Dynamic Modelling and Process Re-engineering using UML
Part 1 – Use case and Sequence diagrams

Written by: Robin Beaumont e-mail: robin@organplayers.co.uk

Date last updated: Tuesday, 06 September 2011
Version: 4

How this document should be used:
This document has been designed to be suitable for web-based and face-to-face teaching. The text has been made to be as interactive as possible with exercises, Multiple Choice Questions (MCQs) and web-based exercises.

If you are using this document as part of a web-based course you are urged to use the online discussion board to discuss the issues raised in this document and share your solutions with other students.

This document is aimed for two types of people:

• Those who wish to become involved in database development or process re-engineering but are not interested in the nuts and bolts of programming. Such people are commonly called domain experts and act as bridges between a professional group (e.g. medics, solicitors etc) to which they belong and IT experts.

• As an introduction for those just beginning professional computer science courses

Acknowledgements
I would like to thanks all the students that have used this material from the early 1990’s up until the present who have provided invaluable, references, comments and suggestions.

I hope you enjoy working through this chapter.

Robin Beaumont
## Contents

1. **Before You Start** .................................................................................................................. 3  
   1.1 Required Resources ........................................................................................................... 3

2. **Learning Outcomes** .......................................................................................................... 4

3. **Introduction** ..................................................................................................................... 5  
   3.1 Use case diagrams – a love affair for managers and an irritant to modelers .......... 5
   3.2 Where Does the Dynamic Aspect Fit in the Modelling Process? ......................... 6

4. **Use Case Diagram** .......................................................................................................... 7

5. What is the Dynamic Aspect? ............................................................................................... 11

6. **Sequence Diagrams (SD)** ............................................................................................... 13  
   6.1 Frames ............................................................................................................................... 14
   6.2 Lifelines ............................................................................................................................ 14
   6.3 The Message ..................................................................................................................... 15  
      6.3.1 Message Sorts ........................................................................................................... 16  
         6.3.1.1 Asynchronous signal message ......................................................................... 16
         6.3.1.2 Asynchronous call message ........................................................................... 16
         6.3.1.3 Synchronous call message ............................................................................. 16
         6.3.1.4 Reply message ................................................................................................. 16
         6.3.1.5 Create message ............................................................................................... 17
         6.3.1.6 Delete message ............................................................................................... 17
   6.3.2 Execution specification ................................................................................................. 17
   6.3.3 Slanted Arrows ........................................................................................................... 17
   6.3.4 Parameters, Arguments and return values ............................................................... 18
   6.3.5 Summary .................................................................................................................... 19

6.4 Showing More Detail – Interaction Use ........................................................................... 20  
   6.4.1 Gates ........................................................................................................................... 21

6.5 Control ................................................................................................................................ 22  
   6.5.1 Iteration ....................................................................................................................... 22
   6.5.2 Single Branching ......................................................................................................... 23
   6.5.3 Multiple Branching ..................................................................................................... 23
   6.5.4 Concurrent Messages ............................................................................................... 23

6.6 Nesting .................................................................................................................................. 24

6.7 Incremental Development .................................................................................................. 24

6.8 Exercises - Sequence Diagrams .......................................................................................... 25

7. **Business Process Re-engineering (BPR)** ...................................................................... 29

8. **Relationship between Sequence Diagrams, Use Cases and Class Diagrams** .......... 31  
   8.1 Asynchronous trigger Messages .................................................................................... 31
   8.2 Call Messages .................................................................................................................. 32
   8.3 Use Cases ....................................................................................................................... 32

9. **Instance scenarios** ............................................................................................................ 34

10. **Importance of Dynamic Modelling and testing** .......................................................... 35

11. **Multiple Choice Questions (MCQs)** ........................................................................... 36

12. **Summary** ....................................................................................................................... 39

13. **References** .................................................................................................................... 39

14. **Links** ............................................................................................................................... 40

15. **Appendix A – pre uml 2 message syntax** ................................................................. 41
1. Before You Start

This document assumes that you have the following prerequisite knowledge and skills:

