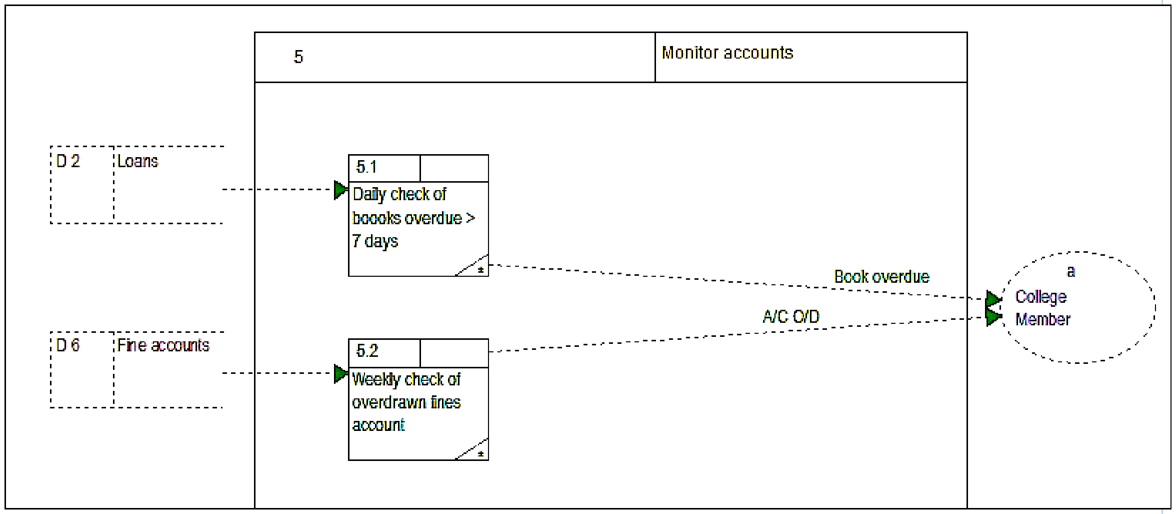
Each of the processes shown in the level 1 diagram is in turn exploded to show a more detailed breakdown of the functions carried out by the process. Here is an example level 2 diagram of the level 1 *Create Delivery Lists* process. Note that the data stores are outside the process box, because that is where they were in the level 1 diagram. We think it helps to include arrows between processes, to show the flow of processing, but they are sometimes omitted, the flow being shown by the sequence of the process numbers.



Exercise 3

Draw a level 2 diagram for one of the processes you identified in your level 1 diagram of the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] (in Exercise 2).

Tutor's comments



2.2.5 SSADM data flow diagrams

So far we have regarded the drawing up of data flow diagrams as a technique to help us understand and discuss the characteristics of a system that we will ultimately model using different techniques. The diagrams are valuable, but they are ephemeral. The design and system documentation will almost certainly be based on another paradigm and use another methodology. Software engineering is, however, not just a matter of producing new systems. It is sometimes necessary to maintain and adapt existing systems, and they are likely to have been designed, and documented, using data flow diagrams. We do not study the full formal system in detail, but here we look briefly at Structured Systems Analysis and Design Methodology (SSADM) and how it differs from the informal approach examined in sections 2.2.2, 2.2.3 and 2.2.4.

When used for system documentation it is important that the conventions of a methodology are followed in detail because the diagrams are not ephemeral. They need to be clear and unambiguous to anyone familiar with the technique. Most of the conventions used in formal data flow diagrams have been covered in the earlier illustrations and exercises, but while previously we have simply seen them as useful or good practices, in a formal diagram they are mandatory. Here are some of the principles we have followed so far based on common sense, but that are mandatory in producing formal documentation:

- Data flows must have a source and a sink. Data cannot just appear, or disappear, within a process. The sources and sinks may be external entities, data stores or (in level 2 diagrams) other sub processes.
- Data cannot flow directly between an external entity and a data store. This is not just a matter of security. Accessing data is intrinsically a process (whether querying a database, displaying a web page or even looking at a diary). For the same reason data cannot flow directly between two data stores.
- Data flows between processes within a system should only be shown if they are immediately sequential. Many authorities suggest it should never be necessary in a level 1 diagram where data flows are usually with external entities or with data stores.
- Data stores should not act as pure sources or sinks of data. (You can easily understand why this would not normally happen, but you may like to think about, and discuss in the Unit 2 forum [link: http://study.conted.ox.ac.uk/mod/forum/view.php?id=13615], whether this rule should be absolute.)

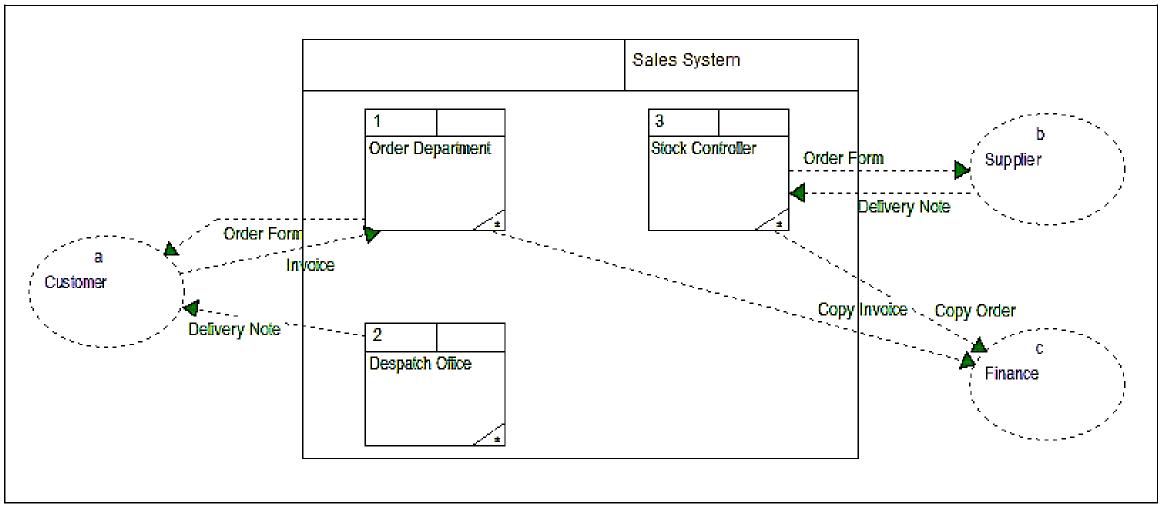
Another rule (one we have already broken) is that data flow lines should never cross. People sometimes go to great lengths to avoid crossing lines. It is better to avoid them, of course, but sometimes legacy diagrams are less clear than they might have been had a few lines been allowed to cross.

The sequence of diagrams in a formal system uses the three levels we have studied (context, level 1 and level 2) but it distinguishes between physical and logical models, and produces different models for the existing and the required system.

The sequence of diagrams produced could be:

1. A document flow diagram of the existing system.
2. A context diagram of the existing system.
3. A physical level 1 diagram of the existing system.
4. A logical level 1 diagram of the existing system.
5. Level 2 diagrams of the existing system.
6. Diagrams 4 and 5 redrawn to include new functions in the proposed system.

The document flow model is one we have not so far considered. This identifies the documents flowing to and from, and around, a system. It provides the basis for the construction of a physical context diagram. The document flow diagram is most useful in a large system where there are document flows between departments of an organisation. Here is an example of a simple document flow diagram:



You may wonder why people bother with this kind of diagram, but in a large system it can provide insight that helps the analyst in drawing up the data flow diagrams.

The formal context and data flow diagrams are similar to those we studied in sections 2.2.2, 2.2.3 and 2.2.4. The physical diagrams show the format of the data flows (as in the document model). The logical models are abstractions that show only the data, and not the form in which it is transmitted (for example *delivery details* rather than *delivery note*).

Over the years analysts have tended to ignore the distinction between physical and logical models (as we have done in our earlier diagrams) but a formal system will always have at least two sets of drawings: one of the existing system and one for the proposed system.

2.2.6 Further reading

Although you can study this topic and cope well with the assignment using only the unit materials, we recommend that you do some background reading on Structured Systems Analysis and Design Methodology (SSADM). You may be able to find books on the subject in a library, or you may like to purchase one of the many books which, though expensive at full price, are widely available second hand for very little. Here is an example of an excellent book you should be able to pick up for a couple of pounds:

Weaver, P. L., Lambrou, N. and Walkley, M., 1998 (2nd edn)
Practical SSADM Version 4+ – A Complete Tutorial Guide
(London: Financial Times Management)
ISBN 0273626752.

2.3 Static Data Modelling

2.3.1 Introduction to data modelling

When we make process models we model things that happen to data, so we include in our models such things as:

- data flows to and from the system;
- manipulation of data items within the system;
- storage and retrieval of data items.

In these process models, however, we do not concern ourselves with the way in which the persistent data is stored: we simply assume we can retrieve any stored data item. We similarly model the addition, updating and deletion of data items without thinking too deeply about the way the persistent data is stored. This is why in data flow diagrams we refer to data *stores* rather than tables or databases. Such models focus on the way the data is processed, rather than how it is stored.

In this section we look at data from another point of view. We seek to model the data in terms of its logical structure, so that we can design storage systems that ensure persistent data retains its integrity (including links between data items) and is easily accessible.

Later (in Unit 3) we ask you to reflect on the meaning of data and information but for the moment do not get too philosophical about it. We have already talked about data in terms of its flows into and out of a system, and of the processes that manipulate and channel it, and we are here talking about the same thing. It may not be easy to define data but we can all recognise it when we see it.

The number of types of data items in a system can be very large, and they clearly need to be organised. We would instinctively group together closely related items (for example the name, address etc. of an individual) and this is what we do when we carry out an analysis of the static data in a system. We seek to describe the data in terms of **entities** and **attributes**. Again, we will look at these terms more closely in Unit 3, but for the moment we stick with the way we would group data instinctively. Each data item is an **attribute** that is part of a group of items describing, or relating to, an **entity**. In the early stages of analysis we make informed guesses about these entities and attributes. We might find later that our first impressions were wrong, and will need to revisit our informed guesses: that is all part of the iterative process of analysis.

We will later look at a more structured way of identifying entities and attributes, but in essence it is quite straightforward. Some data items are single values which need no further data to complete them, for example a *date of birth*, *a course name* or the *length of a race*. These are simple attributes. Other data items cannot stand alone and need some attributes to describe or extend them, for example *student*, *course* or *race*. These are entities.

The concepts are not mutually exclusive. Something can be at the same time an attribute and an entity. For example, it may be clear from your investigation that Tutor is an attribute of Module (each module has its own tutor), but clearly Tutor is also an entity that will have attributes of its own (Name, ExtensionNumber etc).

The identification of entities and attributes relies on understanding the nature and meaning of the particular data in the system under review. Consider data about a BMX bicycle. It would have attributes like frame colour and frame size, but we might also need to store data about the pegs these machines often have fixed to their back wheels. Is the Pegs data item an Entity?

Tutor's comments

The fact is we do not have enough information to decide. If we were producing a system for a holiday camp hire organisation, the data we would need to record about Pegs could be simply Yes or No, in which case Pegs would be just an attribute of BMXBike. But if the system were for the hoped for Olympic event in this sport it is likely that team managers and judges would need to know the weight and size of the pegs and the material they are made from. In this scenario Pegs would become an entity in its own right, as well as being an attribute of BMXBike.

Data analysis is as much an art as a science but in essence you can test your decisions by asking simple questions.

- Does this entity have any attributes? If not it probably is not an entity, but just an attribute of some other entity.
- Does this attribute have attributes of its own? If so it is probably an entity as well as an attribute.

Individual activity

Consider whether the following data items are entities or attributes. For those that are attributes, suggest an entity they may belong to. For those that are entities suggest two or three attributes of the entity.

1. Price
2. Film or Video
3. Department
4. Colour
5. ISBN (book number)

Tutor's comments

1. A price is likely to be an attribute of an entity like Product, or Performance.
2. A Film or Video is a entity that will have attributes like, Title, Certificate, Directed By.
3. There are different possibilities here. It could be that the only data about a department is its name, in which case it is simply an attribute, probably of an entity like Employee.

   If, however, the system contains other data about departments, for example the department head, and contact number, it would also be an entity (with DepartmentHead and ContactNo as attributes of it).
4. A colour is usually just an attribute. In the context of computer science, however, you might need to record not only the name of the colour, but the number of bits in which it was stored and its RGB values. In that case it would also be an entity.
5. An ISBN is an attribute (along with attributes like Author and Title) of a Book entity.

In this exercise allocating attributes to entities was fairly straightforward, but in complex systems it is not so easy. There may be more than one candidate entity for an attribute. If we are not careful we might include the same attribute more than once. Data analysis and modelling procedures are designed to ensure attributes are in the right place and that there is no unnecessary duplication of data.

There are two broad approaches to data modelling, sometimes described as top-down and bottom-up analysis.

The top-down approach concentrates on the entities and how they relate to each other. Diagrams are produced showing entities as rectangles with relationships shown as lines between the rectangles. The end product of this approach is a list of entities, to which we then assign the relevant attributes.

The bottom-up approach starts with the attributes and, using some formal analysis rules, divides them into the relevant entities.

Both are used in systems analysis and they should produce the same result. Obtaining the same model with different techniques is reassuring and minimises the risk of error. In this topic we will look at both approaches to data analysis, starting with entity relationship modelling.

2.3.2 Entity relationship modelling – concepts

An entity relationship model shows, in diagrams and textual descriptions, the entities in the system and the nature of the relationships between them. In the formal documentation of a system there are two types of entity relationship model. The first is a model of the real world data, representing the data entities in the observed system and the apparent relationships between them. The second is an extended model showing the data structures needed to implement a relational database. In terms of the final documentation (and the approach of most text books) these are separate stages of the analysis which should be done sequentially. In this topic, however, we are focussing on entity relationship modelling as an aid to analysis and understanding, rather than as a way of producing formal statements, and we will make use of both kinds of model throughout the topic.

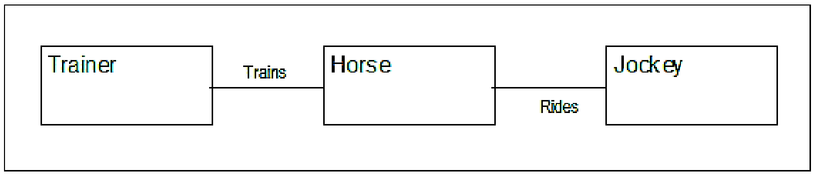
In this section we explore some of the concepts of entity relationship modelling. The diagrams we draw here will represent small parts of a system, so that we can concentrate on the underlying concepts. We will explore the drawing of diagrams for full systems later in the topic.

The essence of an entity relationship model is that entities are represented by rectangles and relationships between them as lines. The requirements or scenario is studied and decisions made about the likely entities or groups of data in the system. We look later at ways of undertaking this analysis, but in this section we use simple examples where the entities are quite obvious. Remember the simplistic but useful generalisation: an entity is something that has attributes.

Example

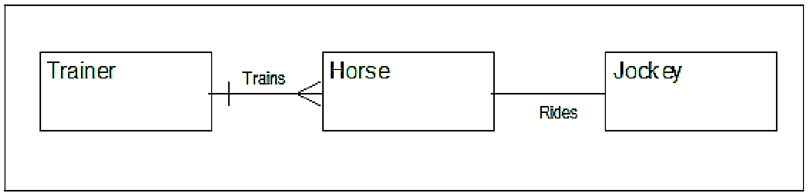
The data to be modelled relates to a riding stable. The stable needs to keep records of horses, including who trains the horse. They also need a record of which jockey rides which horses.

We draw labelled rectangles for what seem to us to be entities and join the rectangles with lines to show the relationships between the entities.



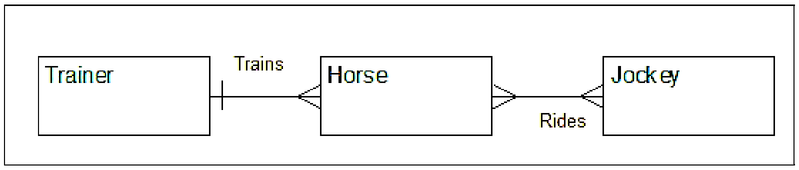
It is a very simple diagram and may appear to overlook some nuances of the system. That does not matter, indeed it is an advantage. A first cut identification of entities and relationships should stick to the obvious. Our analysis of the obvious entities provides insights that will help in identifying and modelling less obvious entities.

We then add symbols to show the cardinality (or some call it the multiplicity) of the relationships. Imagine you are standing in one of the boxes, look along the line and ask how many instances of the entity you see at the other end of the line you might be associated with. Standing in the Trainer box you would conclude a trainer might train several horses, so you would add the 'many' symbol to the Horse end of the relationship line.



Then stand in the Horse box and ask how many trainers might a horse be associated with in this stable. Assume the requirements indicate that a horse only has one trainer at a time and that there is no need for historical records. The cardinality of the Trains relationship as seen by the Horse (that is at the Trainer end) is 1, so there is no need for an extra symbol.

Turning around to face the Jockey, a horse will be ridden by several jockeys and here we are told we do want historical details, so the Rides relationship as seen by Horse (that is at the Jockey end) is 'many' so we need to add the 'many' symbol. Standing in the Jockey entity, we follow the same reasoning to deduce that the Rides relationship also has 'many' cardinality when viewed from Jockey (that is at the Horse end of the line). So we end up with an enhanced drawing which shows not only the entities and relationships, but information about the ways in which they can be related.



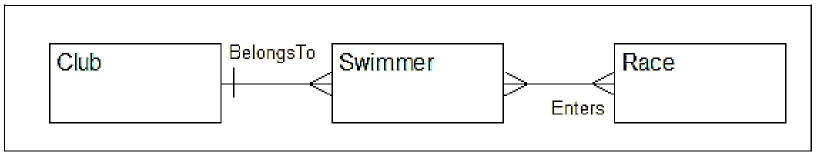
This approach may seem very simple, but it is at the heart of entity relationship modelling and before moving on check you have fully understood the representation of cardinality by sketching (with pen and paper) outline entity relationship diagrams for the following example scenarios. (These examples are over-simplified and we know they do not represent real world data sets).

Exercise 1

The organisers of a swimming gala want a database to store details of the races that take place, the swimmers who enter the races, and the swimming clubs to which the swimmers belong. A swimmer can only be registered as belonging to one club, but can enter several of the races. Given the information so far, sketch the probable entities and relationships.

Tutor's comments

My first draft on the information provided was:



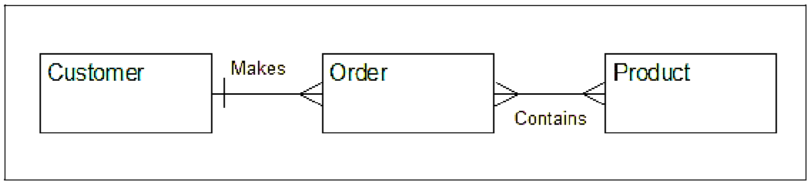
If you decided there must be another entity, one not mentioned in the scenario, do not worry. You are probably just ahead of us. Some analysts see instinctively the need for additional entities and introduce them from the start. For the moment, however, we recommend you stick to the obvious entities and relationships and refine the model later.

Exercise 2

A simple orders system needs to keep details of the orders made by customers, and the products they order. There are no joint customers but orders can be for more than one product. Identify the entities and relationships you would need in your initial model. (Again, for the moment, do not look below the surface of the scenario: stick to the obvious.)

Tutor's comments

My first draft was:



The cardinality at the Order end of Contains is sometimes overlooked. As you stand in the Products box and look out, the question is, as a particular product (for example as product number T127): "How many times might I be represented in an instance of Order?" Looked at from that point of view, the answer is clearly, many times.

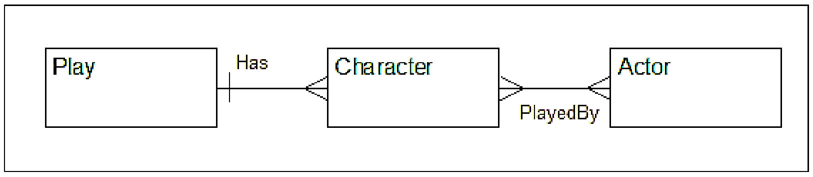
Again, if you have identified other entities and relationships, you may well be right, but for the moment we ask you to go along with the restricted initial models we have identified.

Exercise 3

A theatre company wants to keep a record of the plays in its repertoire, and for each play a list of the characters; (in this example assume a character can appear in only one play). It also wants a record of the actors, showing the characters they have played.

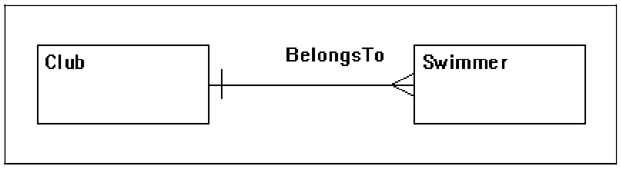
Tutor's comments

This is getting repetitive, I know, but the fact is that most data models are composed of subunits that look like these examples.

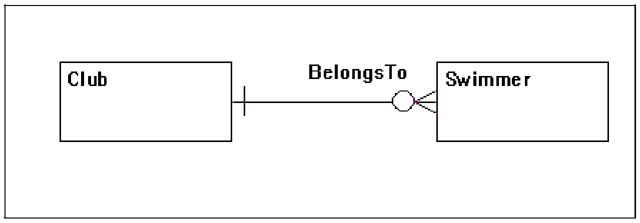


Participation (or existence) symbols

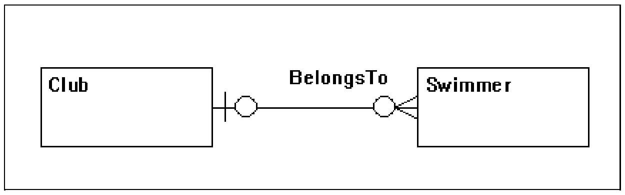
Finally, we add symbols to show the participation (or existence) of each relationship. The way participation is represented in entity relationship diagrams varies. We will look at an alternative way of showing participation later in the unit but here we use an approach that is common to a number of methodologies. It uses a circle symbol to indicate that a relationship with the entity is not mandatory (or is optional). Again, the easiest way to understand the use of this symbol is through examples. In the Gala example it is clear that a Swimmer must belong to a Club, so the Club entity's participation in the relationship is mandatory. Accordingly, seen from the point of view of the Swimmer (stand in the Swimmer box and look along the line) the relationship is mandatory, so there is no option symbol at the Club end of the line.



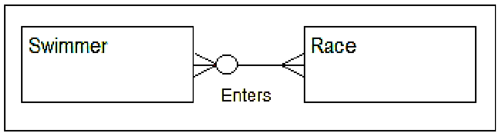
Assume we had established that the system might need to record details of clubs that had, for the moment, no members. Standing in the Club box, it is possible we might see no instances of Swimmer entity at the other end of the line, so the relationship, from the point of view of Club is not mandatory (is optional) so that point of view is marked with a circle.



Imagine that we now find that whilst most swimmers do belong to clubs the organisers sometimes allow unaffiliated swimmers to enter races. Now, standing in the Swimmer box, we can envisage an instance of Swimmer that would not see a club instance at the other end of the BelongsTo line. This optional participation needs to be shown by adding a circle to the Club end of the relationship line.



Consider the other part of the Gala diagram. What are the participation conditions of the Enters relationship? It is, as ever, something you would need to establish through your investigation, but here we can assume the system could hold details of races before any swimmers have entered for them. So, standing in the Race box and looking along the relationship line, we should see an optional signal at the Swimmer end.



Do we need an option symbol at the Race end of the line? It depends on the facts, but here we assume the system would only hold details of swimmers who had entered for at least one race. Looking down the relationship line from Swimmer, therefore, we would see mandatory participation in Race.

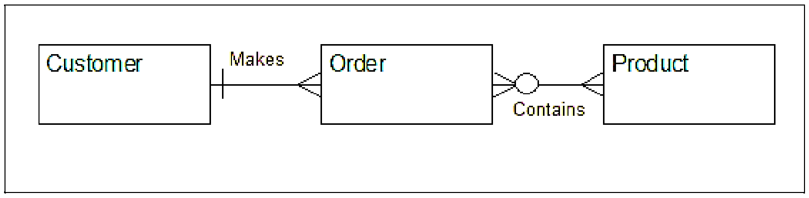
Exercise 4

Return to the exercises you worked on earlier (Customer Orders and Theatre Company) and add to your sketches the participation symbols in the relationships.

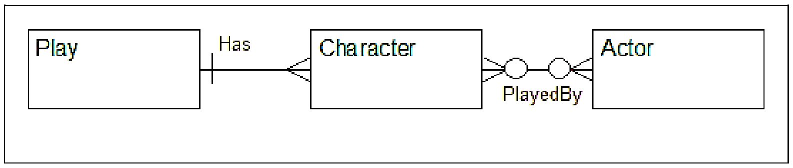
Tutor's comments

Here are my revised diagrams. They reflect my assumptions about the real-life relationships they represent. Yours may be different. The important thing is that you consider the participation conditions of each relationship. Here you have to make assumptions, but in a live analysis you would need to go back to the client to seek clarification.

In the orders example I assumed we would not keep records of customers until they made their first order. Clearly an order will always have a customer. I took the view that an order will always contain at least one product, but that a product (perhaps a new one) might not appear in any orders.

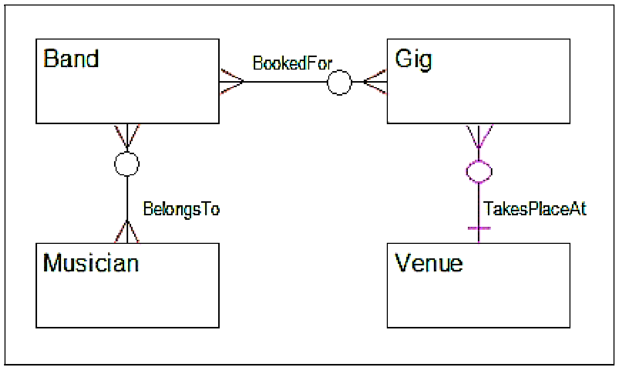


In the theatre example I assumed that the company might enter details of plays and their characters before they were cast, so the PlayedBy relationship, as seen from the Character entity should be optional. I was not sure about the other way round, but decided the system might hold details of actors who had not yet had a part in a play.



Real systems, of course, have many more entities and relationships, but they are made up of sub-units like the ones we have worked through so far. Read the following outline of a slightly more complex system and look at my initial entity relationship diagram of the system. Notice how it is composed of one-to-many (1:n) and many-to-many (n:m) relationships.

An entertainment group runs musical performances in a variety of venues. It calls the events 'gigs' and each gig has a date, time and venue. A record is kept of the name, phone number and seating capacity of each venue. Bands are schedule to play at gigs. For each band in the system a record is needed of the name of the band, the genre of music played and a contact phone number. A separate record is kept of the names and phone numbers of musicians that play in the bands and the instrument(s) they play



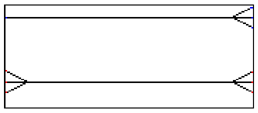
If you are already familiar with database design, or you have thought through the implications of the scenario you may think this diagram does not fully model the data. Remember that at the moment we are using the methodology to help us think about the data structures. It helps to keep things as simple as possible in the first conceptual diagram, and that does sometimes mean postponing decisions about the precise location of some of the attributes.

In my interpretation of the scenario a musician might be temporarily not part of a band but still retained on the books, so the participation of the BelongsTo relationship from the point of view of Musician is optional. On the other hand a band with no members would be removed, so the BelongsTo relationship, as seen from Band, is mandatory. Similarly, a particular band might not yet have been booked for a gig, but a gig would only exist if at least one band was booked for it. The participation of the TakesPlaceAt relationship from the point of view of Gig is mandatory (a gig must take place somewhere) but the other way round participation is optional (a venue may be on the books but not yet used for a gig.)

This is a typical first cut diagram of the obvious entities in the system and the relationships between them. It will almost always involve some many-to-many relationships like those above. In the next section we will discuss the problems caused by many-to-many relationships, and a standard technique to enable us to overcome those problems.

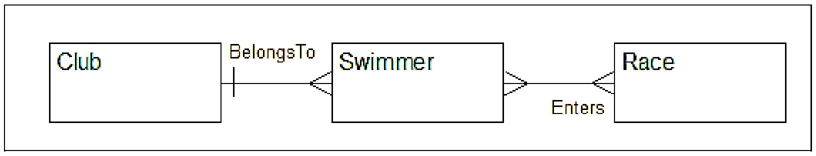
2.3.3 Entity relationship modelling – further concepts

The diagrams we have drawn so far involve two kinds of relationships: one-tomany (1:n) and many-to-many (n:m).



These are the most common relationships in real world models, and in this section we examine the way in which they can be implemented in a fully relational data model. For the moment we are omitting participation symbols from relationship lines, in order to concentrate on the cardinalities. (The concepts and techniques covered in this section apply equally whether or not the participation symbols are shown.)

Here is the Swimming Gala example we considered in section 2:



These are representations of the real world relationships between entities. A club will have many swimmers, but a swimmer can belong to only one club. A swimmer may enter many races but equally a race will have many entrants.

At some stage we will need to consider how these real world relationships will be represented so that they can be modelled in a relational database. In some textbooks this stage of the process is only considered when the real world model of the data has been completed. It is, however, worth an initial look at these issues now, because they provide insight into the process of identifying the entities and attributes in the initial model.

We will look in more detail about relational theory in Unit 3, but we consider here one of the basic rules, which is that an attribute value must be 'single-valued' (or atomic). Imagine a Swimmer table with columns representing the entity's attributes, for example, Name, DateOfBirth, Phone, etc. Each row in the table would consist of cells containing relevant values. Accepting for the moment the Name column (which is a special case we look at in Unit 3) the cells in each row will contain a single value. They will be atomic.

Swimmer table

Name	DateOfBirth	Phone
Smith	02/06/1984	012349
Jones	05/12/1979	020345

Now consider what would happen if we tried to show the races each swimmer had entered.

Name	DateOfBirth	Phone	Entered
Smith	02/06/1984	012349	Race 1 Race 3 Race 4
Jones	05/12/1979	020345	Race 3 Race 4

We end up with multiple attribute values in the Entered column, something we cannot have if we are going to represent the data in a relational database.

Where you have a one-to-many relationship multiple attribute values can be avoided quite easily. It is just a matter of choosing the right way to link tables. If you try to represent the BelongsTo relationship using an attribute of the Club table you will get multiple values.

Club table

Name	Phone	Members
Town ASC	012349	John Mary Bill
City Dolphins	020345	Joe Susan

But representing the link using an attribute of the Swimmer table will produce only single values

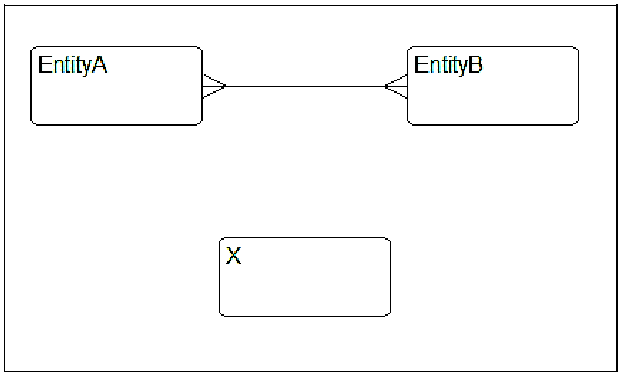
Swimmer table

Name	DateOfBirth	Phone	Club
Smith	02/06/1984	012349	City Dolphins
Jones	05/12/1979	020345	Town ASC

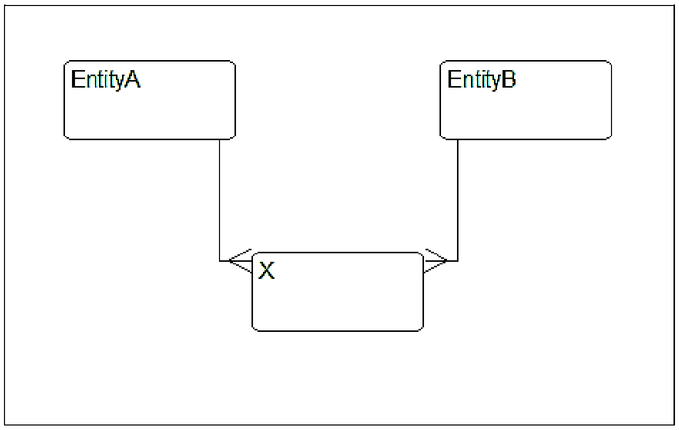
Where you have a many-to-many relationship you cannot avoid multiple values. Either you have multiple values of swimmers in the Race table, or you have multiple values of races in the Swimmer table.

You will, therefore, eventually have to eliminate many-to-many relationships from your final model. As you become more familiar with the drawing up of entity relationship models, and with relational theory, you will gain more insight into the way we do this. For the moment, however, there is a very simple and reliable way to eliminate many-to-many relationships.

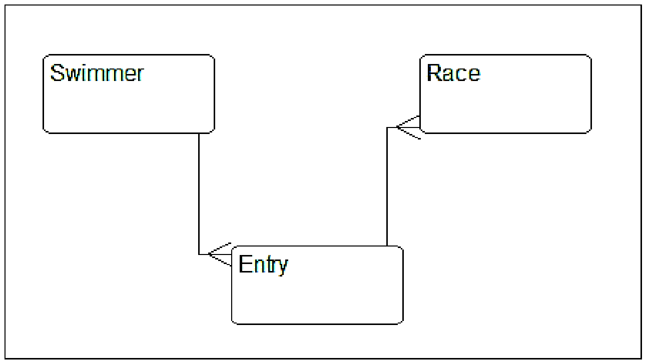
Given a generic many-to-many relationship, you create a new entity and (for the moment) give it a meaningless name (we simply use the name X).



We then replace the many-to-many relationship with two one-to-many relationships. In each of the new relationships it is the new entity that is at the many end.

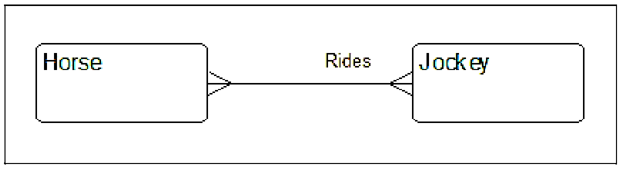


This is a generic operation you need to carry out whenever you need to replace a many-to-many relationship and it is always worth doing it in this generic way, using a meaningless name for the new entity. If you give the new entity a meaningful name from the start, there is a possibility you may be led by the name into making an error about the cardinality of the relationship.

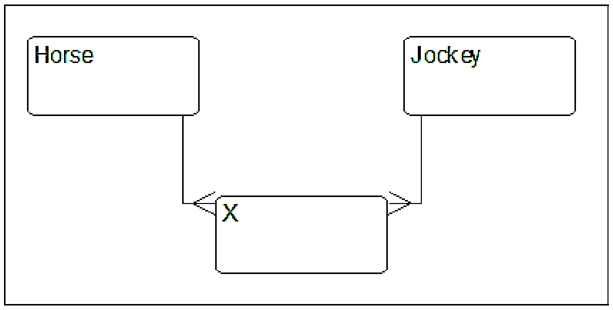


There is clearly an error here, but it is the kind error that is often made when the analyst talks through the cardinalities saying something like “There will be many races entered”. It is better to enter the cardinalities before naming the new entity. They will always be the same: both relationships will be ‘many’ at the new entity. Once the new relationships are drawn, a suitable name can be chosen for the new entity.

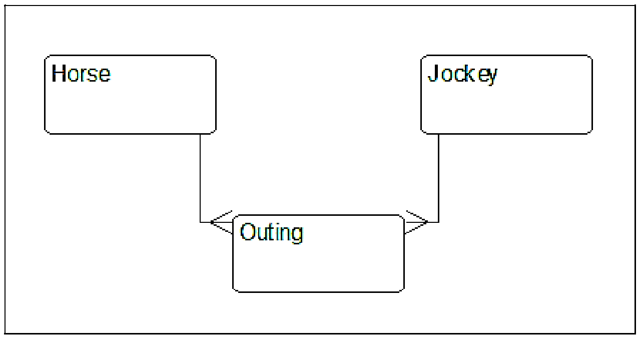
We earlier looked at a diagram representing part of the data about a riding stable which included the relationship:



The first step in eliminating the n:m relationship is to introduce a new unnamed entity and its generic relationship lines.



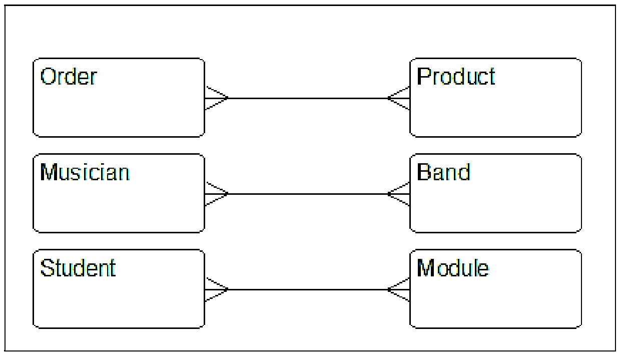
The process is completed by giving the new entity a suitable name. In many cases the name of the new entity will make it clear what the new relationship lines represent and it may not be necessary to label the relationships. Here I have named the new entity Outing, though you might equally use the word Ride. You would not, however, use the word Race because that would represent only a subset of the relationships.



The new entities produced in order to remove many-to-many relationships are called **Intersection Entities**. They are very common in data models and their table equivalents are present in most databases.

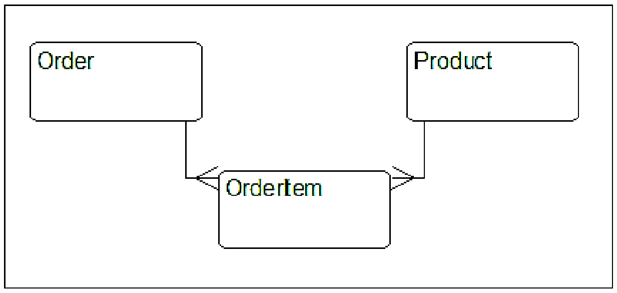
Exercise 5

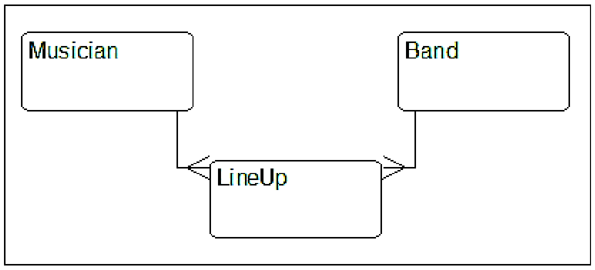
Make sure you fully understand the process of creating intersection entities by sketching the new entity and relationships for each of the following:

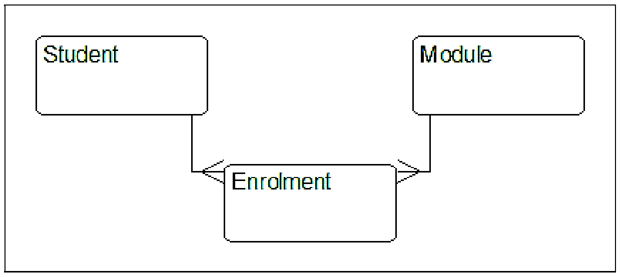


Tutor's comments

Here are the new entities we decided to use:



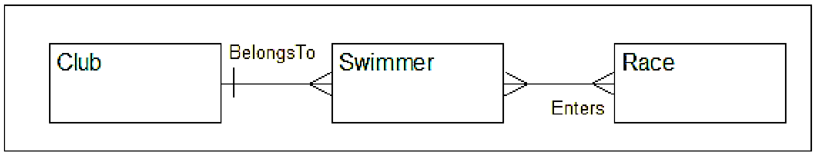




We mentioned earlier the value of knowing about the broad requirements of the full relational model even when undertaking an initial analysis of the data. Here is an example of the insight knowledge of intersection entities can provide, when analysing data. Consider a slightly more involved description of the Swimming Gala scenario.

The organisers of a swimming gala want a database to store details of the races that take place, the swimmers who enter the races, and the swimming clubs to which the swimmers belong. A swimmer can only be registered as belonging to one club, but can enter several of the races. They need to keep records of the time taken and position achieved by swimmers in each race they enter. Given the information so far, sketch the probable entities and relationships.

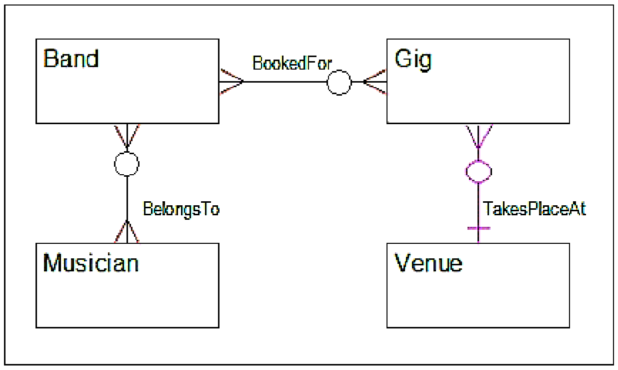
The initial conceptual model of the real world entities and relationships is the same as in the original example:



Given the additional information, however, you might be uncertain how to deal with the 'time taken' and 'position achieved' requirements. Clearly they cannot be straightforward attributes of Swimmer or Race: they involve both entities. Knowing that the *Enters* relationship will require a new intersection entity reassures you that you will eventually have a home for the time and position attributes, and you can safely ignore them in the initial real world entity relationship model.

Exercise 6

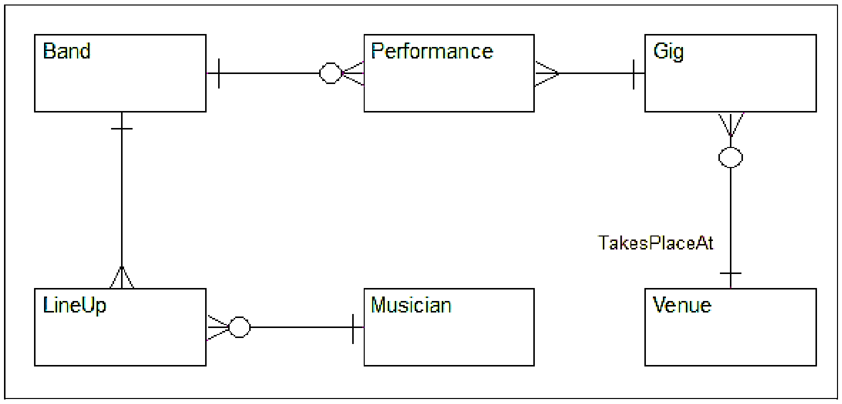
Ensure you have understood the nature and purpose of the *intersection entity* by working through this exercise. Here is a scenario we considered earlier, and the real world model we constructed to represent the system's static data structures.



Use the techniques discussed in this section to convert the above model into a fully relational logical model.

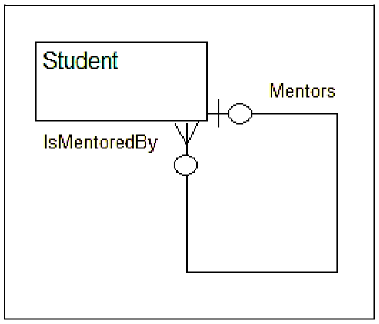
Tutor's comments

The *BelongsTo* and *BookedFor* relationships are many-to-many so we have to replace them with intersection entities. I have called them Performance and LineUp, but there are other suitable names you could use. What is important, however, is the cardinality of the relationships with the new entities.

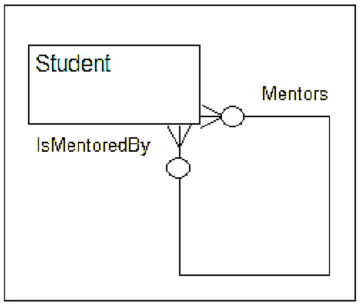


Before we move on to do a full entity relationship model there are two more types of relationship you need to know about. The first is a one-to-one relationship. It has a theoretical place, for example in most cultures, the relationship MarriedTo between entities Man and Woman, would be a one-to-one relationship. In practice one-to-one relationships tend to be used where entities are generalised, for example where there are relationships between a CollegePeople entity and both a Student entity and an AcademicStaff entity. We consider these situations in Unit 3 but you will not need to use 1:1 relationships in this unit.

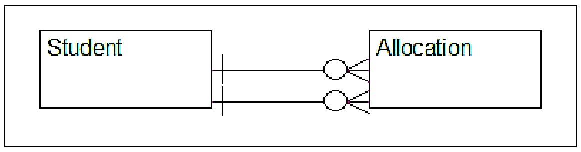
The other new relationship is the reflexive relationship. This occurs where an entity is related to itself. Consider a college where some students volunteer to be mentors for new students, and the system needs to be able to identify a student's mentor. Assuming that a student may be mentor to more than one student, but that a student will only have one mentor, the entity relationship diagram for this part of the college system would be:



Imagine the (improbable) decision is made that a student can have more than one mentor. The new model would be:

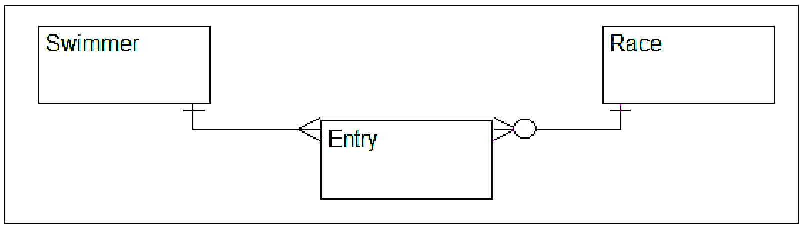


This is a many-to-many relationship and in the final model it will have to be replaced. The procedure is exactly the same as when the relationship is between two entities. Create a new entity. Replace the many-to-many relationship with two one-to-many relationships, then give the entity a suitable name.



Note that, as seen from Student, participation in Allocation is optional, since a student need not have, or be, a mentor. Looked at from the point of view of Allocation, however, participation of Student is mandatory. This reflects the fact that any instance of Allocation must involve two Student instances.

It is usually the case that, looking from the point of view of an intersection entity, the relationships with the original entities will be mandatory. This is because instances of the intersection entity will only come into existence when there is a need to join instances in the original two entities. It is illustrated here in the Swimming Gala diagram from above, but with participation symbols added.



A Race instance can exist without needing to link to an Entry instance, but an Entry instance must, by definition, be linked to instances of Race and Swimmer.

2.3.4 Entity relationship modelling – methods

In a complex set of requirements the identification of attributes, entities and relationships requires lateral thought and the exercise of judgement. A useful starting point, however, is a textual analysis of the system's description and functions. As a generalisation, nouns usually tell us about entities and attributes, whilst verbs usually tell us about relationships.

Note this is a technique to help in analysis, and like all such techniques it is used iteratively. A first analysis is likely to bring to light further questions we need to ask the client about the nature and extent of data kept about the entities identified.

Read the following example description of the data in a system. Note that we have kept it short so that we can concentrate on the technique, rather than the data. Following a full investigation the textual description would be much longer and would contain more information about data attributes. At this stage, however, we are mostly concerned with the identification of **entities** and **relationships**:

> You are asked to design a system for an artists' cooperative. Artists submit their paintings (catgorised as Water Colour, Oils or Mixed Media) to the cooperative and from time to time exhibitions are arranged. Each exhibition has a name and relates to a single art category and occurs at a named venue (for which we need to have a contact number and the size of the permanent display area). An exhibition may be sponsored by one or more local companies that are approached from time to time by email. Each exhibition is coordinated by one of the members of the cooperative. There is no guarantee that a submitted piece of work will appear in an exhibition, but most do, and many appear in several locations.

The organisation has a long tradition of mentoring new artists. It is not compulsory but all new members are encouraged to have a mentor, and all members can choose to have one. There is no special qualification for being a mentor and it is left to the good sense of the people involved. The collective do, however, want a record kept of these unofficial mentoring links.

The first stage is to identify the nouns and noun-like phrases. (In practice you would highlight the nouns and verbs in different colours on the same document, but so that the analysis can be read if printed in monochrome, we here print the analysis twice, with first the nouns, and then the verbs, highlighted.)

1. Highlighting nouns and noun-like phrases

You are asked to design a **system** for an **artists' cooperative**. **Artists** submit their **paintings** (catgorised as **Water Colour**, **Oils** or **Mixed Media**) to the **cooperative** and from time to time **exhibitions** are arranged. Each exhibition has a **name** and relates to a single **art category** and occurs at a **named venue** (for which we need to have a **contact number** and the **size of the permanent display area**). An exhibition may be sponsored by one or more **local companies**, that are approached from time to time by **email**. Each exhibition is coordinated by one of the **members** of the cooperative. There is no **guarantee** that a **submitted piece of work** will appear in an exhibition, but most do, and many appear in several **locations**.

The **organisation** has a long **tradition** of mentoring new **artists**. It is not compulsory but all new **members** are encouraged to have a **mentor**, and all members can choose to have one. There is no special **qualification** for being a mentor and it is left to the good **sense** of the **people** involved. The **collective** do, however, want a **record** kept of these unofficial mentoring links.

Item	Entity	Attribute	Notes
system			What we are making - not part of it
artist	Artist		Need to record attributes of Artist (some still to determine) so will be an entity
cooperative			We are producing the software for the cooperative
paintings	Painting		Need to record things like title and medium - check for other attributes. So is an entity
water colour etc		Category	An attribute of Paintings Also an attribute of Exhibition
exhibition	Exhibition		Need to record name of exhibition, type and venue, so this is an entity
name (of exhibition)		Name	Attribute of Exhibition
art category		Category	An attribute of Exhibition. Also an attribute of Paintings
(named) venue	Venue		Venue is an entity which has attributes including name, contact no and size of display area
contact number		Contact	Attribute of Venue (also of Member and Sponsor)
size of display area		Disp_Size	Attribute of Venue
local companies	Sponsor		These are sponsors about whom we need to record at least contact details. Decide on a Sponsor entity
member			Same entity as Artist
guarantee			Explanatory word; not part of the data
locations			Same meaning as Venue
organisation			Same meaning as cooperative
tradition			Explanatory word - not part of the data
mentor			Mentors are members, so no additional entity needed. Mentoring status will be shown by relationships
qualification			Explanatory word; not part of the data
sense			Explanatory word; not part of the data
people			Same meaning as Artist
collective			Same meaning as cooperative
record			An explanatory word. No need to keep records about records

2. Highlighting verbs and verb-like phrases

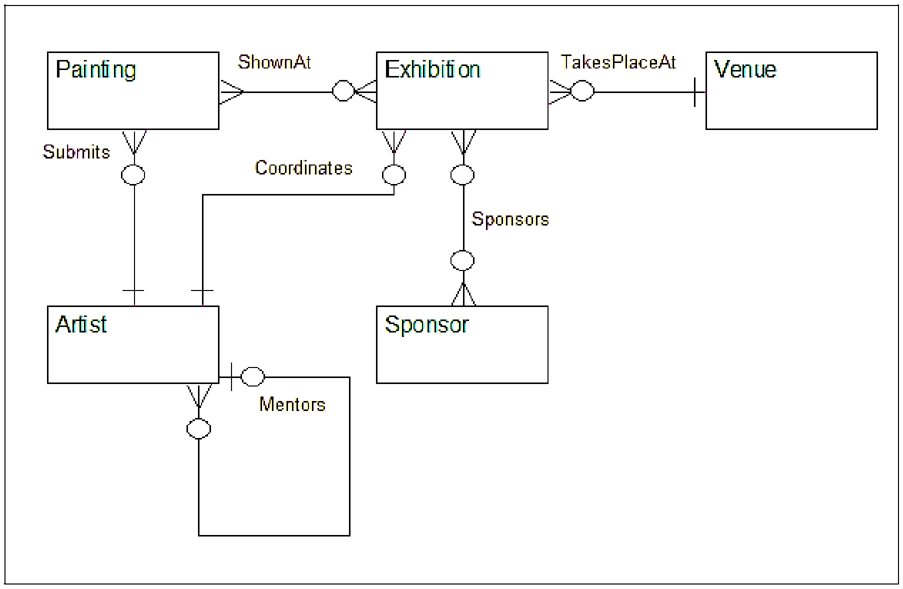
You are asked to **design** a system for an artists' cooperative. Artists **submit** their paintings (**catgorised** as Water Colour, Oils or Mixed Media) to the cooperative and from time to time exhibitions **are arranged**. Each exhibition has a name and **relates to** a single art category and **occurs** at a named venue (for which we **need to have** a contact number and the size of the permanent display area). An exhibition may be **sponsored by** one or more local companies. Each exhibition **is coordinated by** one of the members of the cooperative. There is no guarantee that a submitted piece of work **will appear** in an exhibition, but most **do**, and many **appear** in several locations.

The organisation **has a** long tradition of **mentoring** new artists. It is not compulsory but all new members **are encouraged** to have a mentor, and all members **can choose** to have one. There is no special qualification for **being a mentor** and it is **left to the good sense** of the people involved. The collective do, however, **want** a record kept of these unofficial **mentoring links**.

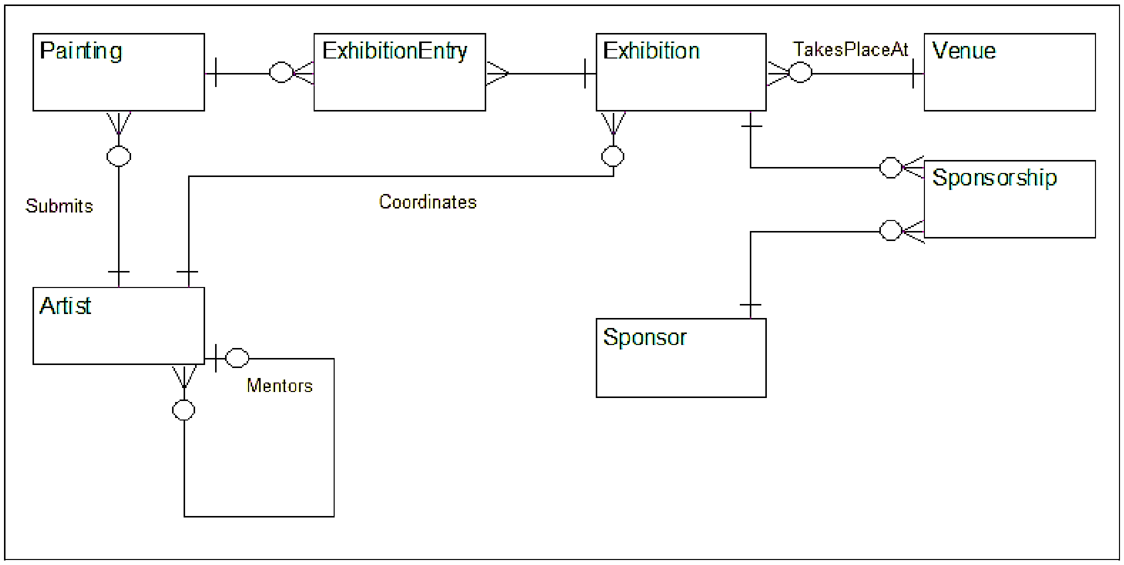
Item	Relationship	Notes
design		Explanatory word; not part of the data
submit	Submits	The relationship between an artist and a painting
relates to (category)		Shown by attribute Category

are arranged		New instance of Exhibition entity
occurs (at venue)	TakesPlaceAt	
may be sponsored by	Sponsors	Relationship between Sponsor and Exhibition
is coordinated by	Coordinates	Relationship between Artist and Exhibition
(will) appear	ShownAt	Relationship between Paintings and Exhibition
most do		Explanatory phrase; not part of the data
has a long tradition		Explanatory phrase; not part of the data
are encouraged		Explanatory phrase; not part of the data
can choose		Explanatory phrase; not part of the data
being a mentor	Mentors	Reflexive relationship of Artist
left to the good sense of		Explanatory phrase; not part of the data
want a record kept		Explanatory phrase; not part of the data

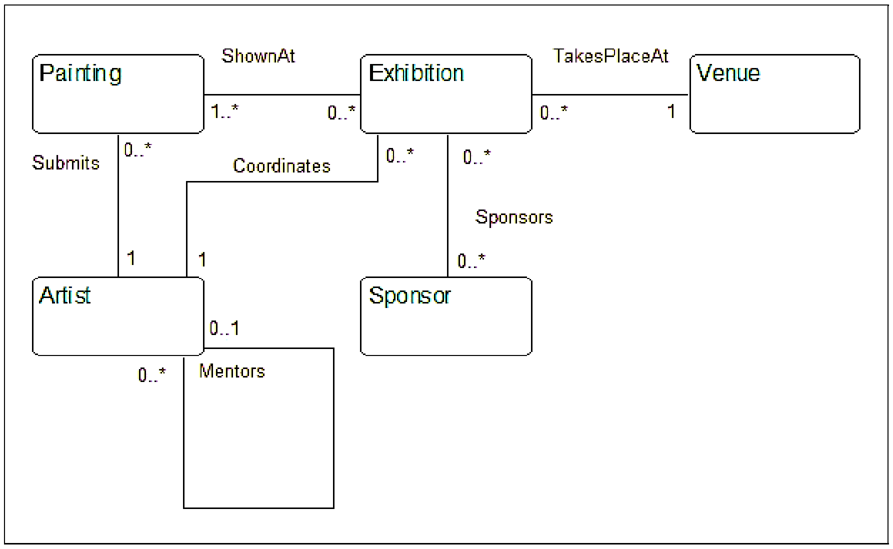
We then express our initial analysis into a 'real world' entity relationship model of the data. This model includes many-to-many relationships and it may correctly exclude entities that will eventually be needed in order to find homes for all the identified attributes. In this first cut model we are concerned with the real world entities and the relationships between them. Here is our interpretation of the summary requirements of the Artists' Cooperative system.

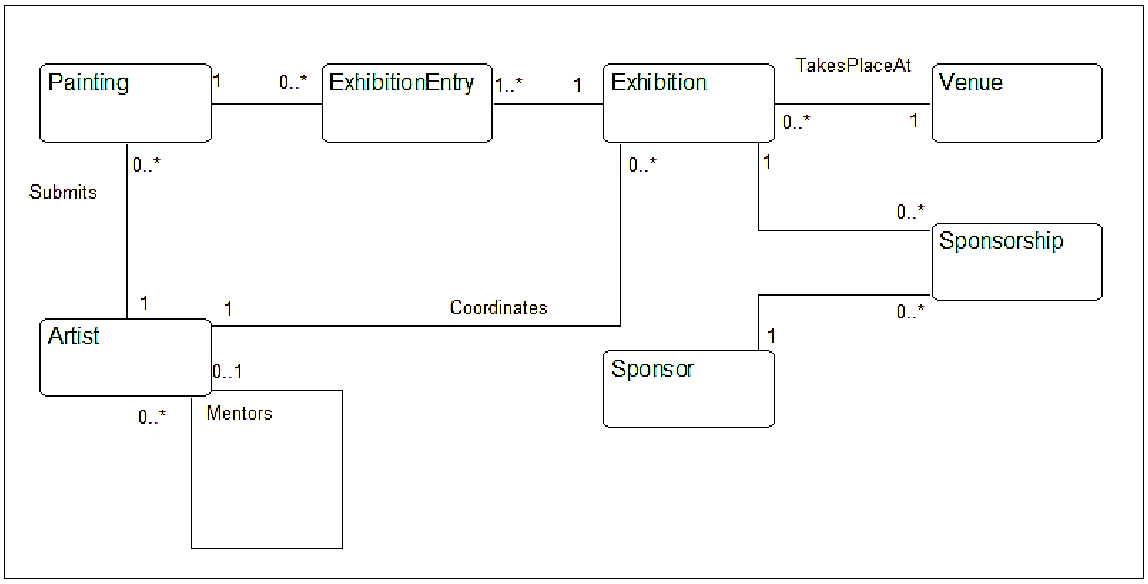


The conversion of the real world model into a logical relational model follows the procedure discussed in section 2.3.3. We identify all many-to-many relationships and replace them with intersection entities. Here is our version of the logical relational model.



The symbols and methodology we use for drawing entity relationship diagrams is one that is widely used and is supported by most Computer Aided Software Engineering (CASE) tools and most drawing packages. If, however, you have difficulty drawing diagrams using those conventions, or you are already used to drawing class diagrams, you can instead use the following notation which uses numbers to show cardinality and participation.





Exercise 7

Carry out a similar analysis of the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] and:

- list the potential entities and attributes;
- list the potential relationships;
- draw the entities and real world relationships as an entity relationship model;
- draw the logical relational structure as an entity relationship model.

Although this is an individual exercise, and you should not post your complete analysis or diagram in the Unit 2 forum [link: http://study.conted.ox.ac.uk/mod/forum/view.php?id=13615], we would like you to discuss any difficulties or points of interest that arise. There is usually some scope for different interpretations and representations and we would particularly like you to post messages about aspects of the exercise that present apparent ambiguity or the need to choose between equally valid representations.

At an appropriate time we will post a message in the forum providing you with our diagrams and with a response to any points that have arisen in your forum discussion.

2.3.5 Normalisation – concepts

Normalisation is the process of organising attributes into entities and relationships. A set of normalisation rules are applied to the attributes, producing a succession of 'Normal Forms' which ensure that the entities and relationships produced conform to the relational model.

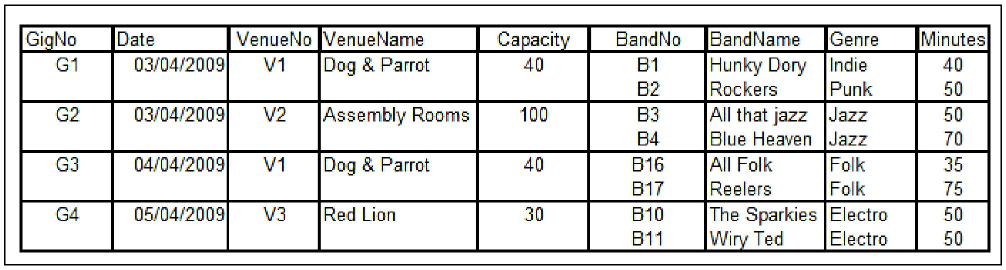
A full understanding of the normalisation process requires familiarity with relational theory, which we do not cover in detail until Unit 3, but the broad concept can be understood using the insights obtained from entity relationship modelling. Though we return to the subject of normalisation in Unit 3, it is properly introduced here because it is often undertaken in parallel with, or immediately after, entity relationship analysis.