<table>
<thead>
<tr>
<th>Skills:</th>
<th>Where can I gain the skills and knowledge?</th>
</tr>
</thead>
<tbody>
<tr>
<td>That you have used the following features of a Database Management System (DBMS) such as open office or Access to: Create tables Create relationships (and therefore know about the relationship window) Create simple queries in the query design window Create a simple form/select query</td>
<td><a href="http://www.robin-beaumont.co.uk/virtualclassroom/contents.html">http://www.robin-beaumont.co.uk/virtualclassroom/contents.html</a> see section 8</td>
</tr>
<tr>
<td>That you have used a case tool such as VP uml or MagicDraw to create ERDs and UML Class diagrams</td>
<td><a href="http://www.robin-beaumont.co.uk/virtualclassroom/contents.html">http://www.robin-beaumont.co.uk/virtualclassroom/contents.html</a> see section 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knowledge:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Concerned with databases:</td>
<td><a href="http://www.robin-beaumont.co.uk/virtualclassroom/contents.html">http://www.robin-beaumont.co.uk/virtualclassroom/contents.html</a> see section 7</td>
</tr>
<tr>
<td>Tables, indexes and fields</td>
<td></td>
</tr>
<tr>
<td>Relationships (and the links between modelling and database implementation)</td>
<td></td>
</tr>
<tr>
<td>Queries</td>
<td></td>
</tr>
<tr>
<td>2. Concerned with modelling and class/object oriented modelling:</td>
<td><a href="http://www.robin-beaumont.co.uk/virtualclassroom/contents.html">http://www.robin-beaumont.co.uk/virtualclassroom/contents.html</a> see section 8</td>
</tr>
<tr>
<td>Class and instance models</td>
<td></td>
</tr>
<tr>
<td>Associations including aggregation and generalisation</td>
<td></td>
</tr>
<tr>
<td>Have an awareness but not in-depth knowledge of encapsulation and polymorphism</td>
<td></td>
</tr>
</tbody>
</table>

1.1 Required Resources

You need the following resources to work through this document:

Many of the concepts introduced in this document are difficult to grasp at first and are helped by experimenting with a case tool in addition to carrying out the exercises with pen and paper. One such tool is Magicdraw. The document “An Introduction to ERDs” provides details of other websites where you download alternative software.


- If you are feeling particularly masochistic you can download the latest UML standards documents at: http://www.omg.org and follow the links.
2. Learning Outcomes

This document aims to provide you with the following skills and knowledge. After you have completed it you should come back to these points, ticking off those with which you feel happy and revisiting those which you still may have problems understanding.

<table>
<thead>
<tr>
<th>Learning outcome</th>
<th>Tick box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be able to describe where the dynamic modelling process sits within the whole modelling process</td>
<td></td>
</tr>
<tr>
<td>Be able to explain what the ‘dynamic aspect’ of an class/instance is</td>
<td></td>
</tr>
<tr>
<td>Explain how the dynamic aspect of an class/instance is modelled within UML</td>
<td></td>
</tr>
<tr>
<td>Be able to describe and use the components of a Use Case diagram</td>
<td></td>
</tr>
<tr>
<td>Be able to develop Use Case diagrams</td>
<td></td>
</tr>
<tr>
<td>Understand the components of a Sequence Diagram</td>
<td></td>
</tr>
<tr>
<td>Be able to develop Sequence Diagrams from a narrative, and if necessary producing a instance scenario</td>
<td></td>
</tr>
<tr>
<td>Be able to draw Sequence Diagrams to carry out BPR</td>
<td></td>
</tr>
<tr>
<td>Be aware of the diagramming complexities of Sequence Diagrams</td>
<td></td>
</tr>
<tr>
<td>Be able to explain what business process re-engineering (BPR) is</td>
<td></td>
</tr>
<tr>
<td>Be aware of the concept of ‘configuration’ within dynamic modelling</td>
<td></td>
</tr>
<tr>
<td>Be able to demonstrate the incremental nature of developing Sequence diagrams specifically regarding message refinement, operation calls and fragments</td>
<td></td>
</tr>
<tr>
<td>Be aware of the importance of dynamic modelling in the health sector</td>
<td></td>
</tr>
<tr>
<td>Be able to describe in detail the relationship between class, sequence and Use case diagrams</td>
<td></td>
</tr>
</tbody>
</table>
3. Introduction

Having worked through the “Introduction to Classes and Instances” chapter [http://www.robin-beaumont.co.uk/virtualclassroom/chap11/s2/omt1.pdf] you will understand a wide range of concepts concerned with the UML (Unified Modelling Language). That chapter focused on the 'what' aspect of modelling, basically the information you would consider collecting to create classes and their associations, called the structural aspect. In this chapter we will move to the how aspect of modelling that is the dynamic, also called the behavioural aspect, specifically in this chapter we will investigate the dynamic interactions between instances of classes whereas in the next chapter we will consider the life cycle of instances within classes. We will do this by considering three types of UML diagram:

- Use Case
- Sequence
- State [machine] – considered in next chapter.

Defining what happens over time is not an easy subject! Rumbaugh 1991 begins his chapter on dynamic modelling with the warning 'Temporal [dynamic] relationships are difficult to understand', but rather strangely removes the sentence from the second edition (Blaha & Rumbaugh 2005) but I still think it is just as true. While I will try to concentrate on understanding the basic ideas and how to apply them I will also be introducing a degree of complexity as the UML specification has grown over the years with the inevitable introduction of more and more concepts representing greater degrees of modelling sophistication.

Before considering what dynamic modelling is exactly, I will describe where it fits in to the whole modelling process, but first a warning about Use case diagrams.

### 3.1 Use case diagrams – a love affair for managers and an irritant to modelers

I feel that it is sensible to provide a warning about use case diagrams, originally they were not in the UML specification, being part of a different modelling technique and you will notice in the section concerning Use case diagrams I warn against their overuse, however you will, like most people, probably find them initially very appealing because of their obvious simplicity. However here also lies their danger as most models are not simple once you start to investigate them in detail and modellers need to end up with detailed complete models which use case diagrams can not provide instead such model make use of more complex concepts provided by the other UML diagrams we are going to look at. Unfortunately these more semantically rich diagrams are not so immediately attractive or understandable and therefore not so attractive to managers. Dobing & Parsons 2006 provide research that validates this viewpoint.
3.2 Where Does the Dynamic Aspect Fit in the Modelling Process?

One possible process of developing a dynamic model is shown in the diagram below, you will notice how I suggest that once you have created a simple overview use case diagram you basically ignore it (some modellers would disagree with this). It should be noted that UML makes a point of stating that it is are not a 'method'. However there are numerous modelling methods around that consultants often charge ridiculous amounts of money for giving you the privilege of using, the table below lists a few.

UML provides the modeller with a set of diagramming tools which she/he is free to use or abuse. In contrast, often modelling techniques consist of very stringent sets of rules as well as diagramming tools. For example, one popular method called SSADM contains over 200 steps. While there are a huge number of methods now available it must be remembered that none of them have been tested rigorously by way of anything resembling a RCT (Randomised Controlled Trial); unfortunately some people argue that they do not require such rigorous evaluation processes.

Even though these methods may have little validity they do provide a practical framework and are popular with consultancies as their workers can follow them blindly and still get some type of end result. Unfortunately modelling is still an art as well as a science, and frequently the end result in such cases is less than adequate. UML assumes you know what you're doing.

While you will understand little of the diagram below showing the possible sequence of events you might go through to develop a dynamic model at this point, what you will notice is that there is a relationship between dynamic modelling and instance/class identification. You can use the instances/classes you have already defined as a basis for dynamic modelling or you can start from scratch, thinking about what happens over time (the dynamic aspects) and ending up with important information that you can feed into any class diagram you may subsequently produce. I think the way round you approach the process depends upon your background.

As with all models, you can develop a dynamic model to describe a proposed system, a current system or help plan improvements. But we are running away with ourselves. Let’s start by considering the easiest to understand diagram – the Use Case diagram.

### A few examples of systems development methods that make use of UML

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Software Development Process - USDP</td>
<td>Bennett et al 2004 p.11</td>
</tr>
<tr>
<td>Guidelines for Rapid Application Engineering – GRAPPLE</td>
<td>(Schumuller 2004 p.253 – 263)</td>
</tr>
<tr>
<td>Zachman Framework: Enterprise Architecture</td>
<td><a href="http://www.zachman.com/">www.zachman.com</a></td>
</tr>
<tr>
<td>Etc. etc</td>
<td></td>
</tr>
</tbody>
</table>

If you have experience with database development you will probably go for developing class diagrams first. On the other hand if you do not have this experience or come from a management background, I think you are much happier thinking about processes – How – first.