The theory of normalisation involves working on all the attributes in the system, as if they were attributes of a single entity. This is clearly not a practical proposition in most projects, and we will shortly examine a less reliable, but more realistic approach, but it is best to start with the theoretical basis, even if that does mean using unrealistically small data sets to illustrate the method.

There are five or six Normal Forms (stages of normalisation) but many of these are highly theoretical and the most important, and most practical, are the first three. In this topic we study normalisation up to third normal form.

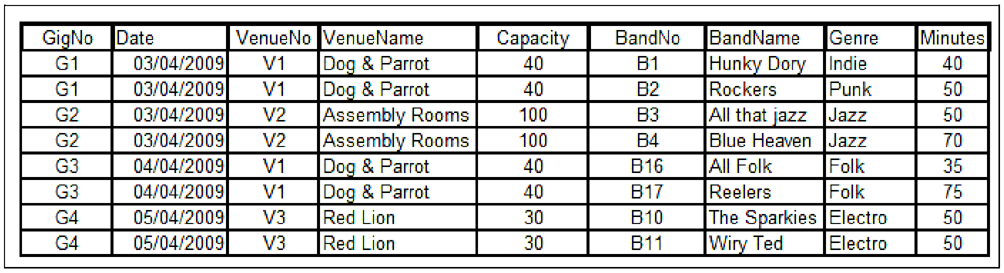
First Normal Form

The definition of First Normal Form (1NF) is that a data set, which in our theoretical example means a table, must not have repeating groups. This reflects a relational principle we met in entity relationship modelling. An attribute value must be atomic: it cannot be a multivalued fact. Many raw data sets have repeating groups. Here is an example based on a sub set of the Gigs scenario you met earlier in this topic.



GigNo	Date	VenueNo	VenueName	Capacity	BandNo	BandName	Genre	Minutes
G1	03/04/2009	V1	Dog & Parrot	40	B1	Hunky Dory	Indie	40
					B2	Rockers	Punk	50
G2	03/04/2009	V2	Assembly Rooms	100	B3	All that jazz	Jazz	50
					B4	Blue Heaven	Jazz	70
G3	04/04/2009	V1	Dog & Parrot	40	B16	All Folk	Folk	35
					B17	Reelers	Folk	75
G4	05/04/2009	V3	Red Lion	30	B10	The Sparkies	Electro	50
					B11	Wiry Ted	Electro	50

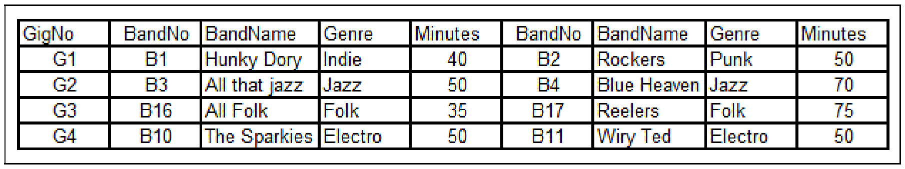
In this table the BandNo, BandName, Genre and Minutes attributes hold values that involve repeated values. These repeating groups must be removed so that the table is in 1NF before we can continue with the normalisation process. In this exercise (which is to provide insight rather than a practical technique) we do so by producing a rather strange table.



GigNo	Date	VenueNo	VenueName	Capacity	BandNo	BandName	Genre	Minutes
G1	03/04/2009	V1	Dog & Parrot	40	B1	Hunky Dory	Indie	40
G1	03/04/2009	V1	Dog & Parrot	40	B2	Rockers	Punk	50
G2	03/04/2009	V2	Assembly Rooms	100	B3	All that jazz	Jazz	50
G2	03/04/2009	V2	Assembly Rooms	100	B4	Blue Heaven	Jazz	70
G3	04/04/2009	V1	Dog & Parrot	40	B16	All Folk	Folk	35
G3	04/04/2009	V1	Dog & Parrot	40	B17	Reelers	Folk	75
G4	05/04/2009	V3	Red Lion	30	B10	The Sparkies	Electro	50
G4	05/04/2009	V3	Red Lion	30	B11	Wiry Ted	Electro	50

It is an odd looking table and not one you would introduce in a practical analysis, but it now satisfies the rule for 1NF. It does not including repeating groups or multi-valued cells.

Note that the way the repeating groups are set out in a table is not relevant. It is the presence of repeating groups that prevents a table being in 1NF. The following table appears to have atomic attribute references, but that is because it uses repeated columns for the same attributes. It is not in 1NF because it contains repeating groups of attributes.



GigNo	BandNo	BandName	Genre	Minutes	BandNo	BandName	Genre	Minutes
G1	B1	Hunky Dory	Indie	40	B2	Rockers	Punk	50
G2	B3	All that jazz	Jazz	50	B4	Blue Heaven	Jazz	70
G3	B16	All Folk	Folk	35	B17	Reelers	Folk	75
G4	B10	The Sparkies	Electro	50	B11	Wiry Ted	Electro	50

If you were certain there could never be more than two bands you might decide to create a database which included a table like this (renaming the second group of columns, for example BandNo to Band2No) but that would be a later decision to 'unnormalise' the data. During analysis and initial design data should always be normalised.

Second Normal Form

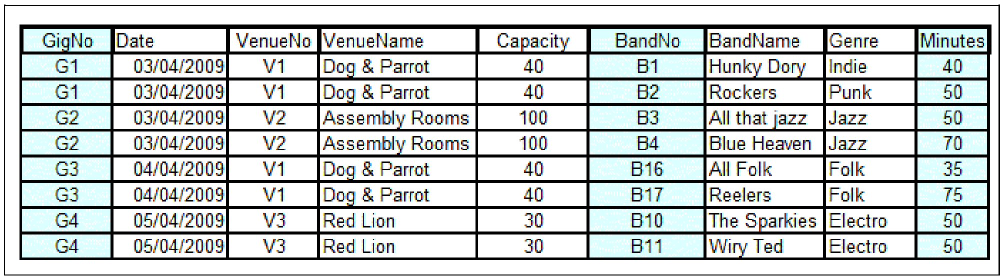
To be in Second Normal Form (2NF) a table must be in 1NF, and all non-key columns must be dependent on the whole of the table's key. Determining the key of a data set can be a difficult process, which we will be exploring in Unit 3, but here the example tables have been designed to make it easy to see the key to the 1NF table.

Exercise 8

We find the key by identifying a column or group of columns that can uniquely identify a particular row in the table. In the 1NF table we produced above it is possible to identify a row by specifying the values of two attributes. Can you identify them?

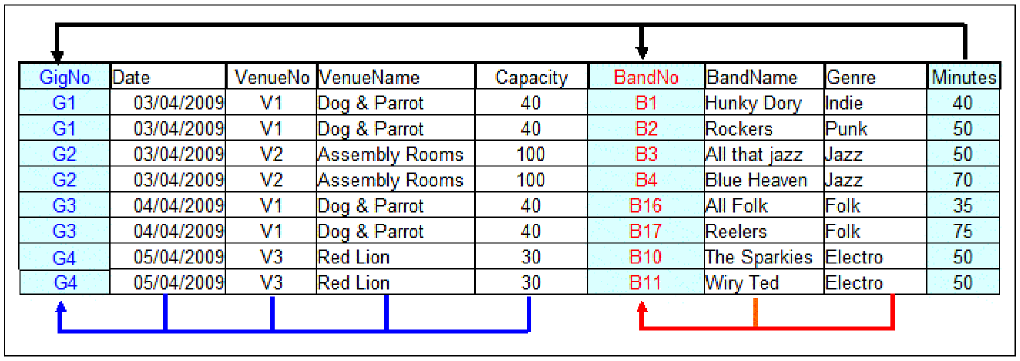
Tutor's comments

If you know the gig and you know the band, then you can find out the number of minutes the band is playing. Knowing just the band will not give this information, nor will knowing just the gig. You need to know them both. So the key to this table is a joint key consisting of GigNo and BandNo.



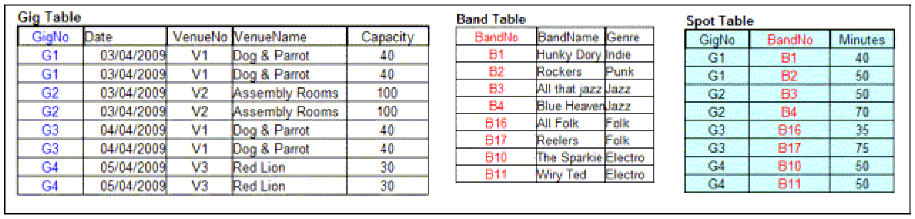
GigNo	Date	VenueNo	VenueName	Capacity	BandNo	BandName	Genre	Minutes
G1	03/04/2009	V1	Dog & Parrot	40	B1	Hunky Dory	Indie	40
G1	03/04/2009	V1	Dog & Parrot	40	B2	Rockers	Punk	50
G2	03/04/2009	V2	Assembly Rooms	100	B3	All that jazz	Jazz	50
G2	03/04/2009	V2	Assembly Rooms	100	B4	Blue Heaven	Jazz	70
G3	04/04/2009	V1	Dog & Parrot	40	B16	All Folk	Folk	35
G3	04/04/2009	V1	Dog & Parrot	40	B17	Reelers	Folk	75
G4	05/04/2009	V3	Red Lion	30	B10	The Sparkies	Electro	50
G4	05/04/2009	V3	Red Lion	30	B11	Wiry Ted	Electro	50

Now we have found the key to the table we can see that the Minutes column is the only dependent column that is dependent on the whole of the primary key. That is the only column that cannot be defined without knowing both parts of the key. Every other dependent column is dependent on part of the key, for example, Date and VenueName are dependent only on GigNo. If you know the gig number you can find its date and venue. Similarly, BandName and Genre are dependent only on BandNo. For this table to be in 2NF we would have to remove all the columns that do not depend on the whole of the primary key. We do this by creating two new tables each of which has as its key field one of the fields in the original key. In our example GigNo will be the key to one of the new tables, and BandNo the key to the other. We then need to identify the appropriate dependent columns, for example:



GigNo	Date	VenueNo	VenueName	Capacity	BandNo	BandName	Genre	Minutes
G1	03/04/2009	V1	Dog & Parrot	40	B1	Hunky Dory	Indie	40
G1	03/04/2009	V1	Dog & Parrot	40	B2	Rockers	Punk	50
G2	03/04/2009	V2	Assembly Rooms	100	B3	All that jazz	Jazz	50
G2	03/04/2009	V2	Assembly Rooms	100	B4	Blue Heaven	Jazz	70
G3	04/04/2009	V1	Dog & Parrot	40	B16	All Folk	Folk	35
G3	04/04/2009	V1	Dog & Parrot	40	B17	Reelers	Folk	75
G4	05/04/2009	V3	Red Lion	30	B10	The Sparkies	Electro	50
G4	05/04/2009	V3	Red Lion	30	B11	Wiry Ted	Electro	50

The lines at the top and bottom of the table show the dependencies and enable us to convert the data set into three tables.



Gig Table

GigNo	Date	VenueNo	VenueName	Capacity
G1	03/04/2009	V1	Dog & Parrot	40
G1	03/04/2009	V1	Dog & Parrot	40
G2	03/04/2009	V2	Assembly Rooms	100
G2	03/04/2009	V2	Assembly Rooms	100
G3	04/04/2009	V1	Dog & Parrot	40
G3	04/04/2009	V1	Dog & Parrot	40
G4	05/04/2009	V3	Red Lion	30
G4	05/04/2009	V3	Red Lion	30

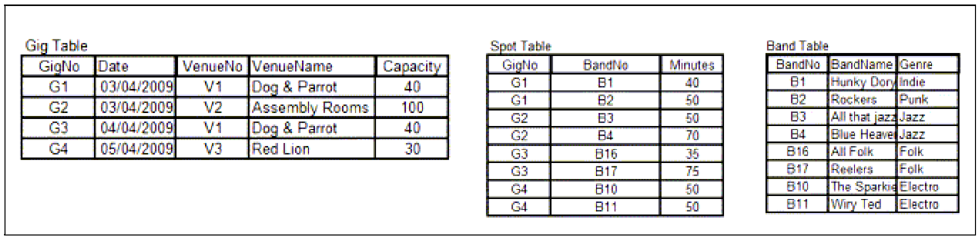
Band Table

BandNo	BandName	Genre
B1	Hunky Dory	Indie
B2	Rockers	Punk
B3	All that jazz	Jazz
B4	Blue Heaven	Jazz
B16	All Folk	Folk
B17	Reelers	Folk
B10	The Sparkie	Electro
B11	Wiry Ted	Electro

Spot Table

GigNo	BandNo	Minutes
G1	B1	40
G1	B2	50
G2	B3	50
G2	B4	70
G3	B16	35
G3	B17	75
G4	B10	50
G4	B11	50

Some of these tables may now have duplicate rows, and these must be removed before the data is in 2NF.



Gig Table

GigNo	Date	VenueNo	VenueName	Capacity
G1	03/04/2009	V1	Dog & Parrot	40
G2	03/04/2009	V2	Assembly Rooms	100
G3	04/04/2009	V1	Dog & Parrot	40
G4	05/04/2009	V3	Red Lion	30

Spot Table

GigNo	BandNo	Minutes
G1	B1	40
G1	B2	50
G2	B3	50
G2	B4	70
G3	B16	35
G3	B17	75
G4	B10	50
G4	B11	50

Band Table

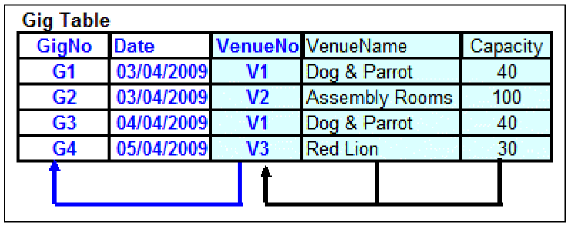
BandNo	BandName	Genre
B1	Hunky Dory	Indie
B2	Rockers	Punk
B3	All that jazz	Jazz
B4	Blue Heaver	Jazz
B16	All Folk	Folk
B17	Reelers	Folk
B10	The Sparkie	Electro
B11	Wiry Ted	Electro

Third Normal Form

A table is in Third Normal Form (3NF) if it is in 2NF and all dependent fields are dependent only on the key field(s) of the table. To put it the other way round, no field should be dependent on a field that is not a key field. Looking at the Band Table we can see this rule is satisfied. Both the name of the band and its genre are dependent only on BandNo. Both are facts about a band and nothing else.

If you look at the Gig table, however, you will find some columns that are dependent on non key fields. VenueName and Capacity are dependent on GigNo (it is because the gig number is G1 that we know the venue name is Dog & Parrot and the capacity 40) but they are also dependent on VenueNo. Venue number V3 will always find Red Lion, whatever the value of GigNo.

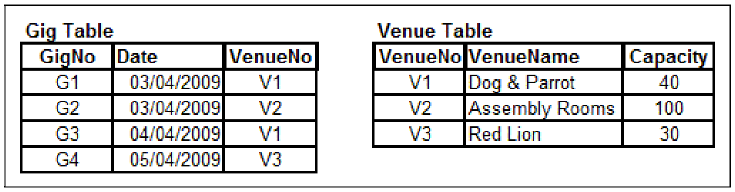
Another way of looking at 3NF is to say that a table must not contain transitive dependencies. You could say VenueName and Capacity are dependent on GigNo only transitively, since it is only through their dependence on VenueNo that they are dependent on GigNo. You can identify transitive dependencies in the same way we identified dependencies in arriving at 2NF.



Gig Table

GigNo	Date	VenueNo	VenueName	Capacity
G1	03/04/2009	V1	Dog & Parrot	40
G2	03/04/2009	V2	Assembly Rooms	100
G3	04/04/2009	V1	Dog & Parrot	40
G4	05/04/2009	V3	Red Lion	30

Again we retain the key to the table (*GigNo*), and any fields wholly dependent on it (*Date and VenueNo*), but we move into a new table the transitively dependent fields (*VenueName and Capacity*), together with a copy of the field they are dependent on (*VenueNo*) producing:



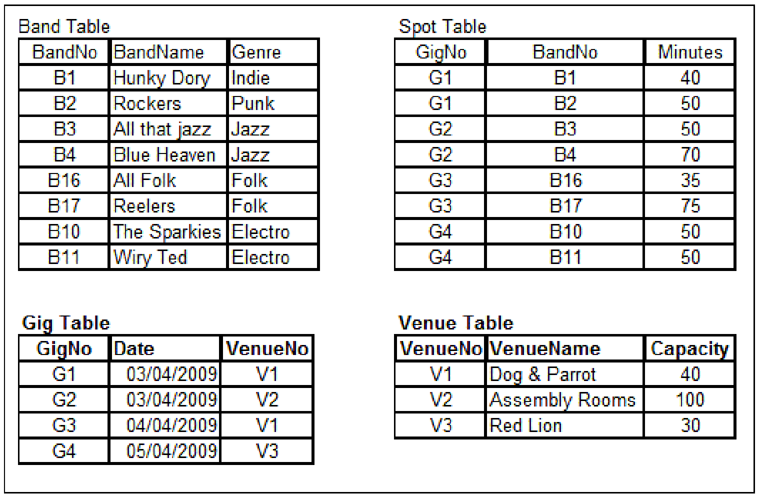
Gig Table

GigNo	Date	VenueNo
G1	03/04/2009	V1
G2	03/04/2009	V2
G3	04/04/2009	V1
G4	05/04/2009	V3

Venue Table

VenueNo	VenueName	Capacity
V1	Dog & Parrot	40
V2	Assembly Rooms	100
V3	Red Lion	30

The attributes in the data set have now been organised into normalised tables:



Band Table

BandNo	BandName	Genre
B1	Hunky Dory	Indie
B2	Rockers	Punk
B3	All that jazz	Jazz
B4	Blue Heaven	Jazz
B16	All Folk	Folk
B17	Reelers	Folk
B10	The Sparkies	Electro
B11	Wiry Ted	Electro

Spot Table

GigNo	BandNo	Minutes
G1	B1	40
G1	B2	50
G2	B3	50
G2	B4	70
G3	B16	35
G3	B17	75
G4	B10	50
G4	B11	50

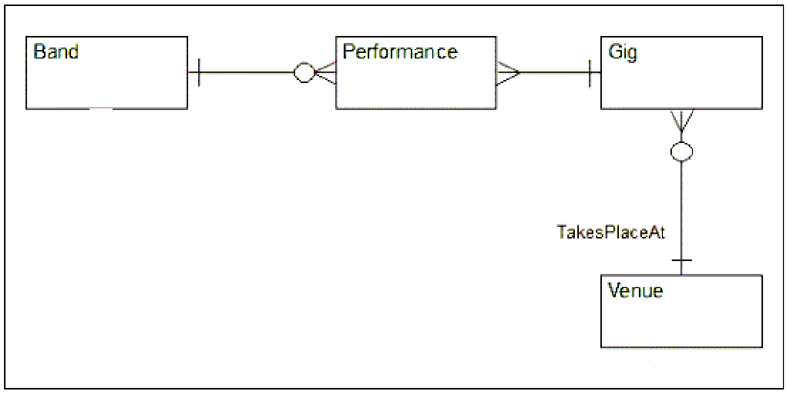
Gig Table

GigNo	Date	VenueNo
G1	03/04/2009	V1
G2	03/04/2009	V2
G3	04/04/2009	V1
G4	05/04/2009	V3

Venue Table

VenueNo	VenueName	Capacity
V1	Dog & Parrot	40
V2	Assembly Rooms	100
V3	Red Lion	30

which are the same tables we arrived at using entity relationship modelling. (As we here used a sub set of the data, we do not have the Musician and LineUp tables.)



Though this approach to normalisation is valuable for the insight it provides, it is clearly not a practical proposition when analysing a real system or designing a substantial database. In the next section we examine how the principles learned in this section can be applied in practice.

2.3.6 Normalisation - methods

The normalisation process, through strict normal forms, involves the sorting of data by applying successively the rules of first, second and third normal forms and assumes you have collected together the data attributes in the system into one data set. But if the number of attributes were large it would be difficult to understand the data and impossible to document it in a single table. For this reason practical normalisation is often undertaken in stages on groups of data. Where possible the analysis is carried out on completed forms, card index records, diaries, lists etc. currently used in the system. Where they are not available, lists of apparently related attributes are drawn up. The attributes in each of these documents are normalised, and the resulting tables rationalised and merged into the final tables.

Here are some documents used by the organisation that runs the entertainment business, part of which we modelled in the previous section. Though still very simplified, they contain more data than we had available in the last section.

Gig Record Card	
Gig Number	45
Date	23/5/2009
Venue	Dog and Parrot
Contact	0123678
Capacity	40
Start Time	19:30
Entry Fee	£5

Bands booked for the gig			
Band	Genre	Website	Minutes
Hunky Dory	Indie	www.hunkey.com	40
Ranters	Metal	www.Rants.com	40
Rollers	Rock	www.roll.com	50

Band record card	
Band Number	22
Name	Hunky Dory
Genre	Indie
Phone	0019283
Website	www.hunkey.com
Email	hunky@dory.com

Band members		
Musician	Phone	Instrument
George Thompson	01234	Lead Guitar
Bill Giles	01345	Piano
Joe Cluny	01348	Drums

First Normal Form

The first step is to list the attributes of each set of data, identifying (usually by insetting) any attributes that could involve multiple values. From the Gig Record Card we can draw up this list:

GigNumber (PK)
Date
Venue
Contact
Capacity
StartTime
EntryFee

Band
Genre
Website
Minutes

We have assumed for now that GigNumber would be the primary key field, and check that all other attributes are dependent on it. But it is clear the table does not satisfy the rule for first normal form. There are repeating groups of attributes (or you could say, there would be multiple values in a cell). We therefore extract the repeating items into another table, together with a copy of the key field they depend on.

Gig Table	Spot Table
GigNumber PK	GigNumber JPK
Date	Band JPK

Venue	Genre
Contact	Website
Capacity	Minutes
StartTime	
EntryFee	

Now we do the same for the band record card:

BandNumber
Name
Genre
Phone
Website
Email

Musician(full name)
Phone
Instrument

Band Table	Musician Table
Band Number PK	BandNumber JPK
Name	Musician JPK
Genre	Phone
Phone	Instrument
Website	
Email	

The attribute Musician is clearly going to cause us some problems. There is a case for renaming it to MusicianName. It may be, however, depending on the semantics of the data and the nature of the use cases, that MusicianName would be a multiple value. If so the Musician Table is still not in first normal form. We will decide that, in this system, full names are multiple values, so will split the Musician attribute into FirstNames and LastName Since we need the full name to identify a musician, both new attributes become part of the primary key.

Band Table	Musician Table
Band Number PK	BandNumber JPK
Name	LastName JPK
Genre	FirstNames JPK
Phone	Phone
Website	Instrument
Email	

You will no doubt see weaknesses in the data structures we have arrived at, but they are in first normal form.

Second Normal Form

We now examine our tables to see if they conform to the rule for second normal form. You will remember that in order to be in second normal form all non-key attributes must be dependent on the whole of the primary key.

There is no need to consider the tables with a single field primary key because they will automatically be in second normal form.

Look at the Spot Table and trace the dependencies of each attribute:

Spot Table	Dependencies
GigNumber JPK	Part of PK
Band JPK	Part of PK
Genre	Band
Website	Band
Minutes	GigNumber and Band

Though Minutes is dependent on the whole primary key (the duration of this band’s set at this gig) the Website and Genre attributes are dependent on only part of the primary key. They must be removed from the table so that it is in second normal form. As always, when we take attributes out of a table, we take with them the part of the primary key they are dependent on. This produces the new tables:

Spot Table	Band (2) Table
GigNumber JPK	Band PK
Band JPK	Genre
Minutes	Website

Now we turn out attention to the Musician Table, and analyse its dependencies:

Musician Table	
BandNumber JPK	Part of PK
LastName JPK	Part of PK
FirstNames JPK	Part of PK
Phone	Musician
Instrument	BandNumber and Musician

The presence of an attribute that is not dependent on the whole of the primary key prevents the table from being in second normal form, so we have to remove that attribute, together with the part of the key it relates to. Here the Phone attribute, though not dependent on the whole primary key, is dependent on more than one field. That means we take a copy of both these fields into the new table:

Musician Table	Musician (2) Table
BandNumber JPK	LastName JPK
LastName JPK	FirstNames JPK
FirstNames JPK	Phone
Instrument	

Before we move on to consider third normal form there is usually a need for some rationalisation of the tables. The tables we have at the moment are:

Gig
Spot
Band
Band (2)
Musician
Musician (2)

Where tables have the same name we need to merge their attributes, if that is possible, or rename tables to avoid ambiguity. Sometimes, as in the case of the band tables, here, they can be merged into a single table:

Band Table	Band (2) table	Table 2 attributes
Band Number PK	Band PK	Merge with Band Number
Name	Genre	Merge with Genre
Genre	Website	Merge with Website
Phone		
Website		
Email		

The other tables with ambiguous names, are less straightforward because they cannot be fully merged:

Musician Table	Musician (2) Table
BandNumber JPK	LastName JPK
LastName JPK	FirstNames JPK
FirstNames JPK	Phone
Instrument	

Though you could merge the LastName and FirstName attributes, Phone cannot be part of the first Musician table because it is not dependent on the whole primary key. So here, rather than merge we rename. The second Musician table is the one that ought to bear that name. It is about musicians. The first table is more about the relationship of musicians with bands and their instruments. It would be better to rename this table. A suitable name would be Lineup.

If you have not addressed it earlier (it is a matter of tactics and judgement when you address these issues) you should at this point consider the suitability of first and last names as the primary key. In most cases you would opt for a unique musician identifier. The Lineup and Musician tables would then become:

Lineup Table	Musician Table
BandNumber JPK	MusicianNumber PK
MusicianNumber JPK	LastName
Instrument	First Names
	Phone

We now have the following tables in second normal form:

Gig
Spot
Band
Lineup
Musician

Third Normal Form

Each of the tables is now examined to see whether there are any non-key or transitive dependencies.

Gig Table	Spot Table	Band Table	Lineup Table	Musician Table
GigNumber PK	GigNumber JPK	Band No PK	BandNo JPK	MusicianNo PK
Date	Band JPK	Name	MusicianNo JPK	LastName
Venue		Genre	Instrument	First Names
Contact		Phone		Phone
Capacity	Minutes	Website		
StartTime		Email		
EntryFee				

Ignore the Gig Table for the moment and look at the others. All attributes in these tables are dependent only on the primary key, so they are already in third normal form.

In the Gig Table, however, there are some attributes (Contact and Capacity) that are dependent on the non-key field Venue. Removing these transitively dependent fields provides another table:

Gig Table	Venue Table
GigNumber PK	Venue PK
Date	Contact
Venue	Capacity
StartTime	
EntryFee	

Again you would, at this point, consider whether Venue (which in the example documents holds the name of the venue) is an adequate primary key.

Normalised tables

We have once again ended up with the same tables produced by entity relationship modelling:

Gig
Band
Musician
Spo
t Lineup
Venue

Instinct and experience

It is important to understand the theoretical basis for normalisation, and the step by step approach of the methodology in this section. We would go further and suggest that, like an airline pilot's pre-flight check, it is worth working through each of the normal forms routinely on any analysis. It has to be said, however, that in practice many analysts can see almost at a glance the broad structure of the tables required, and their first draft of tables may already be in third normal form. If you take this approach, you cannot rely entirely on instinct, however, and you should check each of your final tables for their adherence to the rules of normal forms.

Other normal forms

Beyond third normal form, there are other rules, many of them esoteric, to determine whether a data set is in Boyce-Codd Normal Form, and in 4th, 5th and 6th Normal Forms.

In order to understand these advanced normal forms you need to know more about relational theory than we have covered in this unit. For the purposes of this course and its assignments there is no need for you to consider normalisation beyond third normal form, but if you are interested, we would be happy to discuss higher normal forms in the

Unit 3 forum, after you have studies more relational theory.

2.4 Dynamic Modelling

2.4.1 Introduction to dynamic models

Dynamic process models focus on processes and the sequence in which they take place. The characteristics of these models are:

- Process elements are identified, which may be many and various, including processes, events, entities, tasks and activities.
- Links showing sequences between process modules are identified, which may also be many and various, including hierarchical connections, direction arrows and data flows.
- They focus on processes and data.
- The basic, logical dynamics of processing need to be modelled, including:
  - **Sequence**, that is, the order in which process modules are processed;
  - **Selection**, that is, the choice between two processing sequences (programmed using if…then…else constructs); and
  - **Iteration**, that is, the same process being repeated (programmed using for…do, while…do and repeat…until constructs).

You need to know about the models and methodologies covered in this section, because you are likely to meet them when maintaining, adapting or extending legacy systems. Most contemporary analysis and design does, however, use object oriented dynamic models, which we will cover in Unit 4.

Process or event?

The modelling of processing may focus on the identified processes, and how they associate and progress over time, or on the events that drive the processing, where the events that happen to the system and cause processes to run are identified and the sequence of processing is based on that. Instead of identifying the various transactions that describe the processing of orders in the system, we might for instance identify the events that affect the life of an order. These events might be the creation of the order, its acceptance, update, fulfilment and closure. We can associate operations or low-level modules with these events that are similar to those identified for transactions, for example, enter order header details, enter customer details, enter product order details, validate customer details and check credit limit/outstanding balance. We can see that we should develop a similar set of operations, whatever our approach, and that these different techniques could corroborate and complement each other. If you are going to use only one approach then using events is probably best, as it ties in with the modelling of inputs and outputs used in user interface design, which is also primarily event driven.

2.4.2 Dynamic process modelling - Jackson structures

Section learning outcomes

After completing this section you will have knowledge of the following:

1. The necessary elements for dynamic process modelling.
2. Process-focused design using Jackson structure modelling.

Introduction to topic

There are innumerable dynamic modelling techniques and in this topic we will look at three in some detail and some others in passing. We will look in detail at Jackson process design, entity life histories and activity modelling. Between them, these exemplify the fundamental concepts used in dynamic processes, namely process dynamics (sequence, selection and iteration) and the use of process-, event- and data-driven and task or activity approaches to process design and specification.

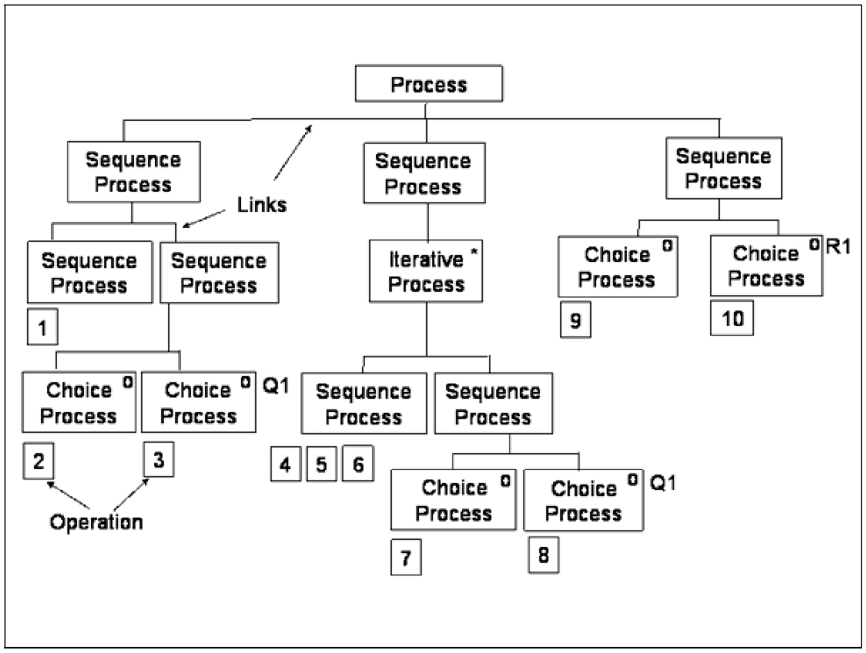
Jackson structures

Jackson structures are named after their inventor, Michael Jackson, and come from his Jackson Structured Programming method, which he originally developed in the 1970s and documented in *Principles of Program Design (Jackson, 1975)* . The technique incorporates the concepts of process dynamics and models each process as a combination of sequence, iteration and selection. Jackson structures (the diagrams the modelling technique produces) have been adopted and developed for a range of techniques, and we shall first look how they can be used for simple process design, and then consider their application to the production of entity life histories.

Every process has a beginning, a middle and an end, and the Jackson method focuses on analysing and designing the beginning and the end of a process and then extrapolating the middle (especially when designing, and specifying, application code). So, in essence, it considers what the process needs to achieve and what it is given to start with. The design is developed so that the logic of the process is fully explored and the basic operations identified. This means that it is quite low level as a process specification technique, although the processes could be developed as a part of a hierarchy.

The structure and elements of a Jackson diagram is shown in Figure 1.

Figure 1. The Basics of a Jackson Structure Diagram



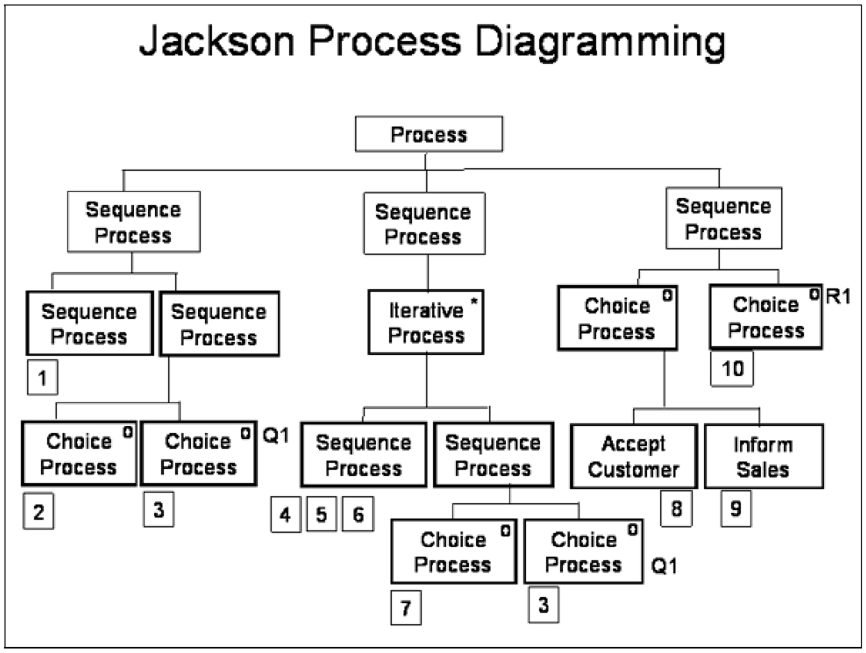
The diagram is basically a collection of boxes combined sequentially. The sequence is read from left to right across the diagram. Excluding the very top box, which identifies the process that is being modelled, there are the types of boxes denoting **sequence**, **selection** (identified by the 'o' in the top-right corner of the box) and **iteration** (identified by the '*' in the top-right corner of the box). These boxes fall into two basic groups: those that are intermediate and have other boxes linked below them and those that are at the bottom of a hierarchy, often identified as 'the leaves on the tree'. The latter model the processing that takes place and the sequence. They can be thought of as modules.

The basic structural elements are:

- **Process identification.** At the top is a box identifying which process is being modelled.
- **Sequence.** A sequence is shown as a series of sequence boxes connected by horizontal linking lines. The boxes are in sequence and are read from left to right.
- **Hierarchy.** Each of these modules may be developed into a hierarchy of further modules, and so on down through various levels.
- **Selection.** For selection you need two or more selection boxes at the same level on the diagram to show that there is a choice involved in the processing. Any hierarchy below a selection box is part of the selection, and if that selection is not made the modules in the hierarchy will not be processed.
- **Iteration.** For iteration the iteration module and any hierarchy below it will iterate 0, 1 or many times.
- **Operations.** Beneath those modules, which are the leaves on the tree, there are numbered boxes which identify operations. The operations concern such activities as reading and writing data to variables, files and databases, performing validation, receiving input and sending output. Operations are listed separately.
- **Level restrictions.** Only boxes of the same type can exist at the same level of a hierarchy on the diagram, that is sequence, selection and iteration boxes cannot be mixed.
- **Quits and resumes.** These are indicated by *Qn* and *Rn* on the diagram where *n* is an integer. The process sequence in the diagram may be left at the quit point and resumed at the resume point on the diagram with the same integer identifier. This type of jump in program design is not thought good practice and should be avoided if possible. Typically, quits and resumes are used when a module causes the process to terminate and the closing code of the process is needed to close it properly.

We can develop a program specification from this structure by selecting the leaves on the tree of the diagram (the lowest-level boxes in the hierarchy) plus the intermediate sequence, selection and iteration boxes immediately above them. This is shown in Figure 2, where the relevant boxes have been emboldened.

Figure 2. Programming Sequence



We can produce the program definition by first creating a list of operations (briefly described) in the order in which they are processed on the diagram. We can then use some form of design language, such as structured English or pseudocode, to insert the selection and iteration commands at the appropriate places. An outline program specification based on Figure 2 is shown in Table 1.

Table 1. Outline Program Specification.

(Program Specification for Process (*identifier*))
1. Operation 1 (description)
2. If (selection test)
2.1 Then Operation 2 (description)
2.2 Else Operation 3 (description)
3 While\For\Repeat (iteration selection test)
3.1 Operation 4 (description) *) These may*
3.2 Operation 5 (description) *) be in*
3.3 Operation 6 (description) *) any order.*
3.4 If (selection test)
3.5 Then Operation 7 (description)
3.6 Else Operation 3 (description)
4. If (selection test)
4.1 Then Operation 8 (description)
4.2 Operation 9 (description)
4.3 Else Operation 10 (description)

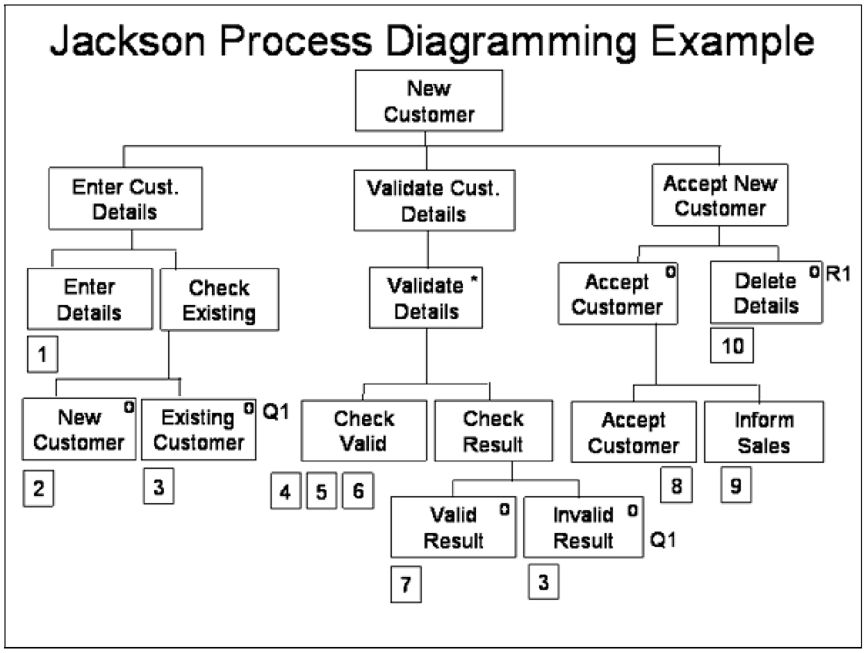
Notice that if several operation are assigned to the same module on the diagram, then the sequence of these operations is not defined for that module. If it is important, they should be placed under separate sequence boxes. Finally, notice we have used a structured line-numbering system, as this is essentially for future reference and review.

We will now look at an actual example to better exemplify the method.

An example of a Jackson structure

Figure 3 shows a simple example of a Jackson structure with Table 2 showing the operations list and Table 3 the outline program specification.

Figure 3. Jackson Structure Example



This model is based on the following scenario.

The New Customer process inputs a new customer to the system. When the new customer data is input it is placed in a temporary file and the customer details are checked to see if this is really a new customer or an existing customer. Here a selection process is required. If it is not a new customer, the file is deleted. If it is a new customer, the details are validated iteratively detail by detail. If the details are sufficient and validated, then the customer is accepted as a new customer. If not, the file is deleted.

To construct this model, something like the following method would be used.

To construct the diagram:

1. Identify the modules using the requirements documentation and any higher level process models developed for this area. Typically there will be start, middle and end modules but there may be others, depending on the complexity of the process involved. Any data to process cross-referencing will be consulted, as will the data indicate the processing.
2. Build the structure. The hierarchy for each of the high-level modules will be developed, first for those modelling the input and output. The Jackson method first designs structures for the input and output (the beginning and end of the process, and it then merges them and develops the middle processing.
3. Operations may be identified either as the model progresses or as a separate exercise at the end. The project will have standards for operation definition.
4. The diagram is checked for internal consistency.
5. It is reviewed of accuracy and completeness.

To construct the outline program specification:

1. The operations are listed and described (see Table 2).
2. The process route through the model is highlighted.
3. The program specification is produced using some type of definition language (see Table 3).
4. The specification is reviewed using basic inspection or structured walkthroughs.
5. The diagram and the specification are revised.
6. The Jackson structure is maintained. This is essential because, as other structures for the system are developed, they will impact or be impacted by this structure.

Table 2. New Customer Operation.

Operation 1 (read in customer details to temporary file)
Operation 2 (set valid initial valid flag)
Operation 3 (set invalid flag)
Operation 4 (check postcode)
Operation 5 (call check creditworthy function)
Operation 6 (check other mandatory input)
Operation 7 (set details valid flag)
Operation 3 (set invalid flag)
Operation 8 (write customer details to database)
Operation 9 (call new customer report function)
Operation 10 (delete temporary file)

Table 3. Outline Program Specification.

(Program Specification for Process (*New Customer*))
1. Operation 1 (read in customer details to temporary file)

```
2. If (new customer)
2.1 Then Operation 2 (set valid initial valid flag)
2.2 Else Operation 3 (set invalid flag)
3 While\For\Repeat (initial valid flag true and details to process)
3.1 Operation 4 (check postcode)
3.2 Operation 5 (call check creditworthy function)
3.3 Operation 6 (check other mandatory input) )
3.4 If (results acceptable)
3.5 Then Operation 7 (set details valid flag)
3.6 Else Operation 3 (write customer details to database)
4. If (details valid flag true)
4.1 Then Operation 8 (write customer details to database)
4.2 Operation 9 (call new customer report function)
4.3 Else Operation 10 (delete temporary file)
```

If we review the above outline structure, we can see there are a few issues that will need resolution.

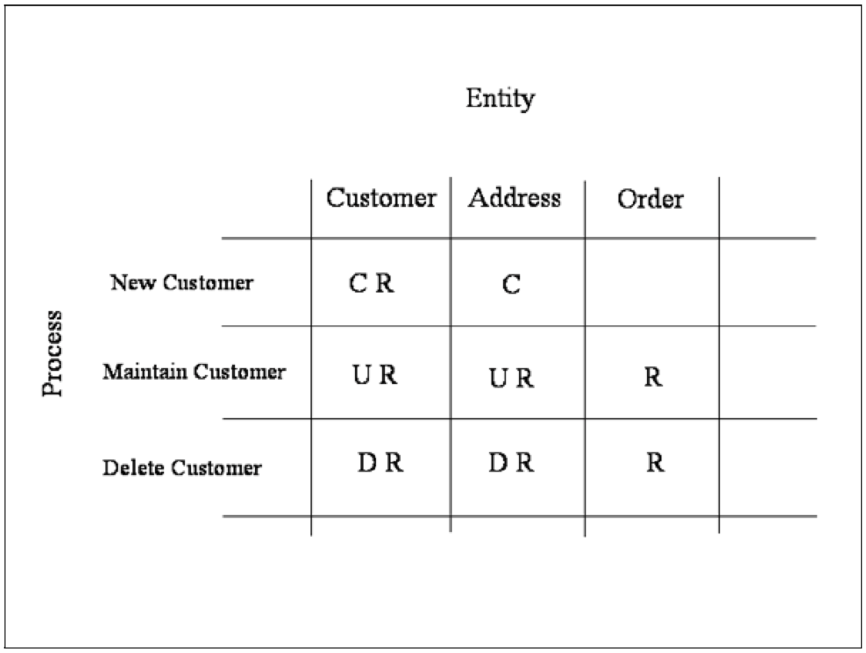
- Firstly, how will we know whether the new customer is already an existing customer? The first 'if' test, in line 2, tests whether the customer already exists, but how does the program know that? The answer is that it does not, and we need a further operation to run a check of the customer name against the customer database. We may also need to then refine our search if we find a match to checking the postcode and other data. This sounds like quite extensive coding, and coding that may well be used elsewhere, so it is probably a separate function and our operation would be something like 'Operation 1a (call customer validation function)'. We would expect this function to set a flag saying whether the customer exists on the database. Notice that this operation is numbered '1a' as it is difficult to renumber every operation. (A CASE tool would be useful.) This is why we highlight the sequence box immediately above the choice modules for this selection as it reminds us that, for all but the simplest choices, there may be some further operations to be defined.
- There are similar issues with the other 'if' statements and we need to develop operations for them as well.
- We do not seem in the rest of the code to use the invalid flag set in line 3. We use the valid flags, but the invalid flag and Operation 3 seem redundant.
- Consider the end of the model. If we accept the new customer, we do not delete the temporary file. There may not really be a selection here but rather a sequence of the Accept Customer module (and its lower-level sequence modules) followed by the Delete Details module. This will change the code specification.

You may think of other issues, but those issues described here show again that review and update is a massively important tool in modelling techniques. The act of modelling and the rules of the technique have forced us to ask the relevant questions. This first-cut design has given us a good start to our program specification and something valid to work with and improve. The use of a diagram to develop the high-level definition enables the design to be developed quickly, with the lower-level detail being added at the end. If we had begun by trying to identify the operations before the overall structure, we would not have succeeded.

Finally, we consider in a little more detail how we get the information to draw the structure. There are many ways, but they all essentially involve decomposition from higher-level diagrams. Some options are:

- Use the data flow models and take the lowest-level process boxes as our starting-points for these structures.
- Produce higher-level Jackson-type structures (in which, for example, New Customer would be a leaf on the tree).
- Use entities to identify the need for processes for creating, updating, reading and deleting them. For this type of process development, we can use a **CRUD matrix** (see Figure 4).

Figure 4. The CRUD Matrix



Entity

Process

	Customer	Address	Order	
New Customer	C R	C		
Maintain Customer	U R	U R	R	
Delete Customer	D R	D R	R	

This matrix plots entities (or other data objects) against functions or processes. In each of the intersecting cells it identifies whether the process Creates, Reads, Updates or Deletes the data. Hence the term **CRUD matrix**. This matrix is useful as a first step in identifying modules.

Processes are usually divided into update and enquiry processes and have separate process models. Enquiry activities, unless required as part of an update activity, will complicate the models, as typically they can happen at almost anytime and do not add any meaning to the sequence in the model.

This has been a brief look at the Jackson method. There is much more to it than we have covered here, but this review has conveyed the basics of the techniques. We will look at it further in the next section where we will use Jackson structures to model events using entity life histories.

Exercise 1

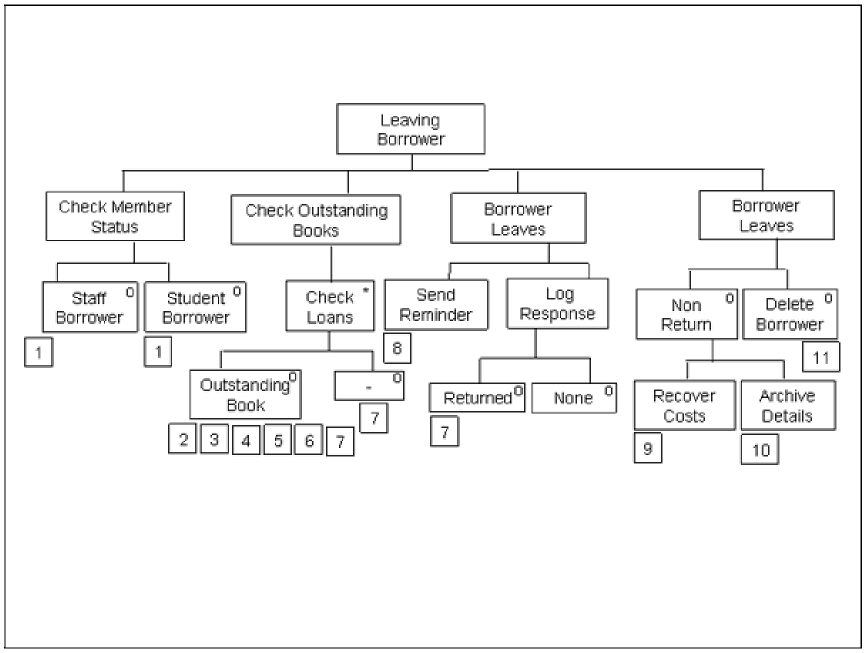
Using the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] case study, produce a Jackson structure to design the process for a borrower leaving the library. This process is about checking whether the borrower has returned all their books and paid their fines.

I found this challenging and my solution will certainly not be the best design, so please discuss it with you colleagues. As it is difficult, do not get too involved in the detail; just have a go at the problem, learn the techniques and see how it really forces you to think about what is happening in the process. The information you have is insufficient for you to understand fully what is required, so do use your imagination.

Tutor's comments

This is my answer and I have made several assumptions, for example about archiving borrower details if books have not been returned. A useful strategy here would have been to have had one structure for students and another for staff, as theirs would have been simpler.

Figure E1. Jackson Structure for Leaving Borrower Process in the Library System Case Study.



My operations are:

1. set borrower status indicator
2. call loans read function
3. check book cost
4. check fines
5. total book costs
6. total fines
7. overdue books and fines cleared flag
8. reminder sent flag
9. deduct cost from fines deposit
10. archive borrower details
11. delete borrower details

Section review

Based on the learning outcomes in this section you should have knowledge of the following:

1. The necessary elements for dynamic process modelling.

The necessary elements are:

- Process dynamics with sequence, selection and iteration.
- Support decomposition to enable different levels of detail and complexity to be modelled.

2. Process-focused design using Jackson structure modelling.

   A Jackson structure has the following characteristics:

   - Models processes
   - Models sequences (with diagrams that are read from left to right)
   - Based on input and output
   - Identifies operations
   - Can be used as the basis for program specification
   - Used in design
   - Can be decomposed

   The basic method is:

   - Identify the modules.
   - Build the structure. First, design structures for the input and output (the beginning and end of the process), and then merge them and develop the middle processing.
   - Identify and describe the operations.
   - Check for internal consistency.
   - Review for accuracy and completeness, and revise.

2.4.3 Dynamic process modelling - event modelling

Section learning outcomes

After completing this section you will have knowledge of the following:

1. Event focused design using entity life history modelling.
2. Other data focused process modelling techniques including effect correspondence diagramming, enquiry access paths and walkthroughs.

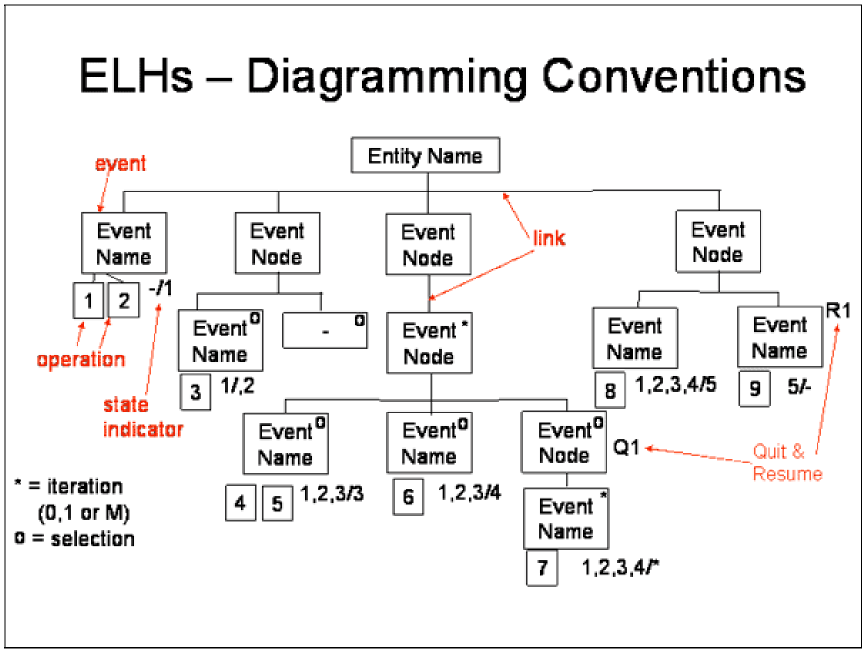
Entity life histories (ELHs)

An **entity life history (ELH)** uses a Jackson structure to model how the entity is processed during its life in the system – that is, to model, for example, the life of an order. The ELH model uses events that affect the entity during its existence in the system, from creation to deletion.

In this context, an **event** is an action that causes the state of the system to change (which in essence means a change to the data) and is a trigger for a process, such as the receipt or cancellation of an order, or an update to an order. The modelling of events is another way of identifying the processing that occurs in a system, as all processing requires events to activate it. As we change the state of the system, we are modelling update processing and ELHs are developed for enquiries.

An ELH diagram looks and works exactly like a normal Jackson structure as can be seen from Figure 5, which shows the basic elements of an ELH.

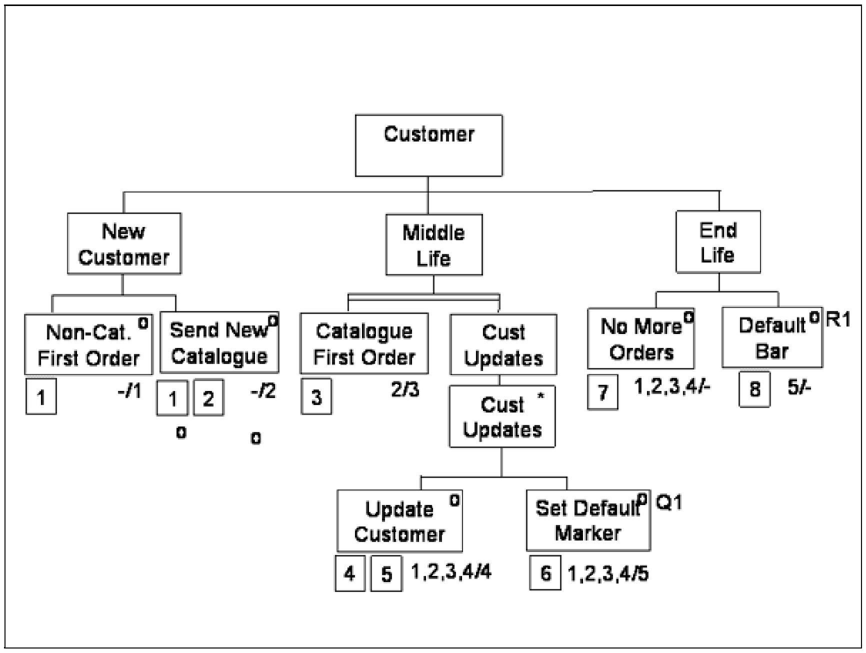
Figure 5. The Elements of an ELH



ELH example

The ELH example is of the life of a Customer entity and is shown in Figure 6. Here a customer is first logged on the system due to either a first order or the sending of a catalogue to them. If a customer is logged as a potential customer, on receipt of a catalogue, their data will be updated differently if they eventually send in a first order, as not all data is kept for potential customers. (For example, no credit limit is set.) This may happen at any time, while the customer details may also be maintained due to changes in circumstances (such as a change of address). These two actions can happen in any order and are said to have parallel lives. They are not sequential. This is modelled on the diagram as a bar of two parallel lines, under the Middle Life node. The updating of the customer entity is iterative and there is one update that will result in the person no longer being a customer of the company and this is the default marker. This is set if the customer has defaulted on payments and the company has decided to bar them. There is a quit for this event and a resume at the Default Bar event that ends the customer data. Finally, under the Middle Life node, if the customer does not place any orders for a set period of time they are removed as a customer.

Figure 6. Entity Life History of a Customer



The method for drawing this diagram is as follows.

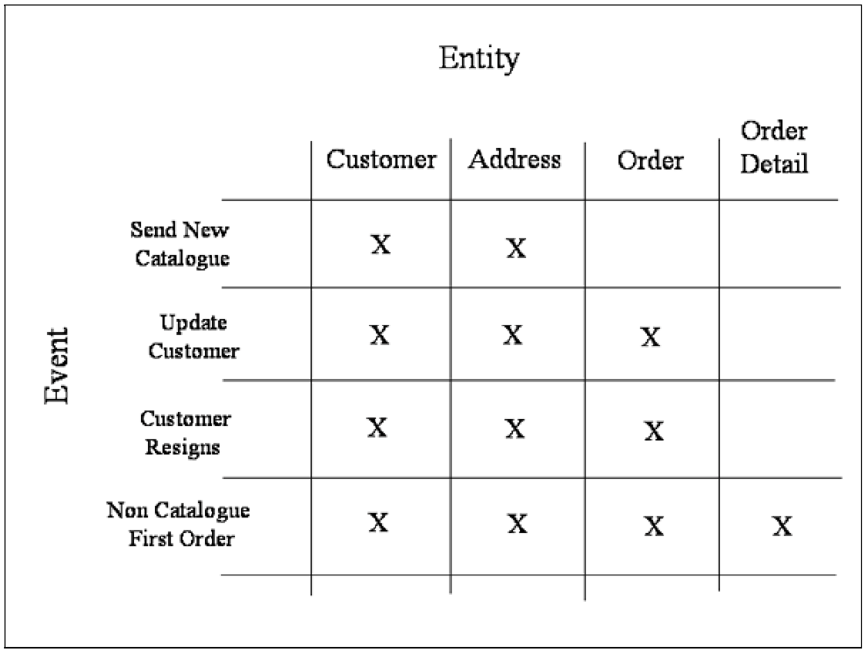
1. **Identify events**. These can be identified from the DFM process descriptions and the data flows passing into them, and also from other functional definitions.
2. **Draw entity event matrix**. To identify which events affect which entities, the entity-relationship model should be inspected by following the relationships the event processing will have to navigate. (An example is shown in Figure 7.) This is a type of data model walk through and these are dealt with in Figure 8 and the associated description in the text.
3. **Draw diagram**. This is done using Jackson conventions. It is best to start by drawing the simple life ignoring all the possible difficulties and errors. Once the simple life has been completed, the complexities can be added.
4. **Add operations and state indicators**. Once this has been produced, the operations and state indicators can be added. When adding state indicators, the end state

indicators are added first and the initial state indicators added later.

5. **Review and refine diagram**. The diagram and any accompanying descriptions of operations are reviewed for consistency and accuracy. Again the events and the entity modelled should be articulated against the entity relationship model to check that every change that can occur to the data is included.

As a rule, code specifications are not written for operations developed from ELHs, although they can be. This is because the structuring of code modules does not necessarily focus around entities. Usually the operations are reviewed and rationalised and then grouped into modules. For example, the operation (3) under Catalogue First Order is more likely to form part of an order process than a customer process, even though it updates Customer.

Figure 7. ELH Entity Event Matrix



Entity

Event	Customer	Address	Order	Order Detail
Send New Catalogue	X	X		
Update Customer	X	X	X	
Customer Resigns	X	X	X	
Non Catalogue First Order	X	X	X	X

In reviewing the design I would highlight one issue. What happens if the customer is rejected? Currently *nothing* happens. But the client might wish to contact the customer or, if the new system takes telephone orders, the problem may be resolved instantly. We can model these new scenarios using these diagrams and instead of using them just for design we can use them in analysis, to elicit and model requirements. Typically, Jackson program designs are used only in design and specification, but ELHs like DFM, is a tool for both analysis *and* design.

This next bit is optional and is if you really want to get serious with state indicators.

State indicators

Quite often, these take a lot of time to get right but if they are used they are a good way of checking and reviewing the logic of the diagram, and ensuring that it has been drawn correctly. The best way to demonstrate how they are used is to go through the example event by event.

Non-Catalogue First Order. This is the beginning so the state indicator starts from a null value and is set to 1 as it is the first event on the diagram.

Send New Catalogue . This is the second possible start event. It and Non-Catalogue First Order can create the Customer entity instance. Its state will start at null but be set to 2 as the Non-Catalogue First Order event may have been set to 1 already.

Catalogue First Order . This can happen at any time during the middle life but it can only happen once. It will only happen if the Send New Catalogue event has occurred, so its initial state can only be 2 if it is to be allowed to happen. The state will be set to 3, as this is the next unused value.

Update Customer . This can happen iteratively and could happen after any of the previous events so its initial values the first time round could be 1, 2 or 3; once an update has occurred, its state is set to 4. The next time this happens, as it has already occurred once, the initial state will be 4 as the state indicator was set to that. So the initial state on the diagram could be 1, 2, 3 or 4. Every time an update is completed, its state will be set to 4, which indicates we are in the middle life updating the data.

Set Default Marker . This will only happen once, but it may happen any time a customer update occurs as it is inside the iteration. Like Update Customer, it can occur after any of the previous events: 1, 2, 3 or 4. Its state will be set to 5 if the event occurs. The quit against it will direct the processing to the Default Bar event which ends the entity life.

No More Orders . This event could occur after any of the previous events, except for Set Default Marker which quits to Default Bar. The iteration for customer updates may never happen as any iteration on these diagrams can happen 0, 1 or many times. So its initial state would be 1, 2, 3 or 4. It will be set to null as it will be the last event in the life.

Default Bar. The event will end the life if the default marker has been set. Its initial state will be 5 as this is the final state of the Set Default Marker event.