As with all models, you can develop a dynamic model to describe a proposed system, a current system or help plan improvements. But we are running away with ourselves. Let’s start by considering the easiest to understand diagram – the Use Case diagram.

### Possible sequence of stages in developing a Dynamic Model

1. **Use Case analysis of various scenarios**
2. **Identify classes**
3. **Develop Sequence diagrams**
   - Looking at different scenarios using objects (from classes)
4. **Consolidate Sequence diagrams**
5. **Develop one or more state diagrams for each class identified**
6. **If appropriate consider how processes can be ‘improved’ (i.e. if doing BPR)**
### 4. Use Case Diagram

Use Case diagrams provide an overview of the model you are developing. They show the main processes (‘use cases’) along with the things (‘actors’) that may interact with them.

There are numerous sites providing some interesting tutorials on UML (but make sure it is UML 2 you are looking at), and as an example I have included a section below from Practical UML: A Hands-On Introduction for Developers by Randy Miller ([http://edn.embarcadero.com/article/31863](http://edn.embarcadero.com/article/31863) - this site may no longer be active as the article was written in 2005). I hope you like the medical nature of the example. Whereas the other UML diagrams have changed since this time the use case diagram has remained stable.

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how.

Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system, in this example the system is a medical clinic.

"A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot."

A use case is a summary of scenarios for a single task or goal. An actor is who or what initiates the events involved in that task. Actors are simply roles that people or another system plays. The picture below is a Make Appointment use case for the medical clinic system. The actor is a Patient. The connection between actor and use case is a communication association (or communication for short).

Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. We've put Make Appointment as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.

Use case diagrams are helpful in three areas.

- **determining features (requirements)**. New use cases often generate new requirements as the system is analyzed and the design takes shape.
- **communicating with clients**. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- **generating test cases**. The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.
Digging Deeper: Use Case Diagrams

Use case diagrams give an outsider’s view of a system. Every use case diagram has actors, use cases, and communications. A simple use case diagram can be expanded with additional features to display more information.

This page covers the following UML™ use case features.

- system boundaries
- generalizations
- includes
- extensions

Medical clinic diagram, expanded

The following use case diagram expands the original medical clinic diagram with additional features.

A system boundary rectangle separates the clinic system from the external actors.

A use case generalization shows that one use case is simply a special kind of another. Pay Bill is a supertype use case and Bill Insurance is the subtype. You can always replace a subtype by the supertype whenever necessary. Generalization appears as a line with a triangular arrow head toward the parent use case.

Include relationships factor use cases into additional ones. Includes are especially helpful when the same use case can be factored out of two different use cases. Both Make Appointment and Request Medication include Check Patient Record as a subtask. In the diagram, include notation is a dotted line beginning at base use case ending with an arrow pointing to the include use case. The dotted line is labeled <<include>>.

An extend relationship indicates that one use case is a variation of another. Extend notation is a dotted line, labeled <<extend>>, and with an arrow toward the supertype case. The extension point, which determines when the extended case is appropriate, is written inside the supertype case.

End of extract
The above example may have left you with the impression that actors are people; however this is not necessarily the case. Let's consider some examples:

<table>
<thead>
<tr>
<th>System you're trying to model</th>
<th>Possible non person based Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>You are modelling the company project management department, and discover it has links with the main accountancy system for the firm</td>
<td><img src="image1" alt="Diagram" /> main accountancy system</td>
</tr>
<tr>
<td>A departmental clinical system where you wish for it to have links to the internet</td>
<td><img src="image2" alt="Diagram" /> The internet</td>
</tr>
<tr>
<td>A purchasing system within a store where clients can pay by credit card as well as in cash</td>
<td><img src="image3" alt="Diagram" /> credit card system</td>
</tr>
</tbody>
</table>

Also from the above use case examples you may have been thinking that the ‘actors’ could possibly be classes and the various ‘use cases’ in the Use Case Diagram (sorry about the terminology) could possibly be operations in particular classes. This is partly correct; it is better to think of them as candidate classes or candidate operations, only after further thought will you be sure. One thing you will find is that several actors may map onto a single class; this is because an actor describes a particular role. Take for example the above use case diagram; both the scheduler and clerk roles may well be amalgamated into the class secretary, with a title attribute indicating what job she/he actually does.