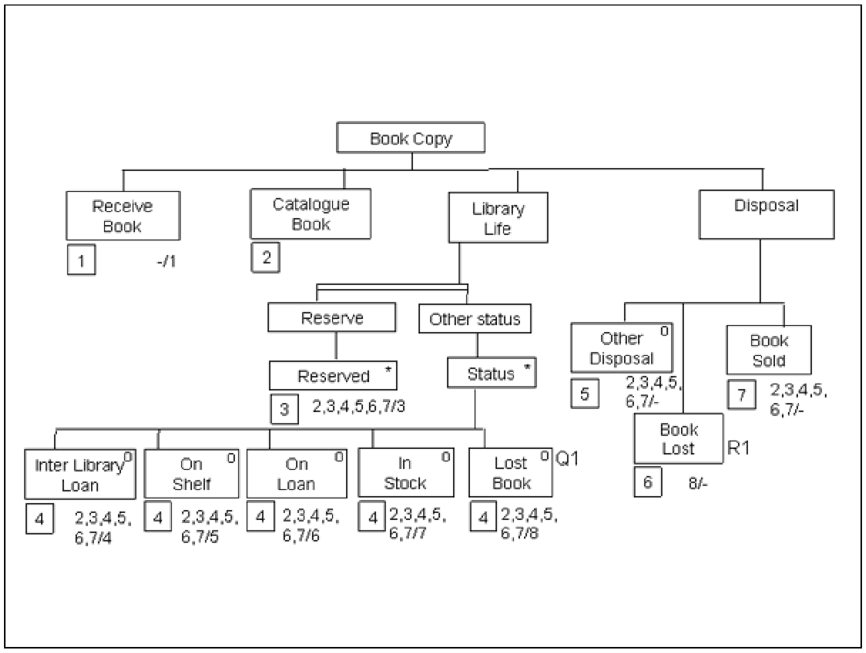
Exercise 2a

Using the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] case study, produce an entity life history for the entity Book Copy. Assume that the book may have several status during its life in the library– for example, On Shelf, On Loan, In Stock, etc. – and that it may be disposed of in several ways.

Tutor's comments

My answer is shown below:

Entity Life History for Book Copy



Exercise 2b

As a supplementary exercise, review the model answer to Exercise 2a and see if you can spot the problem I can. You may spot others as well. Review it in collaboration with your colleagues.

Tutor's comments

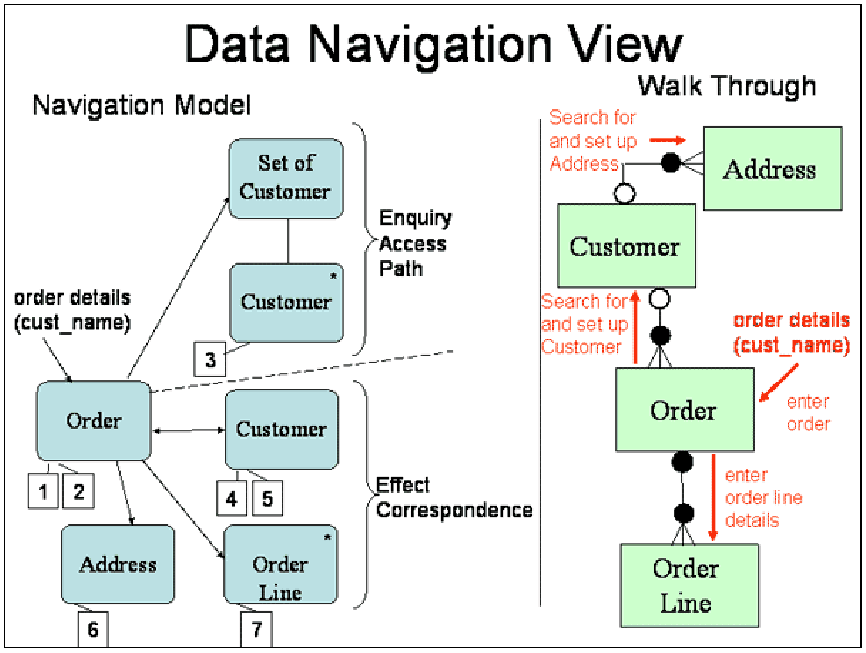
If you follow this diagram you will see that the book can be catalogued and never have a status as iteration allows for zero. So, a book can be catalogued and then disposed of. I do not think this should be the case, and there should be a sequence box after catalogue book that sets the first status, which might be either On Shelf or Reserved.

Data navigation

ELH is an update processing method but we also need to model enquiry as well as update processing, which mainly involves data-focused methods as enquiries are data focused. We will look at two of them, but we will first consider another update model called effect correspondence diagramming.

Effect correspondence diagramming. Here we take the opportunity to explore the opposite view to the ELH, where the entities affected by an event are structured in a diagram that uses sequence, selection and iteration. This is called an **effect correspondence diagram**. We can see from the entity event matrix in Figure 7 that an event can affect more than one entity. Figure 8 shows the effect correspondence diagram for the Receive First Order event, which can potentially change the state of four entities.

Figure 8. Data Navigation Models.



We can see that the event creates a new Order and, based on this, it will interrogate Customer. If there is a new customer, it will create a new Customer entity instance, and it will also create the associated order lines, shown iterated, and set up a new customer address and delivery address. To check whether the customer exists, the set of customer entity instances must be interrogated, and this is modelled on an enquiry version of this type of diagram called an **enquiry access path.** Here the set of Customer entity instances is shown and the individual iterated instances shown underneath. Not all the detail is on the diagram, so some description is required, but the model shows that the associated entities can be navigated by the event process. Operations are also modelled on these diagrams and will cross-correlate with those on the ELH models. Producing these types of diagrams is covered in greater detail in Weaver *et al*. (1998).

Enquiry access paths are one form of data enquiry navigation model. An older, but still valid, technique is to navigate the entity-relationship model or some other form of data model. (For example, class association models in an object-oriented framework show this navigation using arrows on the diagram.) This navigation through the model is known as a **walkthrough** and an example for the order process is also shown in the Figure 8. Here, the given order data enables the Order entity instance to be created using the Customer data to corroborate a new customer. A new Customer instance can be created and from that a new Address instance and the OrderLine instances can also be created. All these entities are linked to Order or to Customer, which is linked to Order, so the data can be set up and navigated using the data provided for the order. The walkthrough shows how the process can be completed using the data structure available.

Section review

Based on the learning outcomes in this section you should have learnt the following:

1. Event-focused design using entity life history modelling.

   ELH have the following characteristics:

   - They model events that trigger changes of state in the system.
   - They are diagrammed like Jackson structures.
   - They are directly linked to and reviewed against the data structure.
   - They identify operations.
   - They develop the status indicators used for monitoring the state of an entity and reviewing the logic of the diagram.
2. Other data-focused process-modelling techniques including effect correspondence diagramming, enquiry access paths and walkthroughs.

   Effect correspondence diagrams are used to model the entities associated with an event in a quasi-Jackson-style diagram. They show the navigation through the data to complete the event.

   Enquiry access paths are the equivalent of effect correspondence diagrams but for enquiries. They have the same structure.

   For a process, a walkthrough will navigate the data model to check that the data required by the process can be input, updated or enquired against as required.

2.4.4 Dynamic process modelling - activity modelling

Section learning outcomes

After completing this section you will have knowledge of the following:

Activity-focused design and business-based activity models.

Activity modelling

Activity modelling is a dynamic process modelling technique that can model from the higher business levels down to lower levels, as it allows for decomposition. Currently it is used in object-oriented analysis and design, but similar techniques are used in business modelling so it has a wide range of applications and can be understood by a wide community of business and IT staff.

Typically it will model the activities associated with a business process or event. An activity is an atomic action describing a state of the process or an event, for example, ‘generate invoice’. Conceptually, a process is viewed as being a collection of activities that define the state of the process at a particular time. So, for making a cup of tea they might be: get cup, put in tea-bag, pour in water, add milk, add sugar, stir, remove tea-bag. Each activity is a distinct step in the process and the state of the process moves on a distinct step with its completion. An activity can be composite and decomposed into lower-level activity diagrams to define necessary detail.

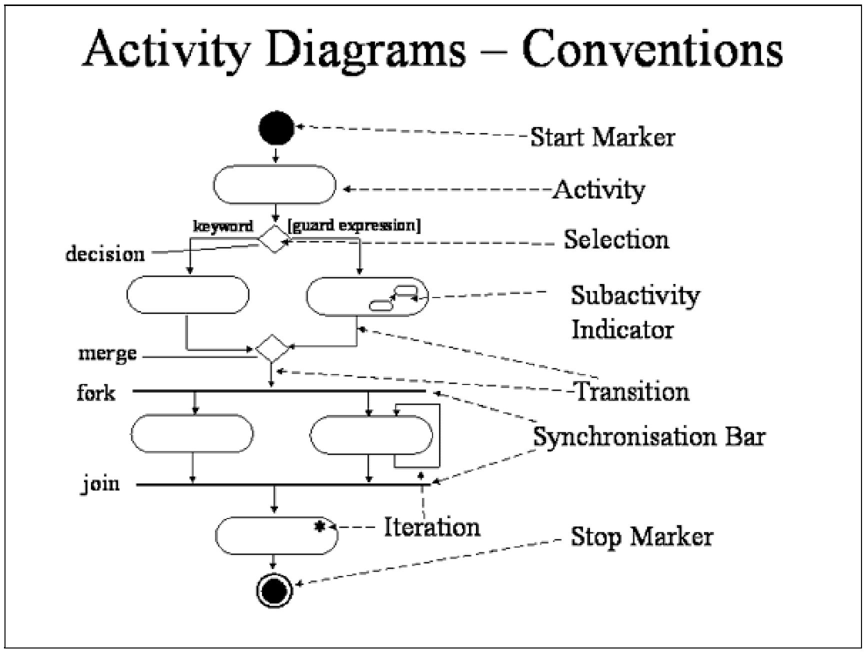
Activity modelling models the transition between activity states. The diagrams are drawn so that the sequence is shown flowing vertically from top to bottom. The basic elements and structure of the diagram are shown in Figure 9.

Activities are modelled as oval boxes with the name of the activity inside. They are connected by transition lines which have arrows to show the direction the transition takes. Transition flows can split or merge and the point at which they do so is shown by a small, diamond-shaped box. When they split, it means that a selection has been made and that the Boolean or control command that tests for the selection is written on the diagram as a keyword(s), whilst the business reason can be written as a **guard expression**. Iteration can be shown *either* by a transition flow returning to an activity (or an identified area of the diagram) *or* by an activity being asterisked.

Synchronisation bars are used to denote areas of the diagram where the activities may occur at any time in relation to each other. (This is essentially the same as the parallel lives on an ELH.) The synchronisation bar that shows the start of the area is called a **fork** and that closing the area is called a **join**. The area between the bars may be iterated as a whole.

The points of entry into and exit from the process are shown as **start** and **stop markers**. Finally, the fact that an activity has been decomposed into lower-level activity models is shown by the symbol that shows two linked activities; this is the **subactivity indicator**.

Figure 9. The Basic Structure of an Activity Diagram.



The flow of the diagram occurs through the transitions, and a transition will only occur when an activity has been completed. At that point a transition to the next activity in the sequence will take place. The flow of the process will then halt at this next activity until it is completed. It is easy to see why this technique is useful for modelling tasks and workflow in business.

An example diagram is shown in Figure 10. Here an order is received and the customer is checked to see if they are valid. This is modelled as the decision-point where a selection is made. The keyword used is ‘else’ and the guard expression is ‘invalid customer’. If the customer is invalid, a transition occurs to the stop marker and the process is ended. If customer is valid, the order is set up and processed. Whilst the order is being processed, the invoice can be generated. These two activities may take place during the same time-frame, with the order being processed and the invoice generated concurrently. This processing is shown by the synchronisation bars. The fork shows the start of the two activities and the join shows the end of their period of activity. The processing of the order, and the raising of the invoice, may be iterative as the order and invoice lines are fulfilled. This is shown by a transition flow going from the join synchronisation bar to the fork bar, showing that there may be a transition from the end of synchronisation to the beginning. Finally, once the order processing and generation of the invoice are complete, the order is dispatched and the process stops.

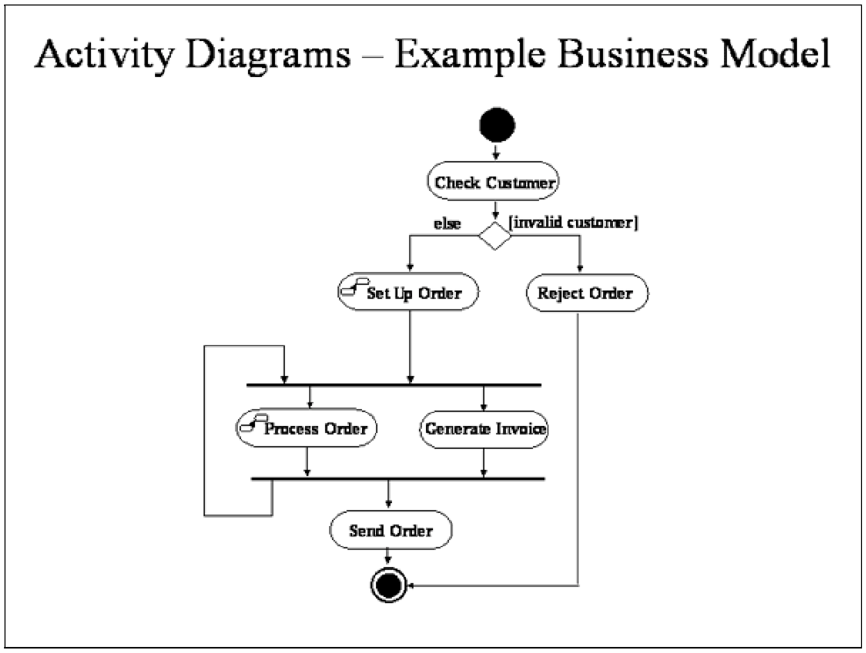
The method for producing this model is as follows:

1. For the process or activity identified from process descriptions (for example, from the data flow model) identify the basic activity states and the transitions between them.
2. Begin to draw an activity diagram, producing the simple, straightforward version of the process first. Try not to make the diagram too complicated and remember that decomposition can enable the modelling of complex areas at a lower level. Try to keep the activities at the same process level. For example, if you are going to model an activity called ‘maintain car’, you will not have other activities, such as ‘change oil filter’ and ‘check brakes’, at the same level. Identify points of decision, iteration and

parallel activity.

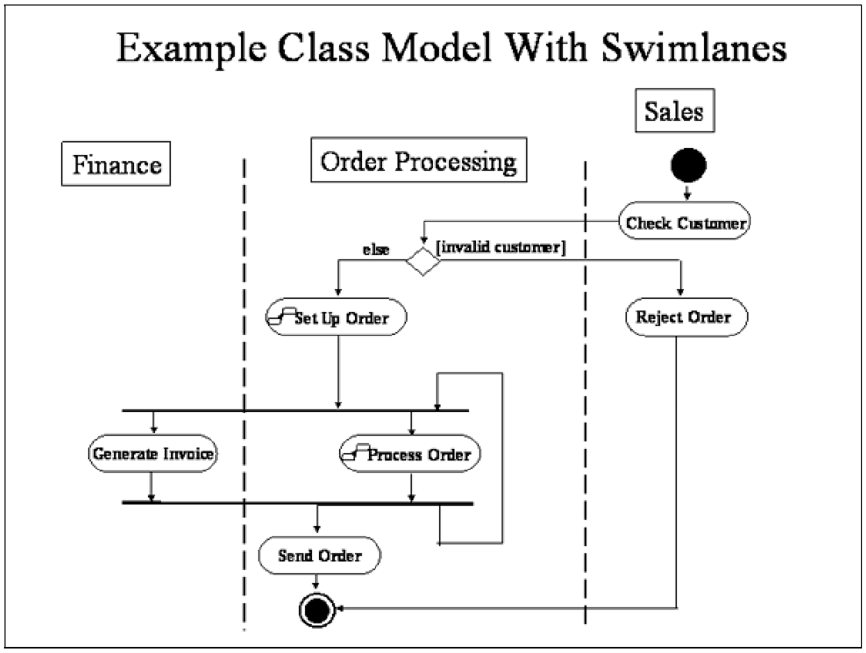
3. Add complexities appropriate to the level of the diagram. For example, if you were modelling a hire car process, the activity 'recover car' (due to breakdown) would be a complexity added at this stage.
4. Model into swimlanes if required. Figure 11 shows this for our example (and swimlanes are discussed below).
5. Review and refine, iteratively, to produce the final version. Start at the beginning and review for consistency and accuracy.

Figure 10. Order Processing Activity Diagram.



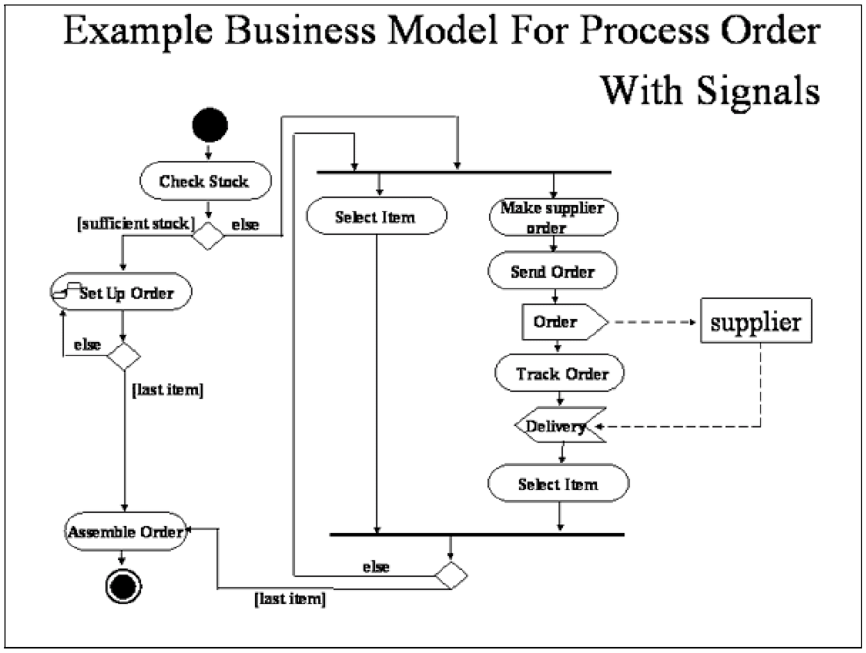
Swimlanes are an important device in business process modelling, where the responsibility of different departments or other areas of the organisation for different parts of a process is modelled. Figure 11 shows swimlanes added to our example. Notice it changes the nature of the model to a more business-focused one. Here the sales department has responsibility for customer data, as it is owned and provided by them. The order is processed by the order process department, which gathers and dispatches the order and owns the system on which the orders are kept. Finally, the finance department generates and sends the invoice. We can see that this modelling has highlighted some interesting aspects of the way this company works and the organisation of its systems. This information is useful for the design of the new system. This example shows how these diagrams can be used for analysis as well as for design.

Figure 11. Order Processing Activity Diagram with Swimlanes.



We have just seen how activity diagramming can model at the business process level, but it can also model at lower levels, closer to the process specification. Figure 12 shows the decomposition of the Process Order activity in Figure 10, to model the fact that stock may either be available to fulfil the order or not. If stock is available, then the order is set up and assembled. If insufficient stock is available, then the stock items that are available are selected, whilst those that are not are ordered from the supplier: Two new types of element are introduced here, which refer to physical objects and indicate processing off the diagram. In this case they are signals indicating that the activity sequence will wait for the supply of items to fulfil an order.

Figure 12. Lower-level Activity Diagram: Process Order.



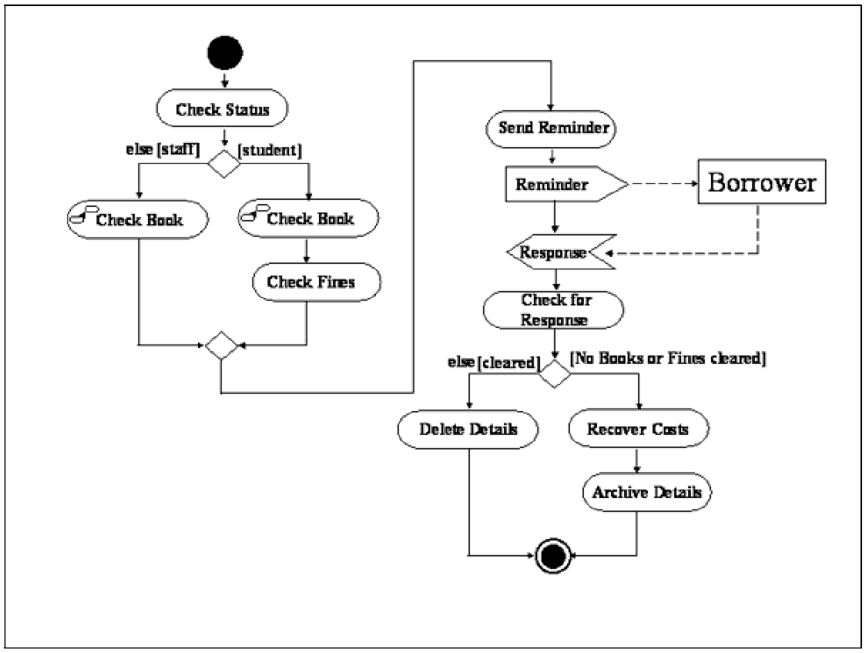
Exercise 3

Draw an activity model for the Borrower Leaving process. Compare the results obtained and the techniques used with those applied to the same problem in Exercise 1. Discuss the merits of the two methods with your colleagues.

Tutor's comments

In comparing the two methods I certainly think the activity model shows the process in a manner that is easier for users to understand, and it also handles the need to send the reminder better, but the Jackson structure has operations and models closer to the level required for program specification (see below). They both have their uses.

Activity Diagram for the Borrower Leaving Process.



Conclusion to topic

In this topic we have tried to give a flavour of the different types of dynamic process modelling techniques and how they can be used for the analysis and design of processes, from high to low levels of definition, to produce requirements and system specification.

All the techniques model the dynamics of sequence, but they model different types of process. Low-level definition of process can be achieved using Jackson structures down to the program specification level. ELH models consider the events that change the state of the data. Activity diagrams focus on the different steps in the process, and the transition from one to another, from intermediate to higher business levels. These techniques give a flavour of the range of dynamic modelling methods that are available.

Obviously the number and kind of techniques that are used will depend on the complexity and size of the system being developed. For a small system using a prototyping environment, only limited modelling may be used but some is still needed.

Section review

Based on the learning outcomes in this section you should have learnt the following:

Activity-focused design and business-based activity models.

Activity models have the following characteristics:

- They use activities to model processes, events and tasks.
- They model the transitions between activity states.
- Their conventions and diagramming methods are used across a range of disciplines from business planning to IT and are liked by users.
- They can be used at all levels from high-level business processes to lower-level functions and processes.
- They can be decomposed
- They complement static models such as data flow models at the business modelling level.

2.4.5 References

1. Eriksson, H.-E. and Penker, M., 1998 *UML Toolkit* (New York: John Wiley and Sons), ISBN 0471191612.

   Includes a good introduction to activity diagrams, although from an object-oriented point of view.

2. Jackson, M. A., 1975 The *Principles of Programme Design* (London: Academic Press), ISBN 0123790506.
3. Weaver, P. L., Lambrou, N. and Walkley, M., 1998 (2nd edn), *Practical SSADM Version 4+ – A Complete Tutorial Guide*, (London: Financial Times Management), ISBN 0273626752.

Well illustrated and includes many different techniques, as well as discussion of the essentials of analysis and design, albeit from an SSADM point of view.

2.5 Interface Analysis and Design

2.5.1 Introduction to user interface design

Section learning outcomes

After completing this section you will have knowledge of the following:

1. The scope of user interfaces.
2. The approaches to user interface design.

Introduction

Computers are used by humans and are by definition human activity systems. The development of a suitable human computer interface to the system, so its users can control and use it, is essential for the system to be successful.

This topic will focus on the development of the typical, almost universal, human computer interface, which uses a keyboard in conjunction with a mouse- (or keypad-) driven graphical user interface (GUI) and a screen to interact with a computer. It is assumed that there is a limited use of sound in this environment to indicate actions on the part of the computer. This is not the only type of interface environment that can be used, however, and we shall briefly consider some others in this introduction in order to indicate the scope of interface design. Some of these other methods are very important as they are widely used (for example, bar code, magnetic strip and embedded chip readers, and ticket printing machines) but they tend to have very focused and limited uses.

The scope of user interfaces

Besides the traditional keyboard, mouse and screen interface, there has always been an interest in other forms of interface that extend the use of computers to the wider community (including, for example, the blind and the uneducated) and different environments (for example, enabling their use while the user might not be able to use their hands to control the computer, as when driving a car or flying a fighter aircraft). This interest is broadening as technology increases the abilities of these types of interface. There is also a drive in society to reduce the **digital divide**, which divides those who do from those who do not have access to the benefits of computer-based technology, in particular the use of information. The world can be divided into those with access to the information superhighway, who are **information rich**, and those without such access, who are **information poor**. Those who are information poor tend to be the illiterate and the poverty stricken.

We shall briefly consider the following:

- speech and non-speech audio interfaces
- handwriting recognition
- haptic (gesture and touch) computing
- bar code and character readers
- biometrics

Speech and non-speech audio interfaces. We all have some experience of non-speech audio from our computers. We understand that when it beeps at us it is trying to catch our attention. This may be a simple alert that some task has been completed (even a microwave can do this), or it may be an alarm or warning sound indicating danger. (A smoke detector can do this but so too can computers if, for example, they control safety critical systems, such as those found in an aircraft or a chemical plant.)

The more pleasing sounds to come out of a computer are music and the spoken voice from CDs and DVDs. Whilst not of great use in the development of UIs for standard data processing applications, these are of great importance in the development of games and entertainment software, and they have forced the physical resources provided for computers to improve dramatically beyond those required to produce a single beep.

Finally, there are also unwanted sounds that can mask the sound the computer is generating. These are called noise, and a lot of research is conducted into how the proper structure of the sounds being generated (such as better articulation in synthesised speech) can combat any background noise, and how the noise levels accompanying generated sounds can be reduced.

Speech recognition and synthesis form an important technology and one that is developing all the time. They are not perfect and in essence require a degree of training of the system to produce viable results. Speech recognition involves a user speaking into a microphone, attached to some form of analogue to digital converter; this converts the speech into a digital signal, which can be analysed by the computer into recognisable words that may be either written to a document or taken as a command. The problems with this are that accents, background noise, the speed of speaking and variations in loudness can all cause difficulties for the software. Also, quirks of language can cause problems, such as homonyms, where a word may sound or be spelt the same but mean different things, for example, 'see' and 'sea', or saw (meaning a tool for cutting wood) and saw (meaning to have seen something). This means that speech recognition systems must be either very limited in vocabulary or trained to recognise the speech of a particular user.

Speech synthesis essentially reads aloud – or records – speech, based on digital information stored in a computer. Computer-generated speech tends to lack the intonation and quality of a human voice and even for devices like speaking clocks we tend to use stored sound recorded by a human. Speech synthesis is useful in certain circumstances, however, as for instance when a user cannot see the screen.

Handwriting recognition. It is not always viable to have a standard keyboard. They take up space and are only really effective in a size that accommodates our hands. They are also expensive in resources. Many more people can write effectively then can use a keyboard effectively. Handwriting recognition is a difficult task for a computer and requires a large amount of process power and training. To avoid this problem, some reduction in the complexity of the symbols used, and the training of the user to use a particular set of symbols, written in a particular way, can be adopted. These symbols are called **glyphs** and are used in some handheld computers.

Haptic computing. 'Haptic' is a Greek word relating to touch. A wide range of devices fall into this category, including the keyboard and the mouse. We will concentrate, however, on the more unusual devices. Personal digital assistants (PDAs) are now being developed that can have some of their functions controlled by physically moving the whole device, for example scrolling up and down through lists by tilting them backwards or forwards. This type of control is known as **tangible computing**, where the device is controlled, or the output read, by feel. A Braille output device would also fall in this category. Devices are also being developed that use the output from onboard computers in cars, which sense other cars, to activate a vibrating device in the seat-belt to warn drivers of possible collisions. There are also **force feedback devices** that try to emulate real-life situations such as flying an aircraft using a flight simulator.

The second group of haptic devices are those that sense gesture. Here the user makes gestures, which the computer captures on camera. A sign language, such as American Sign Language can be used in this way, but again there can be a lot of misinterpretation. Eye gestures can also be used to control devices, for example by military jet pilots.

Bar code and character readers. Bar code readers read a specialised code optically. The information conveyed is limited but they are universally used in shops, libraries and so on. Computers can also print out bar codes.

Character readers read and interpret characters. Originally they had to read specially designed characters but they have become more universal in their abilities. Character readers are useful, as both humans and computers can read the input and output. Character reading software can read scanned-in text pages to a high degree of accuracy nowadays, but it is not 100 per cent. They are widely used, and increasingly so with the advent of document management systems that read both incoming mail and archive material in a digitally readable form.

Biometrics. In computing terms, biometrics are measures of the physical characteristics of a person that are measurable by a computer. Typically they are characteristics that are unique to an individual, such as a fingerprint or an iris scan. They are important as they are increasingly used routinely to identify individuals.

Approaches to user interface development

User interface development follows the same basic life cycle as information systems development in general, namely, requirements gathering, analysis, design and implementation. In using this cycle, UI development has its own focus and techniques and we shall explore these. The core aspect is the concept of user-centred design, where the user is intimately involved in the design of the interface. In addition, the development of the UI is a highly iterative activity with constant testing and review of the design as it develops. This process makes significant use of prototyping, as a proper feel for what the final UI will be like is critical to good design.

In UI design there are two basic approaches. They do not differ greatly in their goals but they evolved differently and have different focuses in their approach, and different techniques. The first approach is based on dialog design and developed in the mainstream data processing environment. The second approach evolved later, in web development, and is based on information architecture principles.

The dialog design approach identifies the content of the UI based on process input and output (I/O) data. It develops sequential models for this I/O data and then transforms them into models of the required dialog. Using these dialog models, it then identifies screens and develops a menu structure. This approach is best suited to a large-database, data processing environment.

The information architecture approach identifies the content of the UI based on user needs and task analysis. It then maps content onto the data sources the system will use, including the web. Based on this mapping, it then develops the menu and navigational structures, called blueprints. From these blueprints it then identifies the necessary screens. This approach can be used for most systems, but it is mainly used for the development of information-rich and web-based systems.

We will consider both approaches in this topic, starting with the information architecture approach.

Dialog design principles

Like all areas of IS (Information System) development there are well-established generic principles (including rules of thumb) as to the way a UI should be developed and work. In UI development, more than in other areas, these principles are essential to a successful development. This is because UI development is a wide and varied activity, which brings together disparate elements and influences. A strong set of design principles around which to focus the development is essential. We review what these principles are.

Section review

Based on the learning outcomes in this section you should have learnt the following:

1. The scope of UI.

   The scope of UI is very wide ranging, from audio to tangible control of the interface by the user and the computer. In this topic, however, we will consider the typical interface development of a keyboard, mouse, screen and printer environment.

2. The approaches to UI design.

   There are two approaches. The first is through the development of dialogs and was evolved in the mainstream data processing area. The second evolved in the web development area and uses information architecture principles.