The term actor is rather misleading, and according to Fowler 2004 the term should be ‘role’. The problem was that the word was mistranslated from the Swedish (p100). One actor can be a specialised role of another actor, for example phlebotomist might be a specialised role of nurse:

The term scenario is rather nebulous. It is best to think of it as being a sequence of steps describing the interaction between actor(s) and one or more use cases in the Use Case diagram. In fact most case tools require you to provide a behaviour specification for each use case, which is basically a list of steps that occur within the particular use case. For example the use case Make Appointment could contain the following steps (behaviour specification):

1. The patient calls the clinic.
2. The patient gives the scheduler ID details for verification.
3. The scheduler checks validity of patient.
4. The scheduler checks system to see if a regular appointment time is planned.
5. The scheduler checks system for next available empty slot, depending on the type of patient.
6. The scheduler discusses options with the patient.
7. The scheduler books an appointment.
8. The scheduler logs the call in the patient’s records.
You can also organise a number of use cases into a group by using the UML package element.

UML 2.0 has introduced several new aspects to UML diagrams, one being the possibility of introducing conditions into the use case. The condition is shown as a comment on the diagram with a line from it to the relevant part of the extend line. For example suppose that employees of the company get their health care bills paid for them, otherwise the insurance company pays.

Several of the above examples are rather complex, including packages and extension conditions, and I feel the inclusion of all this complexity rather defeats the object of these diagrams. They are meant as simple overviews of what is happening but including this type of complexity means there is a danger that you will do all the modelling at the use case stage. I think the success of a Use Case Diagram is by remembering:

**Keep it simple**

There are UML specific Sequence and State [machine] Diagrams for modelling the detailed dynamic complexities and we will start this process by revisiting the dynamic aspects of classes / instances.
5. What is the Dynamic Aspect?

'Well, here's another nice state you've gotten me into.' (Stan Laurel 1890 - 1965 mis-quoted)

Review of class instance syntax in UML (UML 2.4 superstructure page 85)

The top compartment shows the name of the instance and its class. The instance name should start in lower case and have no spaces. You should show the class name of which it is an instance using the syntax: instance_name : class_name. All should be unlined.

The second compartment shows the attributes for the class as a list.

The current value of the attribute should be shown using the syntax: attribute_name : type = value

Each row is called a slot.

Most of the above is optional. The minimal amount of detail is the instancename : class_name.

Example of various ways of showing a object in UML (2.4)

<table>
<thead>
<tr>
<th>smith:Person</th>
<th>or</th>
<th>smith:Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>id=478678</td>
<td>or</td>
<td></td>
</tr>
<tr>
<td>firstName=John</td>
<td></td>
<td></td>
</tr>
<tr>
<td>age=45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>developmentalStage=2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hairColour=gray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>or</td>
<td>:Person</td>
</tr>
</tbody>
</table>

Operations not shown in instances as they do not vary between them (Blaha & Rumbaugh 2005 p25)

Note the underlining

Called an ‘anonymous’ instance. Notice the :

It is important to realise that when discussing the dynamic aspect, we tend to focus at the instance (often called the instance level) rather than the class level. This is because we tend to discuss real world examples rather than a set of possibilities. For example rather than considering all the possible interactions between PATIENT and DOCTOR classes, we would rather consider specific interactions between several instances, for example Patient:John_Doe and Doctor:Smyth. This is particularly true at the start of the modelling process; only later on a great deal of consolidation takes place to incorporate the different possibilities, highlighted by analysing the numerous scenarios etc.

Dynamic modelling initially focuses on analysis at the instance (object) level

Because of this, from now on in this document we will assume that we are working at the instance level unless specifically indicated otherwise.
The dynamic aspect of an instance can be thought of as that aspect which is related to time. Basically only three things can happen to an instance over time:

- It can receive communications (messages) from other instances or itself.
- It can send messages generated by its operations.
- It can change its state.

In other words, the dynamic aspect is concerned with messages and states over time (see the diagram opposite).

Consider this. A particular instance of person, called smith, receives a message from another instance (or from itself). Consequently this message may affect this instance or not depending upon the message and the state of the receiving (target) instance. For example, the message may be 'grow up' which might change the value of the data item 'developmental age' depending upon the 'age' of the instance. Other messages may not depend upon the various values of data items within the target instance. One such message might simply be 'kill' which would have the same affect on the target instance regardless of its state.

Because messages often act as external stimuli on one or more instances they are usually called events. These events then often change the state of the instance(s) receiving them.

It should be noted that an instance can send a message to itself.

We will now begin by looking at the UML diagram that focuses on the interactions between instances.

### Exercise 1. Defining Operations

**Time:** 30 minutes maximum

1. Define some of the operations (i.e. activities) that you carry out at work. (Remember work can be paid or unpaid such as volunteering!) What, if any, messages might be required to act as a stimuli to these operations? Also define the class(es) that might be the originators and targets of such messages?

2. Define some of the activities of a community nurse and GP. What, if any, messages might be required to act as stimuli to these activities? What classes might be the originators and targets of such messages?
6. Sequence Diagrams (SD)

A sequence diagram usually describes a subset of interactions (i.e. which I call an instance scenario) between instances by focusing on the sequence of Messages that are exchanged between them.

The basic idea of a Sequence Diagram has been around for some time, and as the UML reference manual (ver 1.3 p306) stated, "Much of this notation is drawn directly from the Object Message Sequence Chart (POSA) notation of Buschmann, Meunier, Rohnert, Sommerlad, and Stal (1996), which is itself derived with modifications from the Message Sequence Chart notation." Jacobson 1995 calls them Interaction diagrams (p132). Adapted from UML 2.4 superstructure (p.517).

A sequence diagram is made up of two main components messages and lifelines. Let’s begin by taking a simple example describing the interaction that might take place between particular instances of a patient, GP and a nurse.

In the diagram the three vertical lines (called lifelines) represent the three instances in a particular scenario which would be part of a model under study: Abdul Hussein is an instance of the class patient, Dr Amy Smith is an instance of the class GP and Nurse Derek Little is an instance of the class nurse. Instances are used instead of the more abstract classes because we are concerned with a particular scenario not general behaviour at this point in the modelling. There is no significance to the horizontal ordering of the objects. Most case tools allow you to manipulate the order to obtain the most aesthetic result.

There are several types of message the simplest being shown below, this type of message (indicated by a open arrow) has a name and number (this is not obligatory) and passes from one instance (the sender) to another (the target). In UML this is called a asynchronous signal message. Time moves down the diagram, but the exact vertical distance between arrows is not important. However, you can show a time period between events as demonstrated in the above. Also various labels (such as timing constraints, descriptions of actions during an activation, and so on) can be shown either in the margin or near the messages or activations (the time period a particular instance is active (the thicker part of the lifeline).

You can show the data that passes with the message by including it in parentheses e.g. Followup_instructions(leaflet_id) or give_instructions(urine_sample, sputum_sample, faeces), each of the data items is separated by a comma, technically each is called a parameter and represents an attribute.

One interesting thing to note is that because we are dealing with instances you can show messages between two instances of the same class. For example if you wanted to show two doctors conferring you might have Amy Smith:Doctor and Steve Wright:Doctor as two instances.

The diagram is a very simple example and might pass as a rough draft. One would expect a much more complex diagram to develop over time using a wider range of UML symbols. Also you would not have the labels explaining the various parts of the diagram in the final version. We will now look at developing this diagram by way of considering the various elements of a Sequence Diagram.

The naming of asynchronous signal messages as shown in the above diagram represent what is happening from the senders' perspective. We will come across other types of messages later on where the perspective is the target instance. Often in models concerned with screen design a asynchronous signal message represents literally a message that might appear on a screen so a message might be msg("please enter pin") where msg is standard shorthand for message.
6.1 Frames

The Frame concept was introduced in UML 2.0. Each Sequence Diagram should now be within a frame. Within the left hand corner of the frame there is a title along with the keyword **SD** (the casetool I have used for this chapter uses the term **Interaction** instead) indicating that it is a Sequence Diagram. You will notice this feature on the sequence diagram on the previous page. You may think this is rather trivial at the moment, but later on it will become clear how such a simple addition to the semantics of the diagram can allow you to model more complexity.