3. The need for UI design principles.

   The area of UI development encompasses a large range of possible solutions and it is important to adhere to tried and tested design principles to produce a focused and applicable product.

2.5.2 Interface development

Section learning outcomes

After completing this section you will have knowledge of the following:

1. The UI development process.
2. The associated development approaches that can be used in UI development.
3. The use of evaluation.
4. The modelling techniques used in UI analysis and design.

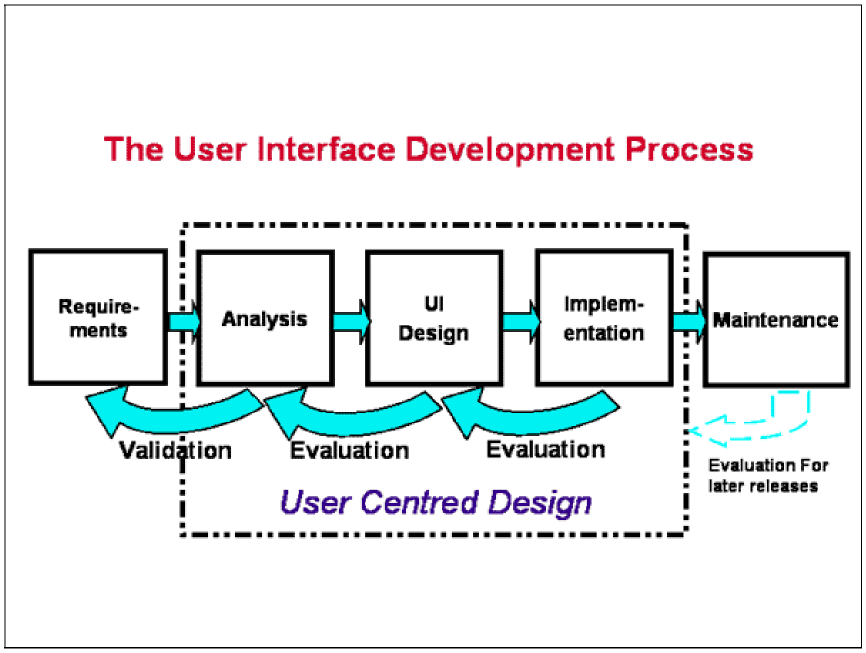
Introduction

More than anywhere else in IS development, it is essential that the users are fully integrated into the development of the UI. The UI can make or break a system and users may reject a functionally viable system if the UI does not properly facilitate their working. A poorly designed interface can negate any expected benefits due to efficiency savings. All possible users of the systems must be involved, ranging from the most casual to the regular business users, including customers, customer facing and non-facing staff, management and system maintenance staff. The user's experience may vary from being fully experienced, trained and computer literate to being a complete novice, one-time user. The systems developers therefore need to study the users, their work practices and their requirements to specify the UI properly. Any development approach taken must incorporate these factors.

UI development process

Typically the UI development process follows the standard analysis, design and implementation cycle, but with an emphasis on an iterative build-up of the design, and the final implementation, that is user centred and uses validation and evaluation to drive the process. This approach is illustrated in Figure 1, where the sequence of requirements gathering, analysis and design, implementation and finally maintenance is shown.

Figure 1. The User Interface Development Process



At each stage, iteration may improve and tune the design and production of the interface. The analysis of the requirements leads to their validation whilst the evaluation of the design may engender further analysis. Evaluation during implementation may require improvements to and extend the design. Finally, the implemented system may be evaluated during its working life, where extended usage may highlight areas for improvement in the UI, which may require changes to be made that cause further analysis, design and implementation. Every stage of the development process is user focused. Validation addresses the user's UI needs whilst evaluation is driven by the user's assessment of the UI that is being developed. The whole design process is therefore user driven.

We shall now look at the some of these stages and activities in more detail.

Requirements

In gathering requirements for UI development, the following factors need to be considered.

- The users who will use the interface:
  - their physical skills, for example, keyboarding ability
  - limiting factors , for example, disabilities
  - their cognitive skills, for example, computer literacy
- The tasks to be performed, what the user must be able to do and how they do it:
  - functional (business requirement)
  - non-functional aspects, for example, the need to use a bar code reader
  - workflow, how the tasks are performed
  - information use: how it is obtained , manipulated at the interface and used
- The usability attributes, such as learning time, control keys and mouse options
- Environment factors:
  - the physical environment, for example terminals used on engineering shop floor, screen visibility and security
  - the cultural, political and organisational environment, for example, multiple language interface, specific security issues

Requirements are gathered from a range of sources using a range of techniques. Obviously some will be in the user's statement of requirements but others will be dependent on observation and close study of the user's working environment and practices. These will include user observation, interviews, questionnaires, expert review and system monitoring. In the area of UI requirements validation, the use of suitable analysis techniques is particularly important, as the user requirements are typically amplified considerably by developers to ensure that they capture the true nature of the user's working practices, needs and expectations. In UI terms, requirements gathering based on these factors is called **user needs analysis**.

Pause for Thought

Use the aspects of requirements gathering described in this section to discover/identify the UI requirements in a work environment in which you have worked.

Tutor's comments

In working for a company that built computing systems I worked in an office with other consultants. The basic UI requirements were:

- The users who will use the interface:
-   - physical skills: **All staff could use keyboards and mice.**

    - limiting factors: **One consultant who suffered from repetitive strain injury could not use a keyboard for long periods, and was provided with speech capture and recognition facilities for word processing.**
    - their cognitive skills: **All were computer literate.**
- The tasks to be performed, what the user must be able to do:
    - functional (business requirement): **Produce reports and other documentation for clients with a need to gather information both from company databases and from the web**.
    - non-functional aspects: **Standard and specialist printers, scanning equipment and integrated fax facilities were required**.
- The usability attributes: **Sensible response times were needed. (Often the local network was slow.)**
- Environment factors:
    - the physical environment: **Standard office environment but fairly cramped and with main walkway through it. Also, users dealt with commercially sensitive information and machines should automatically go to standby when not in use, with password required to reactivate them.**
    - the cultural, political and organisational environment: **Staff prefer the office to be quiet so they can concentrate; they prefer quiet printers**.

Analysis

The goal of UI analysis is to discover how the users work and how they need to use the new system to achieve the functionality they require. The main focus points of this analysis are:

- the user as a person; who and what they do
- the situation; where, in an office, in the field
- the intensity of the work for which the system will be used; its simplicity or complexity and the volume of work
- the type of control needed; mouse/function keys/touch screen
- information flows; what data, information and knowledge are required and how are they obtained
- workflow; the order in which tasks are undertaken and associated and the use of the computer to support this
- computer dialog; the interaction the user expects in order to carry out their tasks
- IT presentation norms; GUI, levels of navigation

There are a range of techniques which can be used to gather and analyse information about the system and the business. The three main areas which require detailed analysis are information flow, workflow and computer dialog. With this is mind we can categorise the techniques as follows:

- User needs analysis, comprising:
    - user characterisation
    - user task identification
    - situational analysis
- Review of existing interfaces:
    - monitor use
    - review against standard UI design principles (discussed in Section 3, on ‘Design Principles’,)
- Development of information architecture (discussed in Section 4)
- Task analysis:
    - workflow analysis using such techniques as activity modelling (see Topic 4 Section 4, where activity modelling is discussed)
    - task monitoring
- Dialog analysis, including the order in which system data is enquired against and updated. (This technique is discussed in Section 5.)

User needs analysis, as with requirements gathering, will use observation, interviews, questionnaires, expert review and system monitoring, as well extensive reading of user documentation, including works manuals and system guides, job descriptions and standards manuals.

Specialised modelling techniques will also be used to analyse the UI needs for information gathering and presentation, computer dialog (including menu structures and navigation), workflow and user processes. Many of the models developed will cross-reference and be interconnected. The access to, and update and presentation of information is linked to workflow, as are also the menu structures and navigation. There will also be further cross-references to the main system development specification. For example, information gathering will refer to the main data model and enquiry access path models (see Topics 3 and 4, respectively).

Information architecture

Information Architecture is the analysis and design of the information flows throughout the system so that the user can access information easily. It maps the information required in the interface to how it is held in the system. It develops the following deliverables:

- Mappings show the connections between the required information and the underlying data repositories.
- Blueprints are developed, which show the users’ information categories and form the basis of navigation.

- Wireframes are developed, which illustrate the screens, the information displayed and the design to facilitate workflows.

Obviously such emphasis on information gathering, and the design of the screens to facilitate its use, has become more important and critical with the rise of networked computing, which enables data to be accessed from disparate sources and locations in large quantities. The proper integration and presentation of the information is essential. Information architecture is a new discipline as the gathering of information is becoming a complex activity that requires specialist skills and knowledge. Its forte is the design of systems that use complex search criteria followed by the sophisticated analysis and categorisation of the results.

In analysis, information architecture techniques analyse the way users use information, the sources from which they access it, the criteria they use to search for information and the way they seek to present and navigate the results.

In design, information architecture produces the specification for information mapping to data sources, and the structuring of information in the interface and on the screen to reflect the user's use of the data.

Background reading in Information Architecture is cited in Section 6 of this topic.

Design

The goal of UI design is to create a system that enables the users to carry out their tasks in the following way:

- safely, especially for safety critical systems
- effectively, allowing full use of the system's functionality and achieving the correct result
- efficiently, requiring the expense of the minimum effort to utilise the interface
- enjoyably, or at least with minimum frustration!

The measurement of these attributes allows the usability of the system to be assessed.

This goal is achieved by adopting proper design criteria and principles (discussed in Section 3), supported by a range of suitable design techniques.

Various design approaches and techniques will be used including:

- participative design (perhaps using Joint Application Development (JAD))
- interviews and questionnaires
- workshops to brainstorm general design and project interface standards, or to design critical elements of the UI
- prototyping (perhaps using Rapid Application Development (RAD))
    - storyboarding, using pen and paper mock-ups
    - software prototypes, for example, using Visual Basic
- working trials (this could be treated as a form of prototyping)
    - small scale test system for a sample of users
- advice from experts (such as psychologists, graphic designers)
- design modelling techniques (discussed in Sections 4 and 5)
    - dialog design
    - input and output diagrams
    - menu design
    - blueprints
    - wireframes
    - content mapping
- design reviews and evaluation

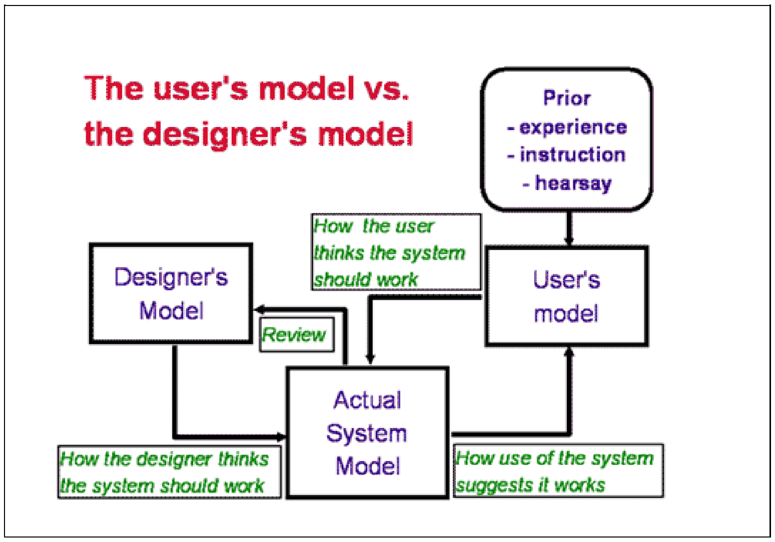
There are many tools available to assist in UI design and development.

- The event-driven 'Visual ...' development environments (such as Visual Basic): good for both dialog and screen design
- '4GLs' – Prototyping application generators (good for data-oriented applications)
- Web page editors (for example, *Hot Metal* , *FrontPage*)
- Hypermedia authoring tools (for example, Adobe Authorware)

These are very effective and easy to use, and they provide good migration to the 'actual' system, but care needs to be taken that they are utilised in a way that produces an effective specification as well as the interface.

These techniques will be used to elicit a specification that is an amalgamation of the designer and user views of the actual system. This process is shown in Figure 2.

Figure 2. The Development of the UI Specification.


The user's model vs. the designer's model
Prior
- experience
- instruction
- hearsay
How the user thinks the system should work
Designer's Model
User's model
Review
Actual System Model
How the designer thinks the system should work
How use of the system suggests it works

The user has one view of the system while the designer will have another. The user’s view is based on experience of the existing system (how it works) and knowledge of what they wish to achieve (how it should work). The designer’s view, on the other hand, is based on a review of the the system (how it works) and their experience and understanding of design principles (how it should work). Ideally the two views (user’s and designer’s models) should be evaluated against each other and a combined view developed that approximates as closely as possible to the actual (true) system model. They develop this actual system model iteratively.

JAD and RAD

These are systems development methods in their own right, but for large-scale systems they can be used for particular aspects of the development, such as UI development. They can also be used in combination.

JAD, or **joint application development**, is particularly appropriate for user-centred design as by definition it uses development teams made up of both developers and users. The users are not just present in the team to provide expert advice in the system domain but are active members of the team, producing project deliverables and being trained in the tools and methods used in the development. In projects implementing off-the-shelf packages, users may well be the main staff used to parameterise and configure the package.

RAD, or **rapid application development**, typically uses prototyping of the system itself, or critical aspects of it, to allow quick, iterative development of the system. Central to such prototyping is the development of the UI. Usually UI development tools are used and they allow quick changes to be made to the prototype UI as the design develops. RAD methods use techniques such as **time-boxing** (where short periods of elapsed time are allocated to the implementation of specific functionality) in order to limit the time allowed for development and force the implementation of only the essential requirements. In UI development, this ensures that users obtain an early view of the expected UI, allowing adequate time for evaluation and further iterations of the design.

Implementation

The completion of the design is not the end of the story as regards the design of the UI, as implementation may introduce new elements that need to be evaluated; while there may also be issues with navigability and controllability that only come to light at this stage. Implementation may be the first time that the users actually see how the UI will look and feel. Prototypes and/or extensive test systems may be set up to demonstrate the response and performance of the UI and the system. With every delivery of a UI element, during implementation, some form of evaluation must be undertaken.

Evaluation

Evaluation is critical to the success of UI design and implementation. Evaluation can be used to assess both the system being developed and the legacy system being replaced. The techniques include those used in requirements capture and some used in design, and they include:

- user observation
- user interview
- questionnaires
- expert review
- system monitoring (using monitoring tools to record usage statistics and the sequence in which tasks are progressed, as well as success rates in searches*)
- workshops (a quick way of canvassing a range of views and agree ways forward)
- formal review and testing (against the design principles and requirements leading to sign-off)

[* Two search success rate measures are the Precision Ratio and the Recall Ratio.

$$\text{Precision Ratio} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}}$$

$$\text{Recall Ratio} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents in the system}}$$

The recall ratio may be a little difficult to measure for web searches but it could be estimated for an in-house system.]

The techniques listed above reflect that not only the UI itself but also the users' use of it must be evaluated. Both user and development staff must be involved in the evaluation.

Evaluation can be an expensive activity, and the methods and techniques used must be judiciously selected to optimise cost, completeness and benefit. Even on the smallest of developments some evaluation must be undertaken, perhaps simply a review of the interface by the user with the developer.

Some form of evaluation should be undertaken once the system has been implemented, as part of a post-project review to ensure that any discovered issues are addressed.

Section review

Based on the learning outcomes in this section you should have learnt the following:

1. **The UI development process.**

The process follows the normal cycle of requirements: capture, analysis, design and implementation, followed by maintenance. It is, however, highly iterative, allowing for changes to the UI design to be rapidly assimilated and permitting numerous opportunities for improvement. This is important, as the proper development of the UI is critical to the success of any system.

The focus of the development cycle is on user-centred design, ensuring that the user is at the centre of the process. Throughout requirements gathering and analysis, the development process concentrates on the user, user tasks and the situation. Within analysis and design, user needs, information flows, workflow and dialog design are the most important aspects identified and modelled. In modelling and specifying the system, the reconciliation of the designer and user view of the system is essential.

2. **The associated development approaches that can be used in UI development.**

There are three associated development approaches that can be used in developing the UI: information architecture development, JAD and RAD. Information architecture allows for the proper gathering of the data, information and knowledge that the user requires. It maps their view of the data through the interface to the underlying data repositories. JAD enables the proper integration of the user into the development teams. RAD encourages the use of prototyping and provides an environment that supports the highly iterative nature of UI development.

The design should ensure a UI that can be used safely, effectively, efficiently and without frustration.

3. **The use of evaluation**

Evaluation, which is the assessment of the effectiveness of a UI, is crucial to successful UI development. It requires the review of both the interface *and* users' use of and response to it. This can be achieved by a combination of techniques that monitor the users using the interface, canvas their opinions of it and assess it against the design criteria and requirements.

4. **The modelling techniques and tools used in UI analysis and design**

The following techniques have been identified as being used in UI development:

- user observation
- user interviews
- questionnaires
- expert review
- system monitoring (using monitoring tools to record usage statistics and the sequence in which tasks are progressed)
- workshops (a quick way of canvassing a range of views and agreeing ways forward)
- formal review and testing (against the design principles and requirements leading to sign-off)
- participative design
- prototyping
- working trials

- design modelling techniques:
    - dialog design
    - input and output diagrams
    - menu design
    - blueprints
    - wireframes
    - content mapping
- design reviews and evaluation.

The following type of tools have been identified.

- The event-driven 'Visual...' development environments
- '4GLs'
- Web page editors
- Hypermedia authoring tools

2.5.3 User interface design principles

Section learning outcomes

After completing this section you will have knowledge of the following:

The design principles that underlie good practice in UI development.

Introduction

The underlying objective of UI design is to provide an interface that enables the user to use the functionality offered by the system or application effectively. This requires two elements to be present. Firstly, the system must ensure that all the functionality we require is available through the interface in a way that enables us to operate it and, secondly, and just as importantly, it must allow us to be what we are – that is human. We shall address the second element first: the fact that the UI is a human computer interface (HCI).

Human capacities

As human beings we have the following capacities, and a well-designed UI should allow for them:

- To make mistakes: a good UI should allow us to recover from errors in data entry or the use of the interface.
- To forget things: a good UI should have help facilities which can be called up or which prompt automatically, with the screen laid out to accommodate the task being undertaken.
- A limit to what we can achieve all at once: the interface should provide an opportunity to store work at or near the point at which we wish to stop working.
- Different physical and mental characteristics: the interface should accommodate a range of capabilities; the UI should allow for experienced and inexperienced, highly capable and less capable users.
- We learn from, and build our skills through, experience:
    - We build our computer skills and literacy on experience and familiarity. A UI should build on this familiarity and utilise or extend the generic interface mechanism (including the introduction of a GUI built on methods already present, such as menus).
    - We learn and build skills in the functionality a system offers. The UI should allow the user to use the basic functionality of a system before extending their usage to include more esoteric elements.
- Users typically prefer the easy-to-do for everyday work. This does not mean that their work tasks need be easy but rather that the tools they use to undertake them are easy to use; the UI should facilitate use, workflow and task completion.

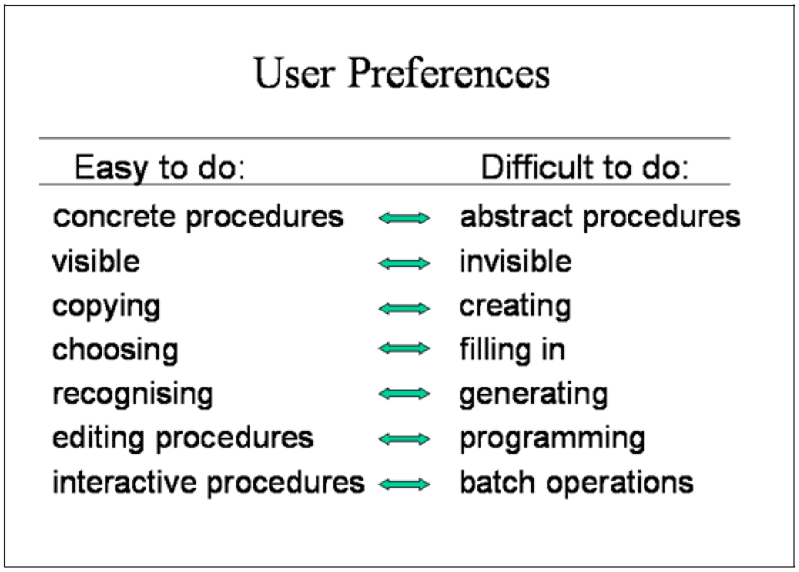
If we think how we learnt to use a word processor, it would have been very difficult for us if our mistakes were not recoverable. I remember when the deletion of text could not always be recovered. I only intermittently use spreadsheets and so need to be able to catch up with my skills when I do, using the help function as necessary. I write long documents and frequently have to save my work. When I started using a word processor I did little more than text edit but, as time went on, I learnt to develop templates and macros. My familiarity with GUI allows me to use a wide range of applications easily which have common interface commands, to print and save files for instance. Finally, the frustration experienced when we are trying to solve a formatting problem, and find we cannot, when we need to finish a report or a letter quickly, is known to all.

From these capacities, we can define a set of principles that make the system's functionality properly available through the interface. These principles concern the user preferences in working and the design of the screens and navigation between them.

User preferences

When first developing GUI-based UI, the developers explored what users of UI found easy or difficult to do. The findings are shown in Figure 3.

Figure 3. User Preferences.



The easy and difficult things to do tended to be at the opposing ends of a continuum (indicated by the double-ended arrows in Figure 3). It was found that users preferred to have concrete, structured procedures to follow rather than grapple with abstract ones that required deliberation and decisions in order to progress. They also preferred the use of interactive procedures that give immediate feedback (for example, the success or failure of a command). They find having the basic commands and facilities readily available, and visible, preferable as it facilitates use and navigation and can act as an aide-memoire. The ability to use existing material as a basis for current work is preferable to creating new work from scratch, and any UI must provide means of identifying, recovering and copying material, as appropriate. In a similar vein, programmers prefer to edit code rather than to generate it from scratch. Users prefer choosing options from a menu rather to filling in a field from scratch, and it has been shown that psychologically we prefer to recognise items from a set of objects rather than to attempt to generate them.

From a consideration of these perspectives a set of design principles was developed:

- Consistent use of a conceptual model which is familiar to the user, for example, workplace metaphors
- Seeing and pointing rather than having to remember and type
- WYSIWYG
- Use of universal commands
- Task analysis
- Use of modelling and prototyping to establish design

We have already visited the idea that users prefer to be familiar with what they are doing and that one of the ways of doing this is to use verbal and visual metaphors. This has consistently been the case since the early days of computing, with the concept of files and records emulating manual practices. In the word processor I am using there is an image of a folder (another reference to manual office procedures) on a tab, which brings up a window (another metaphor) if I click on it, showing the files I have in the folder I am currently using.

It is easier to point than to type, but this tends to suggest that GUI is the only possible type of interface that is viable. This is not true as other forms can be and are used, for instance, command language interfaces where instructions are typed in. The DOS operating system underlying Windows (and still accessible through it) is an example of such an interface. Other interfaces may be dependent on natural language, menu selection or form filling. As is often the case, applications can be controlled by a mixture of these interface styles. For instance, commercially available packages such as booking systems for hotels have been through many generations, from command language through the use of function keys to a full GUI implementation. They still allow users to opt for previous interface styles where necessary. The real principle here is that pointing is best for most purposes, but other forms of control should not be totally excluded when they are appropriate. Typically we control the Windows operating system parameters using the GUI but we will use command language in fields where necessary.

What you see is what you get (WYSIWYG) is a form of editing and text presentation, which we now take for granted, but previously such text showed all its embedded format commands on the screen. It is very useful to able to view what it is you are actually producing.

The use of universal commands is of course beneficial in creating familiarity with the computing environment among users.

In focusing on task analysis, the core of good UI design is emphasised. The UI for an application must be designed to facilitate the task undertaken by it. Similarly, by highlighting prototyping as an approach, the need for an iterative, user-centred development is also emphasised.

Dialog design

We now start to look at the first element outlined in the Introduction to this section: that the system must ensure that all the functionality we require is available through the interface in such a way that we can operate it.

There are three main aspects to this: dialog design, screen design and user guidance (help). We will first consider dialog design. This dialog is the interaction of the user with the system through the interface. Its purpose is to develop an efficient means of commanding the system. Its main principles are:

- To strive for consistency so that the dialogue appears uniform within the application of system; the order of processing tasks is important here.
- To avoid surprises and the introduction of unfamiliar mechanisms.
- To enable frequent and skilled users to use short-cuts.

- To provide informative feedback so that the user understands what has and has not taken place.
- To yield closure so that users can complete a task.
- To provide error support and allow simple error handling so that users can rectify problems.
- To allow for recovery and the reversal of actions, so that errors can be rectified and alternatives and new routes through the dialog may be attempted experimentally.
- To highlight errors as they are discovered but to allow the user to cancel the alerts.
- To reduce short-term memory load.
- To use simple and natural dialog that flows in a logical way that is easily apparent to the user.
- To use the user's language and concepts to use and drive the dialog.
- To use modelling and prototyping to establish the design.

Implicit in dialog design is the structure of the menus that are available to navigate the functionality of the system, and it is good practice to ensure that navigation between screens is well indicated, and that it follows the tasks supported.

Good error management is essential, and it must go beyond the simple message that says that an error has occurred and which is then accompanied by impenetrable code. The user should be supported and error management should seek to prevent, protect and inform. Prevention can be wide ranging, from clear, on-screen instructions and dialog, through well-designed icons and screens, to the articulation and operation of the UI following the user's work practices for particular tasks. Also, where it is known the UI may be unfamiliar to the user, extra guidance may be given. The user text entry may even be spell-checked, particularly if entries are to be used to search data. (An example of this is search engines confirming your spelling of a search word when it appears to be a misspelling of a word with a lot of hits.) In other words, prevention seeks to provide the user with little opportunity to make errors. Protecting the user from errors focuses on mechanisms that allow errors to be reversed, and this usually centres on user actions and updates being saved so they can be reversed. It is also important to provide clear exit points, where the user can leave the activity without changing the way things were. Finally, in making a mistake, the user may want to know (be informed) what they have done and how to avoid the error in the future. For non-trivial errors, clear and non-judgemental messages should occur explaining what can be done, and even which section of the help support material to read or how to contact technical support.

Screen design

Screen design focuses on the presentational aspects of the interface and the visibility it must provide of the functions it supports. The basic design principles are:

- To establish what the user requires from the screen.
- To display the minimum information needed for the task concerned. This is essentially a matter of not cluttering the screen. (Hence the option to add or remove menus from the word processor screen.)
- To highlight changes to enforce feedback.
- To use colour sparingly (no more than five), but also use it to emphasise, or collect together, different themes that may be on the screen.
- To use icons that are efficient and language independent, but to ensure that their meaning is understood and that good affordance, visibility and feedback are built in.
- To use metaphors that are consistent and complete.
- To use modelling and prototyping to establish design.

It is important to remember that when we discuss screen design we are typically talking about something that is multifunctional. The screen I am using at the moment has essentially four areas devoted to different aspects. Firstly, it has header and footer sections which contain links to the menus that control the word processor and also direct commands. Secondly, on the left-hand side is an area showing the formatting of the document I am writing. Thirdly, there is the main area of the screen where the document is produced and the text is edited. Fourthly, there is an area at the base of the screen which allows connection to the operating system, other applications and word processor files, and a clock. Consequently, some elements remain across screens and others change.

The terms affordance, visibility and feedback have specific meaning when applied to icons (widgets) on the screen. *Affordance* is how the icon informs the user how it should be used. For example, if the icon represents a button to be pushed it should look like a button. In Windows, at the bottom of the start menu, there is a red button to turn off the computer which is shaded around the edges to suggest a button to be pushed. The function that this button icon supports has visibility in two ways. Firstly, it has a standard symbol (a circle with a vertical bar in it) that denotes the switching on or off of an electrical device. Secondly, when the cursor is held over it, a small message window opens saying what the icon does. When the icon is activated by clicking the mouse on it, a message appears requesting confirmation of the action. This is feedback on the initial action informing the user that it has been acted on.

Guidance

Users may require guidance in operating the UI and some form of guidance is required. The guidance may be initiated by the system or the user and provided off-line or online.

Online guidance may consist of:

- system initiated:
  - contextual cues (permanent or pop-up)
  - warning messages
  - error messages (which require attention)
  - offers of the Help function
- user initiated:
  - contextual help
  - reference help

    - online tutorial
    - technical helpdesk networked contact

Offline guidance may consist of:

- novice user:
    - user guides (how to:)
    - training courses
    - user support helpdesk
- experienced user:
    - quick reference card
    - reference documents
    - user support helpdesk
- system administrator:
    - administration reference
    - technical helpdesk
- maintenance engineer:
    - technical reference
    - technical helpdesk

How much help functionality is required depends very much on the complexities of the tasks and the expected experience of the users, their training and supervision. For novice users, more contextual, dynamic help functionality is required. Experienced users will want to be able to switch off these features and have the option to access detailed help (for example, online manuals) when they choose.

Section review

Based on the learning outcomes in this section you should have learnt the following:

The design principles that underlie good practice in UI development.

The design principles are characterised as by:

- User capacities: Users are human and as such can commit errors, forget, have a range of abilities and work effectively only for a certain period of time.
- User working preferences: Users like to work interactively, using well defined methods and utilising previous work that can be edited.
- Dialog design: The interface should be as simple as possible but also have the ability to articulate the functionality available. It should allow for errors and subsequent recovery. It should be designed and built iteratively using prototyping.
- Screen design: Screens must be simple to use and must present their concepts consistently, using clear metaphors and icons that provide good affordance, visibility and feedback.
- Guidance: This is provided to assist the user in using the system. It may be initiated by the system or the user and either online or off-line using guides, manuals, training and support.

2.5.4 User interface modelling - information architecture approach

Section learning outcomes

After completing this section you will have knowledge of the following:

1. Content mapping including blueprinting, metadata and controlled vocabularies.
2. Screen design and wireframes.

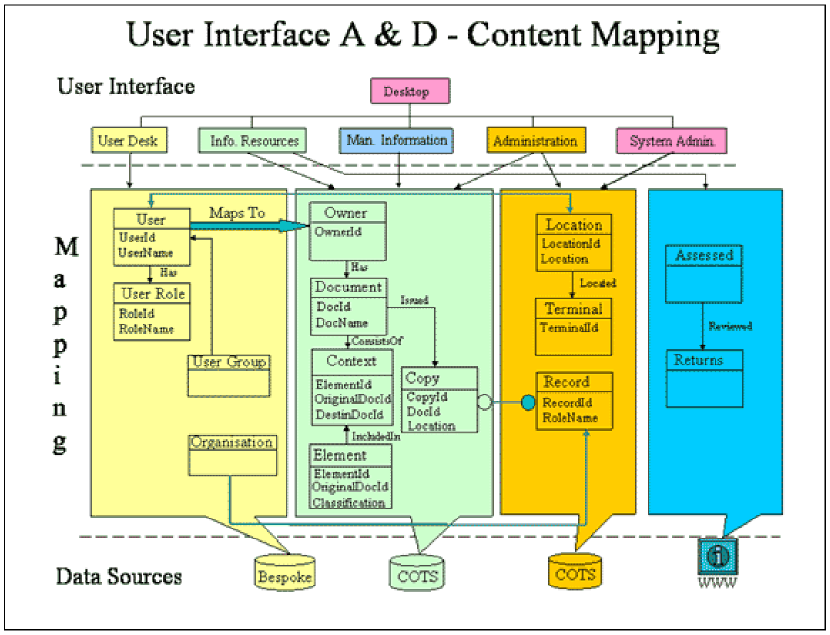
Content mapping and Information Architecture (IA)

Content mapping is the relating of information held in the system to the UI and the way it is to be presented there. It is important to separate the content from the container, because the way the information will be held and structured in the source databases may not be the same as that required for presentation in the UI, where disparate data may be associated in a single screen. The amount of processing required will be dependent on the nature of the system. If it is a single relational database, then basic SQL enquires may suffice, although the need to map to the UI may introduce new queries. If it is a large system, using a range of different databases containing many different types of object (for example, text data, images, documents), as well as searches on the web, then considerable and sophisticated software and processing will be necessary. Either way, the content needs some form of mapping and typically some kind of super schema, will be needed to map the content of the data sources to the UI. We use the term 'schema' loosely because information that is mapped from differing sources to a screen in the UI need not necessarily be combined but could be displayed side by side. If the information is to be combined, then some form of pre-processing will also be required.

To enable us to access content, search mechanisms must be in place. These fall into two categories. Firstly, the menu structures in the UI enable the user to focus in on specific content. Secondly, search strings may be key or other data that are entered into a field and used to access data in a database, or generic terms that are directed at a search engine. The modelling technique of **blueprinting** addresses the first, as it models the navigational structure of the UI in terms of functionality and content. The second may use techniques such as **controlled vocabularies** and **metadata** to ensure that accurate terms are used in the search. The last two techniques are implemented for the user to use and need to be designed. We shall look more at these later in this section.

Figure 4 shows a content mapping of data from various databases, and from the web, to the UI for a system for the control of documents within a company. It is incomplete and schematic as its purpose is to illustrate the method. It can be treated as a working document for developing the full content mapping.

Figure 4. Content Mapping – A Working Document.



At the bottom are the data sources, which in this case are a mixture of bespoke and commercial off-the-shelf (COTS) databases (the document and asset management systems) and the web. In the middle, the mapping is shown in terms of extracts from the data models of the sources (shown in black) and any new interconnections between them required for the UI (shown in green). These mappings are then linked to the areas of the UI that they support, shown at the top of the diagram.

There is an obvious method here to develop these mappings.

1. Determine user contents needs. These should be available from the requirements, user needs analysis and user task analysis. In particular, the various associations of data required to complete the user tasks must be established.
2. The content of the data sources needed to produce the UI content must be defined. Data models should already exist, but this is not inevitable, and some may need to be drawn. There may be no definable content, for instance if the web is to be searched. In this case the data is gathered and then processed to discover which returns are relevant, whether against predefined criteria or those entered interactively by the user. The data from the relevant returns may then be processed further to extract data in a form relevant to the user. This process is shown in Figure 4 as the relations Returns and Assessed (meaning processed returns).
3. The two contents are mapped to each other and any further structuring of the data, either within or between sources, is undertaken.
4. A mapping schema is produced and any necessary metadata and controlled vocabularies defined.
5. Data extraction and mapping processes are identified and described.

In deciding which content is required, we need to know how it needs to be chunked. For example, the users of our document control system may wish to abstract elements from many documents (based on a search string), review them one element at a time, and use those selected in a new document. At what level would they wish to extract these elements from a document that contained the search string? A sentence, a paragraph, or a page? If they chose a paragraph, then that would become our contents chunk: having entered the search string they could then receive a set of paragraphs displayed one at a time. For the system administrator who wanted to know who had which computer, the contents chunk would be the resultant table. Our contents mapping must reflect the chunking.

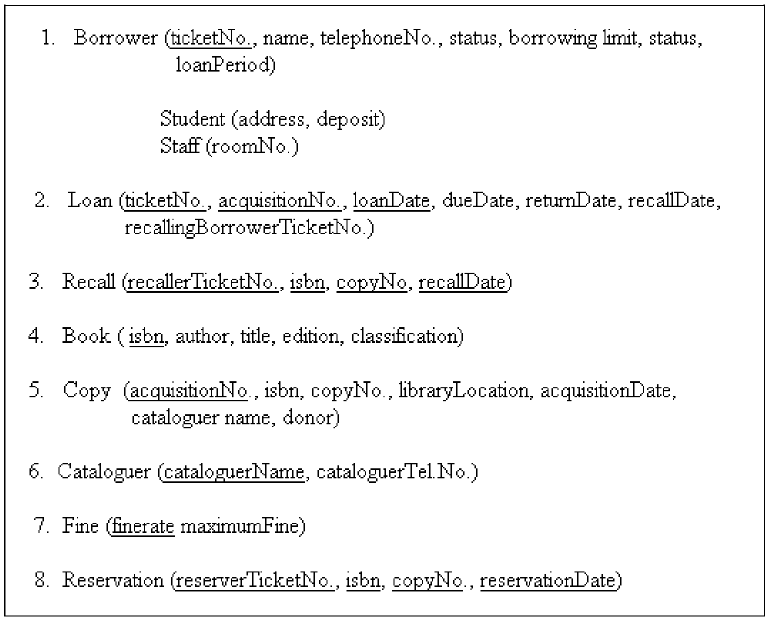
Content mapping may seem a little remote from the UI, but presenting the contents of the system to users, in the way they want it, and structured so that they can easily navigate it, is essential to the successful implementation of the interface.

Exercise 1

For this exercise you need to use Tables E1 and E2 below. Table E1 contains the entity headers for the new Porterhouse College Library System. Table E2 contains a user task description of how a librarian will use the new system to loan a book.

Using the entity headers to define the data source for the new library system, and the user task description to define the user needs, you will produce a contents mapping for the loan book contents of the UI.

Table E1. Entity Headers in the Library System Case Study



1. Borrower (ticketNo., name, telephoneNo., status, borrowing limit, status, loanPeriod)

   Student (address, deposit)
   Staff (roomNo.)

2. Loan (ticketNo., acquisitionNo., loanDate, dueDate, returnDate, recallDate, recallingBorrowerTicketNo.)

3. Recall (recallerTicketNo., isbn, copyNo, recallDate)

4. Book (isbn, author, title, edition, classification)

5. Copy (acquisitionNo., isbn, copyNo., libraryLocation, acquisitionDate, cataloguer name, donor)

6. Cataloguer (cataloguerName, cataloguerTel.No.)

7. Fine (finerate maximumFine)

8. Reservation (reserverTicketNo., isbn, copyNo., reservationDate)

Table E2. Task Description for Normal Book Loan in Library System Case Study.

When the borrower brings the book for loan to the loans desk, the librarian will enter the borrower's ticket number, using a bar code reader to read the code on the borrower's library card. The system will check to see if the borrower has reached their loan limit (20 books if a student), exceeded the outstanding fines allowance (£20) or is trying to borrow a book outside term time (if a student). If they do meet any of these criteria, then the loan is rejected and a reason output. If the borrower has reached the loan limit they may request the librarian to print them out which loans are outstanding. (The borrower name, the title and the loan date are printed out for each loan.) If the loan is accepted, then a bar code reader is used to read the acquisition number from the book. The date of the loan is recorded.

To simplify this question, do not attempt to draw a diagram to show the mapping but use a table to map the data content in the UI to the data content in the sources. Think about any processing that might be needed.

Tutor's comments

First we can consider what UI contents are put there by the user. Initially, the user enters the ticket number of the borrower to whom the user is making the loan. If the loan is unsuccessful, then the user may enter further commands, for instance to print the list of borrower loans. If the loan of the book is successful, then the user will enter the acquisition number of the copy being loaned. The date will be system generated for inclusion in the loan data.

Now we need to consider which data is needed by the user at the UI. We may assume that in order to support the borrower the librarian will expect details of any outstanding loans, and the value of any unpaid fines, to be available at the interface. The use of this data is for information. The actual check on the borrower is a process and the data for this need not necessarily be placed in the interface, only the result. Other data which might be useful is the borrower's name for identification purposes.

In summary the information expected is:

- Loan information for the loan being attempted; ticket number, acquisition number, current date.
- Loan information of the borrower's existing loans: ticket number, acquisition number, loan date, title.
- Borrower details: borrower name, total value of outstanding fines.
- Results: if number of outstanding loans equals 20, if the total of outstanding fines is equal or greater than £20.
- Dialogue elements: rejection messages.

Initial Mapping to Data SourcesTaking the data identified in the contents, is data available to provide contents and store entered contents? Table E3 shows this initial mapping.

Table E3. Initial Mapping.

UI Contents Data	Source Data	Comments
Data for loan (input)ticket numberacquisition numbercurrent date	Loan entity:ticketNoacquisitionNo	Current date is system generated.
Borrower data (output)borrower name	Borrower entity:name	
Unpaid fines (output)	Loan entity:ticketNodue dateFines entity:finerate	Unpaid fines needs to be derived.
	Loan	The outstanding loans can be output and the total number of

Outstanding loans(output)	entity:ticketNotitledateBorrower entity:name	outstanding loans needs to be derived.
Rejection messages		Not defined in data sources.

Fuller Mapping

We can now review the results and decide which processes are needed to map fully the data from the sources to the contents of the UI and write the data from the UI to the sources.

Table E4. Fuller Mapping.

UI Contents Data	Source Data	Processing	Comments
Data for loan (input)ticket numberacquisition numbercurrent date	Loan entity:ticketNoacquisitionNo	Direct mapping. Current date mapped from system	Current date is system generated.
Borrower data (output)borrower name	Borrower entity:name	Direct mapping	
Unpaid fines (output)	Loan entity:ticketNodue dateFines entity:finerate	Requires the number of days overdue to be determined from the Loan entity, for outstanding loans using the current date and the due date. Number of days overdue times the fine rate will give the total outstanding fines. All other data is directly mappable.	This calculation only works if the fines are only due on outstanding loans. If any other fines are due then no record of them is held, or derivable. Recommend that the total outstanding fines is held as a running total updated daily.
Outstanding loans(output)	Loan entity:ticketNotitledateBorrower entity:name	The total number of loans can be derived by counting them.All other data is directly mappable.	The outstanding loans can be output and the total number of outstanding loans calculated.
Rejection messages			Not defined in data sources.

We can see from this exercise that the mapping of outstanding fines could be an issue and needs to be reviewed. Otherwise, the other UI content for a book loan can be mapped to the data sources.

Metadata

Metadata is defined as data about data, and it forms the core of any schema definition, but in information architecture its use goes much further and it has a multifunctional role. Three major types of metadata occur:

- Descriptive: data about the nature of the thing the data is describing. For example, if the data is a picture for a clothing catalogue, this type of metadata would describe the contents of the picture so that it could be accessed by a variety of criteria, for instance garment colour, gender, season, size, type and number of items modelled in the picture, and its location. This metadata goes beyond that held in the garment database or the catalogue database and is dependent on the way content is to be used and by whom (for example, the catalogue designer).
- Intrinsic: data about the thing itself. For example, what file type is the garment picture held as, and what is its size.
- Administrative: data about how the thing is handled. For example, who owns the picture? And is it cleared for publication?

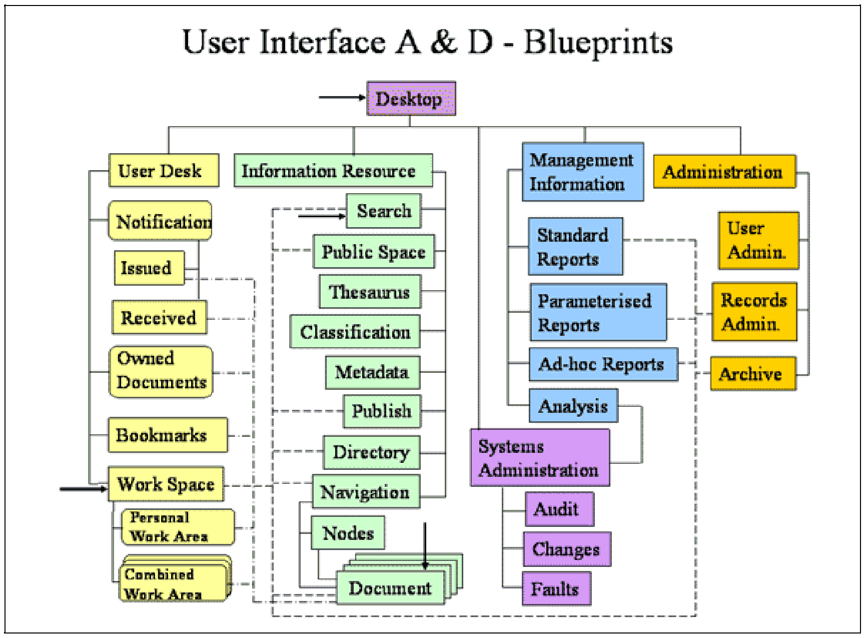
The core of content mapping is the content required by the users, which can vary widely. In the system described in Figure 4, the user may want to look at particular documents depending on their content, whilst systems administration may require a list of the location of all computers. The first requires particular metadata but the latter is a simple database enquiry. In the case of this particular system, where a document management system is in place, the metadata for selecting particular documents is available. In the case of the assessed material from the web, custom metadata may have to be applied to it, for its correct assignment to particular content to be established.

Blueprints

Our content mapping shown in Figure 4 maps only to the highest level of the UI screens (as shown at the top of the diagram), but obviously detailed mapping to the lower levels is also required, and this level of design begins with the development of blueprints.

Blueprints show the relationship between screens and other contents components. They essentially map the application area and provide a view of the information space. A high-level blueprint for our document control system is shown in Figure 5. Again, this is a working document and not necessarily complete.

Figure 5. Blueprint.



This is a high-level blueprint that shows the higher levels of navigation that are available through screens. Blueprints, like many other task- and process-oriented models, can be decomposed and lower-level blueprints developed. Quite often these become more task and navigation focused and contain directional arrows indicating sequence.

I have chosen to include both organisational and task-oriented elements in this blueprint. How the blueprint is structured will depend on a variety of factors including how the organisation views itself and is structured and the function of the system.

In Figure 5, the arrows pointing at Work Space, Document, Desktop and Search indicate points of entry into the interface. The user may wish for general entry via an application icon on the desktop or directly into the user Work Space, or wish to access a document via Document or to enter the Search function. The right-angle-cornered boxes denote screens and the round-cornered boxes denote other content components. The solid lines show the hierarchy (which can be navigated) within the UI structure and the dotted lines show other navigational options across hierarchies. Obviously, if the lower-level blueprints are taken into account, there can be a vast range of navigational interconnections across a whole site or application. The character of these connections can characterise the site, with some being strongly hierarchical whilst others are contextual, allowing navigation across hierarchies as required. Users may even be able to define their own navigational preferences or base their route on search results and 'help' functionality.

The diagram shows two aspects of how the UI works: how the users work and utilise their work space, and how the documents can be used. Let's consider how users work. The system has a concept of a User Desk, where the user can organise their work on documents. The functions they can undertake are shown in the hierarchy under User Desk. Users can see notifications of documents they have published or are expected to read. They can bookmark documents and can work on them in their own work space. Their work space is divided into two areas: one for work they do on their own and another where they can work in association with other colleagues (that is, as co-authors of documents). From this work space, they can access other parts of the hierarchy (information or contents space). They can access documents either directly, via some type of navigation or through a directory. They can publish information and documents into an area that others can access, or look there for recently published material. They also have a kind of chat room called the public space. They can access management information reports and search for manually held material (Records Administration and Archive). Finally, they can also search for material, either internally or externally. We can see that the diagram has succinctly modelled key aspects of the user's tasks and workflow.

The screens and contents components can be content mapped to the data sources. Contents components such as Notification, Owned Documents and Personal and Combined Work Areas identify the information needed from the underlying data sources. The workspace requirement looks interesting and may indicate custom metadata. This feature illustrates nicely that blueprinting, like nearly all design modelling techniques, can be used as an analysis tool. If I were using it to analyse an existing system, I would be asking myself at this point how they identified material that is being currently worked on in the work areas. Conversely, if I were using the blueprint to design a new system, I would be wondering what mechanism I would choose to identify documents being currently worked on. New metadata?

Producing blueprints is rather open ended, but in general it follows the process outlined here.

1. Identify the screens and the other contents components. This really focuses on two aspects. How does the user work? And how is the contents chunked? We will obviously use all the user needs analysis and requirements gathering techniques that are necessary, and make sure we have full data source definitions.
2. Establish, or take cognisance of, project standards for screen design and UI navigation.
3. Identify and understand any applications that are to be used for content management, for example, search engines and data extraction and manipulation suites.
4. Model each hierarchy at the high level and model contextual navigations as well if necessary. Combine onto one model.
5. Model any lower-level blueprints.
6. Review and evaluate. (This should be done throughout the process.)

Remember that this is part of user-centred design, so users should be integral to the process.

Blueprints are a particularly good technique for communicating the overall UI design, and they provide a vehicle for focused discussion on the overall management of content and information in the UI. They normalise a lot of ideas and identify any that will not fly, as it is at this point that we begin to ask how we will physically achieve the contents management and navigations identified in requirements. Blueprints are a good validation technique for requirements.

Screens, using wireframes, are often developed in parallel with blueprints. It sometimes becomes obvious that a screen is getting too cluttered while at other times one screen can do the work of two; the blueprint will therefore need to be updated and the content management refined.

Controlled vocabularies

Controlled vocabularies are used to ensure that the terms used in searching are the ones that are effective in the system context. This works in two ways. Firstly, for specific searches, the data sources in the system use these controlled terms, as use of the terms will maximise data recovery from them. Secondly, for more general searching (such as information on a subject area on the web) a range of alternative controlled terms, to that input by the user, may increase the hit rate. There are various mechanisms for both types of search. For specific searching, authority files and classification schemes are useful while for general searches synonyms and thesauri are appropriate.

Authority files contain the correct terms for a particular thing. They are often used in libraries to ensure that the user is looking for the correct author or subject terms in the catalogue. A postcode file is an authority file as the postcode is either in it or not; a country code file is another (for example, F, USA, S). The system will check the input, generally as an input of known type, and accept or reject it. (Alternatively, it may suggest a term that *is* allowed which is similar to the input, thus identifying to the user what the accepted term is.) Authority files have limited use but are powerful when used correctly.

Classification schemes guide the user to classify their queries by narrowing the search requirements. Again, libraries use these for subject categories. Online catalogues often classify products for sales, first under a generic heading such as electrical goods and then more specifically, for example, under ‘televisions’. Again, these classification schemes can introduce users to preferred terms and rapidly focus their search.

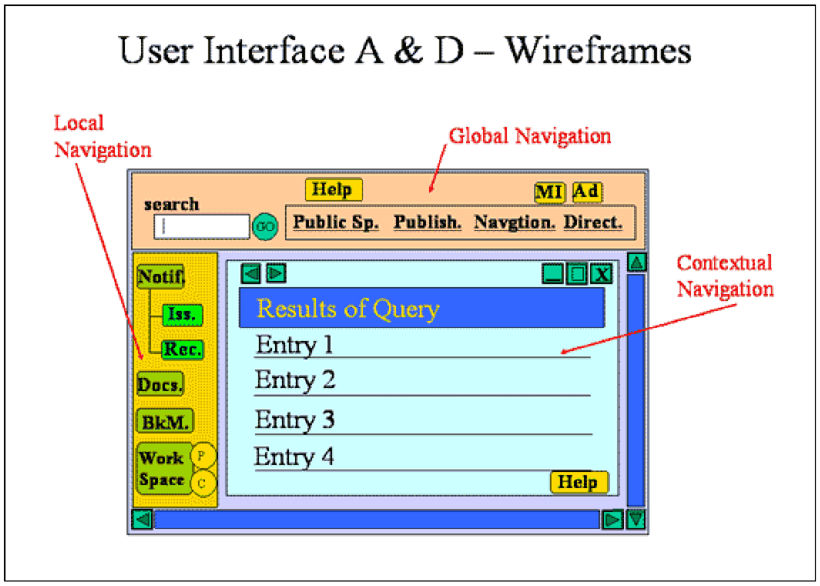
Synonyms (which can include incorrect spellings) can be used to identify variants of terms and to broaden the search criteria. Sometimes applications are programmed to record synonyms users have used, including incorrect spellings, and to build up a list. This can sometimes get out of hand and lead to confusion.

Thesauri are the most complex of controlled vocabularies. A thesaurus of associated terms is built up that contains wider, narrower, related and variant terms, amongst others. For example, for ‘television’, a wider term might be ‘video entertainment’; variant terms might be ‘TV’ and ‘telly’; narrower terms might be ‘flat screen TV’, ‘LCD TV’, ‘widescreen TV’ and ‘portable TV’, while a related term might be ‘set-top box’. Thesauri also need to distinguish homonyms (These are words that are spelt the same but mean different things. For example, the word ‘tank’ can mean a military vehicle or a large container for water.) Thesauri are difficult and resource intensive to build and are only gradually coming into use.

Screen design – wireframes

Screen design typically takes the route of early mock-ups using wireframes followed by prototyping. Wireframes are so-called as they used outline shapes to illustrate the design of the screen but they do not have to do so. I have used *PowerPoint* to produce one as an example. This is shown in Figure 6, which shows a wireframe for the Work Space screen from the blueprint in Figure 5. This is a little busier than some wireframes I have seen where actual text and standard GUI icons (such as the close window button) are not shown. I prefer my approach as it highlights the relationship of the screen to the blueprint and helps to validate it. In Figure 6, the navigations modelled in the blueprint, from the Work Space screen, are supported by the icons modelled in the screen.

Figure 6. Wireframe.



I have tried to follow some of the screen design principles here. I have divided the screen into the functions it supports and tried to emphasise this by the use of colour to distinguish and contrast them (although my choice of colours was limited and is not too good). There are three main areas, each of which is concerned with a level of navigation as this is largely a navigation screen.

Firstly, at the top of the screen I have included global navigation to other hierarchies in the UI, Information Resources and Management Information and Administration. I have included all the links to Information Resources but have only a single icon link to each of the Management Information and Administration functions. This is so I do not clutter the screen and as these functions are not used often I have minimised their linkage. I have chosen to create further screens to navigate to those areas, as the extra navigational level should not prove inconvenient for infrequently used options.

To the left hand side I have place local navigation icons to link to functions in the local User Desk hierarchy.

The rest of the screen, the larger portion, I have devoted to the results of any search, or calling up of documents, notifications or bookmarks. This window will allow contextual navigation to the document base. It also has contextual help, while access to the global help function is at the top of the screen in the global navigation area.

The process of screen design is quite open ended and no two designers would go about it in the same way. There will be some specification as to what the screen should contain and usually a set of project, and perhaps corporate, screen design standards predefining certain aspects of the screen structure, colour palette and typography. Some sort of grid on which to block out the main areas of the screen is useful, similar to the way layout editors design pages for magazines. It will also be useful to think how you want the user's eye to move over the screen. Usually the eye rests on the centre but you may want to wrest them away to the edge (if you are designing a web screen where you want to advertise other offers you may brighten the edge where the adverts are situated, or make them scroll every so often, or make them pop up) or to other areas. Remember movement on a screen is eye catching.

Finally consistency is required both within and between screens. The overall layout of my Work Desk screen should be consistent with similar screens elsewhere in the system. If elements are repeated between screens then it may be useful to extract them and design them on their own so that they are created once rather than on every screen. They will then appear consistent between screens. Also an element, if possible, should appear in a similar position whenever it appears on a screen.

Once a few screens have been designed and a consistent style developed the rest follow more easily and it becomes a little like a production line. It is therefore important to get the process correct right at the beginning.

Section Review

Based on the learning outcomes in this section you should have learnt the following:

1. Content mapping including blueprinting, metadata and controlled vocabularies.

   Content mapping ensures that the contents of the UI can be generated from the information sources, available to the system, in the way the user wants.

   Blueprinting models the linking of pages and supporting contents components in the UI. Blueprints can be decomposed to lower levels. They can contain organisational and task-oriented elements and at lower levels may be structured to reflect task processing.

   In the UI, metadata contains information about the data contents used. This allows for more versatile and focused information access.

   Controlled vocabularies are used to ensure the correct terms are used in searches. Authority files and classification schemes elicit specific terms whilst synonyms and thesauri generate equivalent and more generic search terms to improve the recall and precision ratios.

2. Screen design and wireframes.

   Wireframes are a technique for designing screens. Screens should be designed to guide the user to complete their tasks successfully. They should be uncluttered, with elements such as colour and the division of the screen designed to support the functionality for which the screen is used.

2.5.5 User interface modelling - dialog design approach

Section learning outcomes

After completing this section you will have knowledge of the following:

The dialog design approach, including dialog and menu design.

Introduction

Dialog design seeks to model and specify how a user will interact with the system when completing a task. The inputs to this technique are information about the users and how they use the system, and information about the tasks, and the content and processes that are involved. Dialog design is a process modelling technique as it models the process of the user's interaction with the UI.

In a conventional development project, information about users is contained in user and user role descriptions. User roles are the parts users undertake when using the system. For example, in a sales system there will be customer, sales person, sales manager and stock controller user roles, amongst others. The tasks users undertake will be defined as functions, for example, the 'enter customer order' function. These functions will have been developed using various process modelling methods to validate and specify the requirements, such as data flow modelling (DFM). DFM identifies which data flows into (input) and out of (output) processes (see Topic 4). Similarly, other process modelling techniques will also identify what data processes use. This input and output data can be used to define the information content at the UI and forms the basis of the dialog design.

To identify which dialogs are required, we need to identify which users interact with which functions. This is usually achieved by using a user role to function matrix. It may be that both customers and sales persons can place orders in our sales system (customers do so when ordering over the web), and there are two user roles interacting with the function 'enter sales order'. This identifies two dialogs. They may be so similar that one dialog is sufficient (if, for example, both use the web interface to the system), but they may be different and require two distinct dialogs.

The order in which the activities in a task are undertaken also needs to be understood and this may be provided by such techniques as activity diagrams (see Topic 4) and user task and workflow modelling.

The dialog design can now commence. The sequence in which the input/output (I/O) data associated with each function is processed by the user is modelled in an I/O model. This model is then reviewed and converted into a dialog model by adding any extra dialog not represented by function I/O data. The dialog model is then reviewed to identify suitable screens within the dialogue. Once the screens have been identified, the menu (navigation) structure can then be modelled. Error handling and help functionality is then added. The design is then reviewed and any prototyping commenced. This is the dialog design approach to UI development. The method may be summarised as follows:

1. Cross-reference user roles and functions.
2. Identify I/O data for functions.
3. Produce I/O model (diagram and descriptions).
4. Produce dialog model from I/O model (diagram and descriptions).

5. Identify and develop screens.
6. Produce menu structures (diagram and descriptions).
7. Add error, help and security dialogue.
8. Review and revise.
9. Prototype as required.

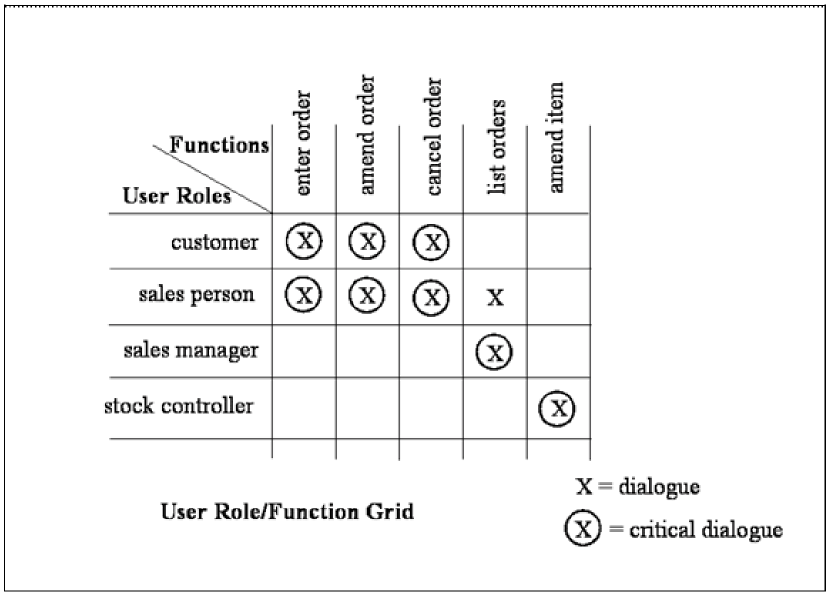
Steps 1 to 6 are essentially sequential. Steps 7 to 9 could happen at various stages throughout the process. Error, help and security dialogue is contextual and will happen throughout Steps 4, 5 and 6. So will the option to prototype. Reviewing and revising can happen at any stage although there will obviously be major milestone reviews for sign-off.

We shall now look at these steps in more detail.

Cross-referencing user roles and functions

Figure 7 shows part of a user function cross-reference grid for the sales system.

Figure 7. User Role/Function Grid.



Functions / User Roles	enter order	amend order	cancel order	list orders	amend item
customer	Ⓧ	Ⓧ	Ⓧ		
sales person	Ⓧ	Ⓧ	Ⓧ	X	
sales manager				Ⓧ	
stock controller					Ⓧ

User Role/Function Grid

X = dialogue
Ⓧ = critical dialogue

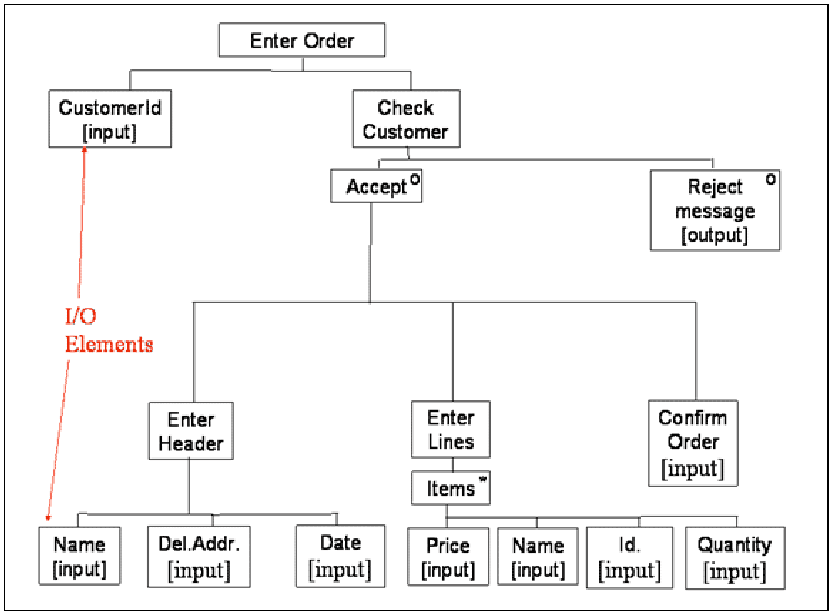
Producing the grid is straightforward once functions and user roles have been identified and documented. If functions have not been identified, then other process definitions such as tasks and activities can be used. User roles are ubiquitous, as some sort of classification of users is required for aspects such as security and data access.

These grids can also be used to prioritise the dialogs and identify those that are critical to the system's success. These may be appropriate for early development and prototyping to develop and validate the UI standards and the proposed implementation environment. Dialogs may be critical for a variety of reasons including high frequency, complex to process, new functionality, and politically sensitive.

Produce I/O model

As noted in the introduction, the I/O data is identified by the process modelling that has already been completed. The I/O data for the function is reviewed against any activity or task models, or descriptions of how the user completes the function. The I/O data is then modelled using the Jackson structure modelling techniques (discussed in Topic 4) to show the sequence in which it is used when carrying out the function processing. An I/O diagram for a sales person entering an order is shown in Figure 8.

Figure 8.I/O Model Diagram.



There are two types of box on this diagram. The first contains items of I/O data which are the leaves on the diagram and are called **I/O elements**. The second are node boxes and show how the elements are grouped. The boxes, in keeping with the Jackson method, are either sequence, selection (indicated by a 'o' in the corner) or iteration (indicated by a '*' in the corner) boxes.

In this diagram, to enter an order, the sales person inputs the customer id, which is then checked for the customer's creditworthiness. If rejected, then the dialog finishes and a brief message is output. If acceptable, then the entity header is input (customer name, delivery address and order date) followed by order lines, one for each item ordered (item price, name, identifier and quantity ordered). The I/O then ceases. As with all models, the I/O model also has a description, which will contain the data that each element contains and specific comments.

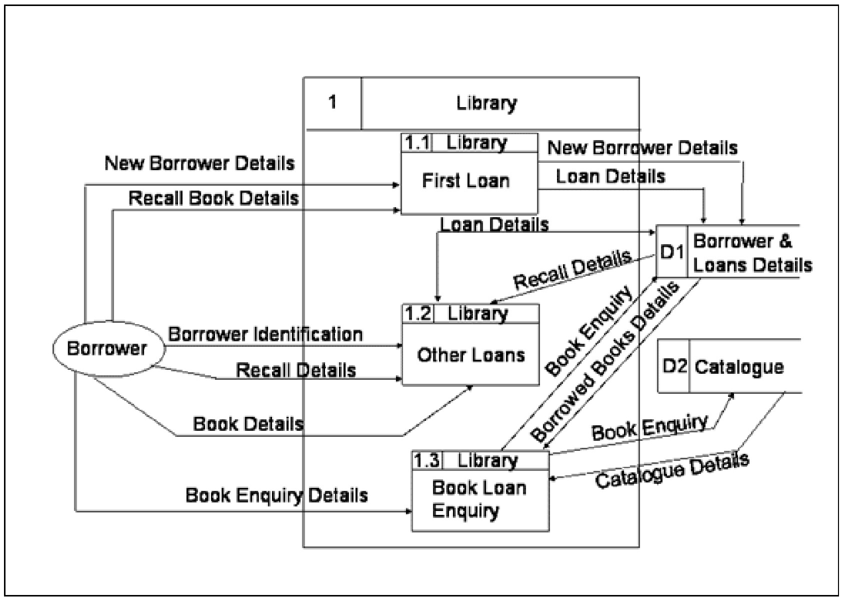
We can see from this description that the I/O model is a halfway house that models the content but not the full dialog.

Online I/O is obviously not the only type of I/O because we could use a printer to output data. This is known as off-line I/O, and such output is modelled as a separate associated model. As off-line models are for output only there is no need to use the [input] and [output] qualifiers in the I/O elements.

Exercise 1

For the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] case study, produce an I/O diagram for the Loan Book function. This function is represented in the DFM by process 1.2, Other Loans, in the Level 2 data flow diagram shown in Figure E1 below.

Figure E1. Level 2 Logicalised DFD.



The Loan Book function does not include the first loan a borrower makes, which is a separate function, called First Book Loan. Do not model the first loan of a book; only model the normal loan function Loan Book. From Figure E1 you will see that there are three data flows involved in the loan of a book associated with process 1.2, Other Loans. These are Borrower Identification, Book Details and Loan Details. The first two are input data flows and the last, Loan Details, an input and output data flow. The data in these data flows is shown in Table E1.

Table E1. Data Flow Content in the Library System Case Study.

Data Flow	Attributes	Comments
Borrower Identification	Ticket No.	Borrower ticket number.
Book Details	Acquisition No.	The unique number identifying an individual copy of a book.
Loan Details (from data store)	Ticket No.Acquisition No.Loan DateTitleBorrower Name	The first three attributes are the key of Loan and returned for each existing loan. Title and Borrower Name are included in case a list of loans is requested by the borrower.
	Fines Total	The total amount of outstanding fines.
Loan Details (to data store)	Ticket No.	Borrower ticket number.
	Acquisition No.	The unique number identifying an individual copy of a book.

A task description for how a librarian will loan a book in the new computerised system has been produced (see Table E2).

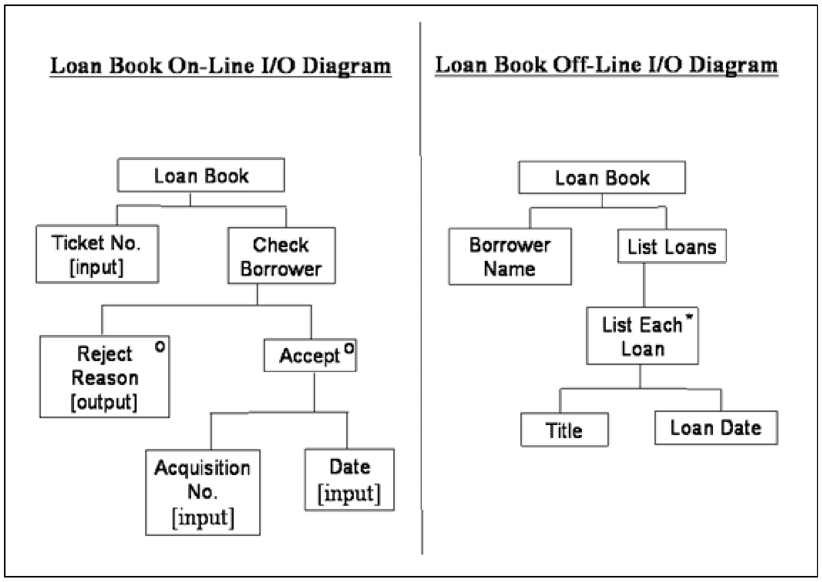
Table E2. Task Description for Normal Book Loan in Library System Case Study

When the borrower brings the book for loan to the loans desk, the librarian will enter the borrower's ticket number, using a bar code reader to read the code on the borrower's library card. The system will check to see if the borrower has reached their loan limit (20 books if a student), exceeded the outstanding fines allowance (£20) or is trying to borrow a book outside term time (if a student). If they do meet any of these criteria, then the loan is rejected and a reason output. If the borrower has reached the loan limit they may request the librarian to print them out which loans are outstanding. (The borrower name, the title and the loan date are printed out for each loan.) If the loan is accepted, then a bar code reader is used to read the acquisition number from the book. The date of the loan is recorded.

Now produce the I/O diagram using the data flows and task description.

Tutor's comments

Figure E2. I/O Diagrams for the Loan Book Function.

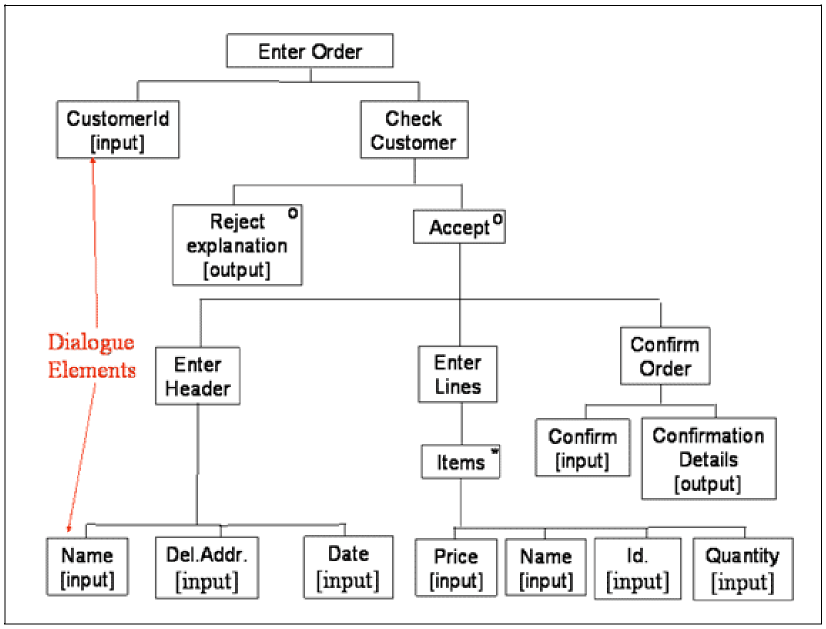


As there is a printing function there are two diagrams, for the online and off-line parts of the Loan Book function. The only additional data to that on the data flows is the Reject Reason. Whether these messages are held in the code or on the database is an implementation issue, which is why it has not appeared in a data flow.

Produce dialog model

The I/O model is taken as the basis for the dialogue model, in which each I/O element becomes a dialog element. Typically the basic dialog diagram simply replicates the I/O diagram, but there is an opportunity here to expand the structure, if necessary, and also reorganise it to better reflect the dialog. There may be extra I/O developed in order to facilitate the task at the UI. Dialog elements may be repositioned to facilitate their grouping into screens (which is explained in the next subsection), or sections of the structure may be reworked to allow for proper dialog. Figure 9 is the dialog diagram for the I/O diagram in Figure 8.

Figure 9. Dialog Structure.



In this diagram we have made some amendments to the I/O model and changed the message, in the Reject dialogue element, to an explanation, giving the user a fuller reason for the rejection. The Reject element has been repositioned closer to the CustomerId element, as they may both be designed into the same screen. Remember you can only reposition elements if the diagram structure allows it. We have also included extra dialog for confirmation of the order, adding an extra dialog element output, Confirmation Details. I would have made these changes based on my experience and consultation with the users. There may be further changes later, during prototyping, for example if it is shown that the customer check takes longer than the average response time to process, and a message output is therefore necessary to inform the user that a check is taking place.

At this stage we also need to review closely related dialogs to see if they can be combined. We have already noted that a customer may place an order themselves over the web or via a sales person. They may both use the same dialog, but customer screens often have more dialog to coach them through the task. There may also be technical design considerations that need to be considered and economies of scale. The sales system may be designed using thin-client architecture with all users using a web-type interface. In this case the sales person may use the customer dialog and screen, but with short-cuts available to enable them to navigate the screen for telephone sales purposes.

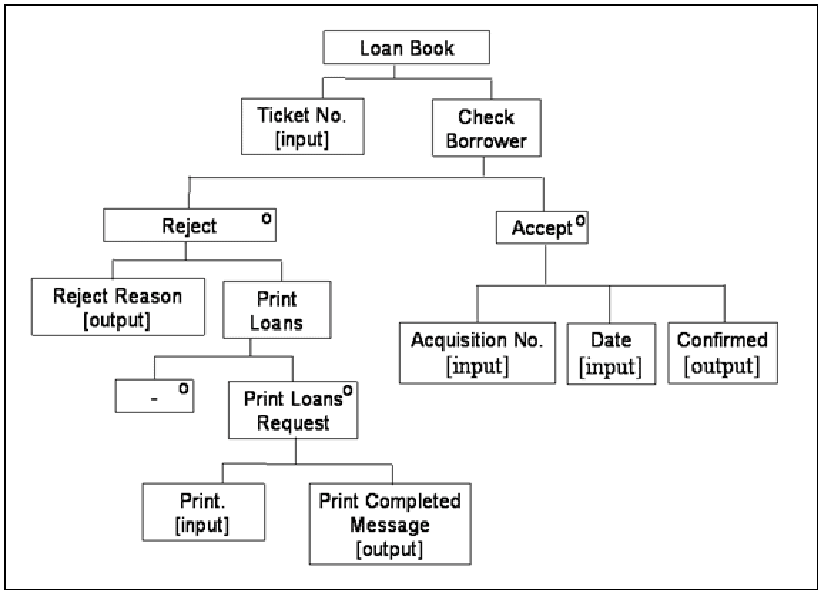
A description of the dialog model is produced which describes each dialog element, the data content and its use.

Exercise 2

Based on the I/O diagram (solution) and task description, given in Exercise 1, produce a dialog structure for the Loan Book function.

Tutor's comments

Figure E3. Dialog Structure for the Loan Book Function.



This is a straightforward task and there is little dialog to add to the basic I/O. The main area requiring extra dialog was the printing of the list of loans. This looks a bit complicated due to the way Jackson structures work. Remember that each box at the same level, in a hierarchy on the diagram, must be of the same type (sequence, selection or iteration). In essence, we start with the sequence; output the Reject Reason and print out the list of loans if the user requests it. We do not have to print the loan list, so there is a choice under the Print Loans node of 'do nothing' (the dialog element with just a hyphen in it) or enter a Print Loans Request. In the latter case, the user enters Print, and when the print is finished a Print Completed Message is output.

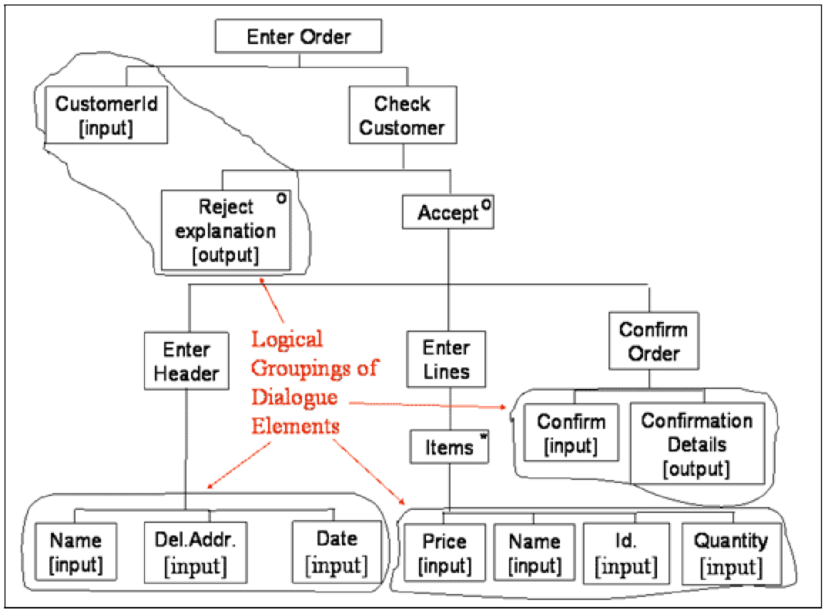
I also added a simple output to confirm that the loan had been successfully entered. This could be an audio beep on the system (as happens in my local library) rather than a message. The decision whether to have a beep is one that will be taken in implementation.

Obviously no dialog structure is required for the off-line I/O model.

Identify and develop screens

In dialog design, the main philosophy for identifying the screens that support a dialog is that they should present closely related activities that are logically associated together. Consequently, to identify screens the dialog model is analysed for logical groupings of dialog elements. Logical groupings of dialogue elements for the dialog model in Figure 9 are shown in Figure 10.

Figure 10. Logical Groupings of Dialog Elements.



There are four logical groupings here which identify four screens. Firstly, inputting the CustomerId and outputting any explanation of a rejection is one group/screen. The entering of order header and order line details are two further groupings/screens. The order line entry screen will repeat for each line entered. Finally, there is a confirmation group that has Confirm input and Confirmation Details output (essentially details of the order). These logical groupings are circled on the diagram (as in Figure 10) and described in the dialog model descriptions.

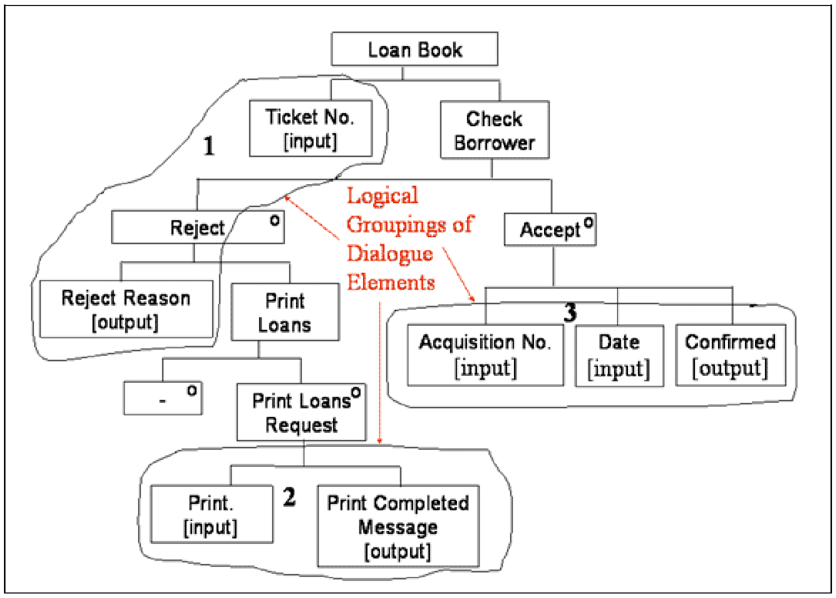
Screen definition and design will begin. The production of screen templates (perhaps as wireframes) and prototyping will commence. All the activities described in Section 4 that are required for the design and production of screens will take place here.

Exercise 3

Identify logical groupings of dialogue elements using the dialog structure from Exercise 2. Identify the screens indicated and produce a very simple wireframe for the main loan screen once the loan has been accepted.

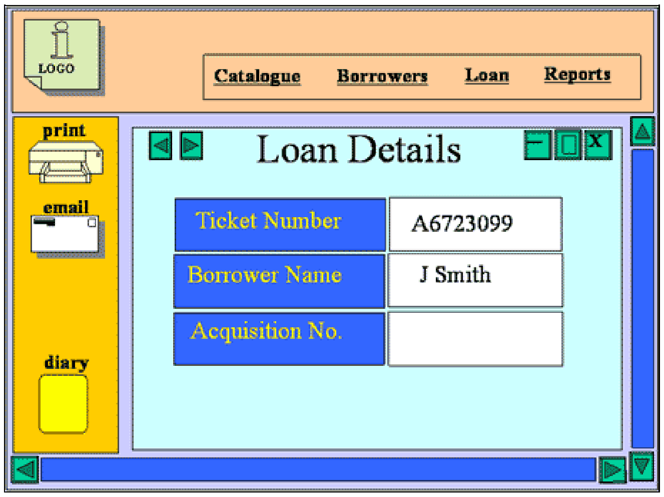
Tutor's comments

Figure E4. Logical Dialog Element Groupings for the Loan Book Dialog.



I have chosen three groupings. Group 1 is for the entry of the borrower's ticket number and the results of the check. Group 2 is for the printing of the loan list. Group 3 is for the entry of the Acquisition Number and Date followed by the confirmation. You may have decided that Groups 1 and 2 could be combined with all the dialog for identifying and checking the borrower put in one group. This is an equally acceptable solution and it may be better than mine. Also another window, or a pop-up, could be used for the print dialog. This is the province of screen design.

Figure E5. Wireframe for the Loan Details Screen.



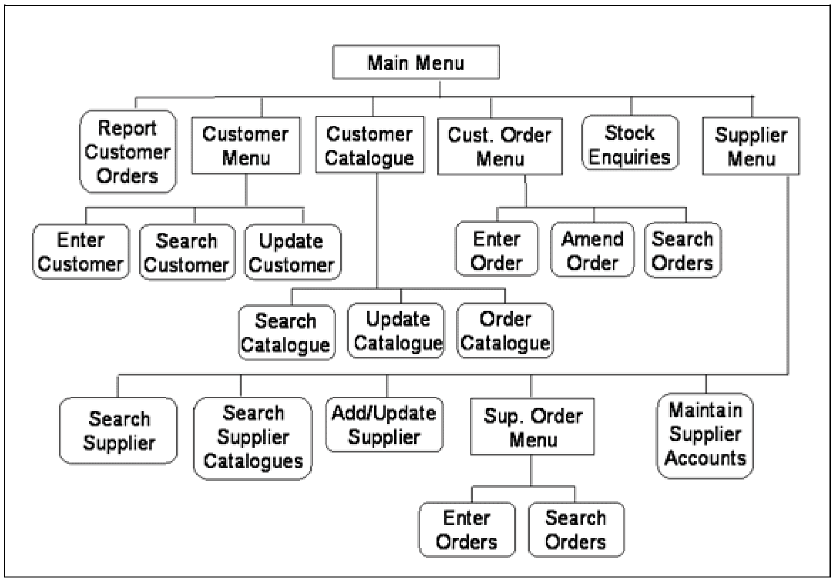
Ticket Number	A6723099
Borrower Name	J Smith
Acquisition No.	

My screen design is a simple one: main loan window overlays the previous ticket entry screen, holds the ticket number and presents the user name in case the librarian needs it. The acquisition number can be entered and the current date is updated to the loan record transparently as it is system generated. Your screen may or may not be like mine, but as the specification was limited, and provided you have the basic loan functionality, and some other useful navigation and generic functions, it is probably OK.

Produce menu structures

Having identified which screens are required, we can begin to design menu structures. These show how the dialogs are to be associated and navigated to in the UI. The basic method is to identify the dialogs that will be at the bottom level of the menu hierarchy and then group them into sub-menus to develop the intermediate level. Intermediate menus, and dialogs operating at that level, can then be further grouped into higher-level menus. The overall design of the menu structure will be very dependent on the nature of the navigations required. Typically, an overall, hierarchical menu will be designed and the accompanying menu structures will show any cross-navigation between the hierarchies. These accompanying cross-navigation structures are often called **command structures** as they show how the user can select their way around the functionality in the menu. Figure 11 shows a menu structure that includes the sub-menu for the 'enter order' dialog, which is developed from the logical groupings dialog structure shown in Figure 10.

Figure 11. Menu Structure.



The round-cornered boxes are dialogs and are the leaves on the model. The square-cornered boxes are the menus and will link to other sub-menus and/or screens. Note that a menu structure is not a Jackson structure model and does not model sequence.

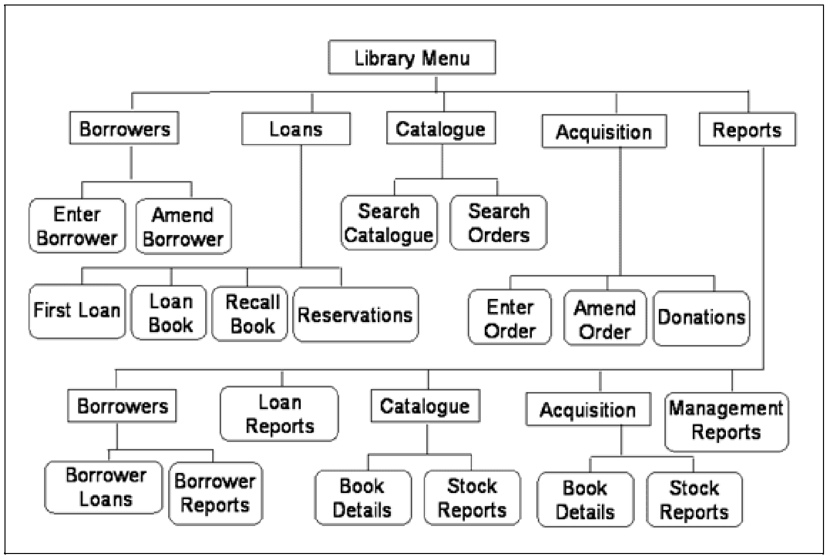
Here the Enter Order menu is a sub-menu of the high-level Customer Order menu, but descent through the Customer Order menu command structure is not the only way that we might navigate to the Enter Order menu. We could browse the customer catalogue and order through it by selecting an item and then going to the Enter Order menu to complete the order. There could be an alternate command structure to the Customer Catalogue menu based on going to the Search Catalogue screen, using the screen to select an item to be ordered from the catalogue, and then progressing to the Enter Order menu.

Exercise 4

Construct a high- to medium-level menu structure for the Porterhouse Library system [link: http://study.conted.ox.ac.uk/mod/resource/view.php?id=13622] case study.

Tutor's comments

Figure E6. Porterhouse Library System Menu.



I have based this menu on the basic functionality identified in the requirements in the case study. The main design choice I have made is to include a reports hierarchy. There will obviously be a lot of cross-navigation from other functions in the menu to the reports that are relevant to them, but there will be a large number of reports and a report menu is required I think.

Add error and help dialog

Dialog dealing with errors, help and security is generally contextual, that is, the types of error handling, help and security options offered to the user will be dependant on what the user is doing at the time.

Error Management. At whatever level the user is operating, the user should be able to recover from errors, and should be alerted to their occurrence and provided with explanations. Error functionality may be global, for example 'undo', or specific to an entry field. At the I/O level, the user should be protected from entering incorrect data in terms of format and scope. Controlled vocabularies and dropdown menus are among many mechanisms available to combat this type of error. Also users should be able to change their input when they realise they have typed it in incorrectly – for example, an incorrect customer number. At the dialog level, users may have completed a dialog and may be ready to submit, only to find that errors have occurred – no values have been entered in mandatory fields. They should be offered the option to revise their entry without have to re-enter correct data. At the next level, they may wish to reverse their actions after a period of time – for example, to cancel an order. At all levels, users need to be able to correct mistakes.

Help. Help should be ubiquitous and present within the application or in the application environment and operating system. It can be present at the level of bringing up a brief description of a button's function when a mouse cursor hovers over it, or present as a high-level global function accessed by a top-level menu. In supporting functionality, it should be context sensitive, focused on what the user is doing as well as providing background reference and information.

Security. In terms of dialog, security focuses mainly on the permissions necessary to access various dialogs. The mechanism for granting permission will focus on passwords and user identifiers. Sometimes specific security on working objects (such as documents) will be required, with passwords generated by the users rather than administrators.

What has been highlighted in this brief discussion of error management help and security is that a great deal of it will be provided by third party applications used by the system (for example, the operating system and document management software), and considerable elements of dialog will be provided by them. This is why when designing dialogs, and screens, knowledge of system architecture, and what the different elements provide, is ultimately required. In fact, consideration of the error, help and security dialog required may dictate elements of that architecture.

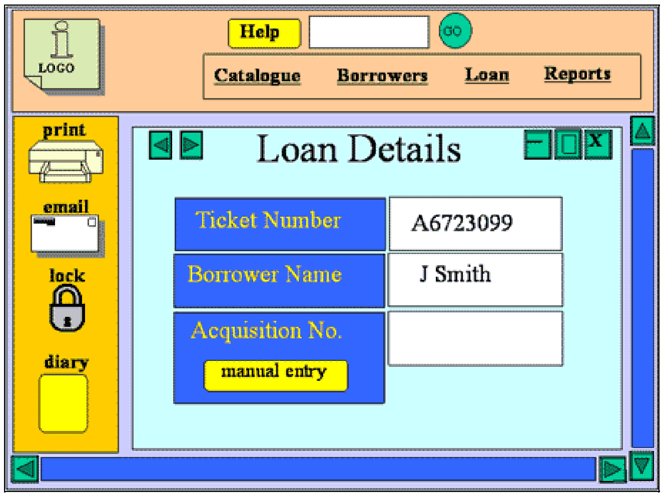
Overall, when designing dialog for a system, developers should concentrate on the error handling, help and security necessary for delivering the functionality of the system.

Pause for Thought

Where might you consider it is worth adding security to the screen given in the answer to Exercise 3?

Tutor's comments

Figure E7. The Addition of Help, Security and Error Handling.



I have extended my screen design. There is an option of manual entry for the acquisition number to forestall any problems in reading the bar code. This is managing errors. A similar approach may have been adopted in the previous screen for Ticket Number entry using a bar code reader. (We all see this option in use in supermarkets and other shops.) Other error management may be implicit as regards mistakes in data entry where the user can overwrite what they have entered. A generic application help function has been placed at the top of the screen and a search facility associated with it. I have included an option to lock the screen as a global function.

I am sure you may have thought of others, but always review them to ensure they are relevant.

Review and revise

As already noted, review and revision is necessary at every stage. The whole development of dialog, like the development of the UI itself, is iterative. Combined dialog, menus and screens are the interface. Modern systems require UIs that are multifunctional to support a wide range of users. The design of the interface is complex, and review is essential in ensuring the system is properly developed.

In reviewing an interface, we need to ask whether it embodies the UI design principles and whether it is fit for purpose, enabling the functions the system is implemented to support. Is it consistent and resilient, enabling the user to conduct their work competently in a safe and secure way?

If we consider the design for the Enter Order dialogue, we need to ask whether it is fit for purpose. It offers a limited and rather basic dialog for the entry of a customer order by a sales person. So it is fit for purpose up to a certain level. It could not be used as a dialog for the customer to enter an order, however, since it does not offer the extra dialog a customer would require, and it does not follow the sequence required. A customer ordering through the catalogue would want to enter the item details in the order line first, rather than the header details (see Figure 9). Also, the user would expect to have their details held, so that they would not have to enter them every time they used the system.

Finally, if the item was ordered through the catalogue, data would be entered into the order by the system rather than by the user. The customer order entry dialog would be completely different from the one we have designed. We can see from this brief review that a focused review can take forward the design of the dialog and expand our development of the UI.

Prototype as required

The main elements of the dialog design that will be prototyped are the screens and the menu structure. Critical screens and menus should be prototyped as early as possible.

Error handling, help and security should also be prototyped. This will lead on to the need for the early testing of the components of the system architecture that support the UI , and to the prototyping of any configuration or development of them.

Section review

Based on the learning outcomes in this section you should have learnt the following:

The dialog design approach, including dialog and menu design.

The dialogue design encompasses the following approach:

- Cross-reference user roles and functions.
- Identify I/O data for functions.
- Identify I/O data for functions and produce I/O model (diagram and descriptions).
- Produce dialogue model from I/O model (diagram and descriptions).
- Identify and develop screens.
- Produce menu structures (diagram and descriptions).
- Add error, help and security dialogue.
- Review and revise.
- Prototype as required.

This approach builds on the process design undertaken during analysis and requirements specification. The identification of input and output data determines the contents of the UI. The dialogs are consequently based on I/O design. The identification and development of screens is based on the analysis of the dialog model by identifying associated dialog elements, which are also used as the bases of menu design. Review and revision are key to the iterative approach that is necessary for dialog design. Prototyping of screens and menus enables critical elements of the UI to be trialled early.

2.5.6 References

References

1. Garrett, J. J., 2003 *The Elements of User Experience*, (New York: AIGA), ISBN 0735712026.

   This is a short book giving a nice overview of the subject from a web designer's point of view.

2. Preece, J. (ed.), 1993 *A Guide to Usability – Human Factors in Computing* , (Addison-Wesley), ISBN 020162768X.

   A good guide to the topic.

3. Rosenfeld, L. and Morville, P., 2002 *Information Architecture for the World Wide Web*, (Sebastopol, USA : O'Reilly), ISBN 0596000359.

   This is an exhaustive textbook with a lot of interesting material on all aspects of the subject and containing good examples.

4. Shneiderman, B., 1992 *Designing the User Interface* , (Addison-Wesley), ISBN 0321269780

   A thorough text on the subject.

5. Sommerville, I., 2004 *Software Engineering* , (Addison-Wesley), ISBN 0321210263.

A general textbook with a useful introductory chapter on interface design.