6.2 Lifelines

Each lifeline, a dashed vertical line with a rectangle at the top indicating the instance name and class, participates in one or more interactions in the SD. The length of the lifeline represents the existence of the instance. If the instance is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point; otherwise, it goes from the top to the bottom of the diagram. An instance symbol is drawn at the head of the lifeline.

If the instance is created during the scenario depicted in the SD, then the arrow, which maps onto the stimulus that creates the instance, is drawn with its arrowhead on the instance symbol. If the instance is destroyed during the diagram, then its destruction is marked by a large “X,” (the stop symbol) either at the arrow mapping to the stimulus that causes the destruction or (in the case of self-destruction) at the final return arrow from the destroyed instance.

An instance that exists when the transaction starts is shown at the top of the diagram (above the first arrow), while an instance that exists when the transaction finishes has its lifeline continue beyond the final arrow.

Enough of the theory. Let’s consider an example. Imagine that you’re in Star Trek working on the Enterprise, and Nurse Derek Little is a hologram which you can create and destroy at will. You would represent this by moving the instance name down to where he is created and use the ‘X’ (stop symbol) to show his destruction as.

The lifeline can also possess other characteristics such as a thickened portion showing when it has ‘focus of control’, this is discussed in more detail after we have looked at the various message ‘sorts’.
6.3 The Message

The message is the heart of the SD and its most complex aspect.

All messages can be said to have the following characteristics:

- They happen at a point in time which is usually considered to be instantaneous being triggered by an event.
- A 'time stamp' is passed to the target instance because the event happens at a point in time.
- Usually there is no significant duration for the message but if necessary you can indicate this (slanted lines).
- They often pass information (attributes).

Let's consider an example of an asynchronous signal message:

In the above diagram the patient instances sends an asynchronous signal message (the result of a send event) to a theatre (target) instance providing information about a particular patient, theatre and implicitly time. Looking at the value of the attributes in the theatre instance (the bottom section of the rectangle), it can be seen that in all probability the theatre instance either accepts the patient or rejects the patient, the response by the target instance is the result of the receive event being triggered by the message.

The above depiction is very similar to a particular UML diagram type called a Communication diagram, formally called a Collaboration diagram. I will not be discussing this type of diagram any more in this chapter as it is very similar to the UML Sequence Diagram, which we are now considering. If you are interested in communication diagrams you can find many examples on the web and Wikipedia.

After you have completed the exercise below we will move on to look in depth at the Sequence Diagram.

Exercise 2. Naming asynchronous signal Messages

Time: 15 minutes maximum

Considering the previous exercise, suggest some data items that might be passed in some of the messages you have defined.

Some aspects of the SD have changed significantly with UML 2.0, not all for the good I feel. The diagram now has much greater semantic richness but many of the new approaches are less intuitive. Because of this loss of some features which modellers have found useful, various writers such as Fowler 2004 (p.60) suggest that people develop UML diagrams that are not always completely standard. I would agree with this as long as the non-standard semantics are clearly explained. Let's now consider various aspects of the message.
6.3.1 Message Sorts

So far in this chapter I have shown messages with an open arrowed line but UML provides a number of different sorts of message (defined in UML as a MessageSort enumerated type). I will consider each of the six 'sorts' in turn

6.3.1.1 Asynchronous signal message

This type of message does not require a response and therefore does not require the sending instance to wait for a response. In software applications that allow you to create UML diagrams this is often referred to as a send message and all the messages you have encountered so far are of this variety displaying a number and a name and also possibly a set of attributes (all this information combined is called the message signature).

An example would be a notification message where the sending instance would just carry on with a sequence of actions once the message had been sent. For example, say an instance of father_cook sent an asynchronous signal message: Dinner_ready they would not need to wait around for you to do anything before continuing on with another activity, they might start the activity washing up or even eating!

Because this is the only sort of message that conveys a ‘signal’ it is often simply referred to as a signal. Within the 2.4 superstructure specification this is called a asychSignal sort of message.

6.3.1.2 Asynchronous call message

Again this sort of message does not require a response and therefore does not require the sending instance to wait. However this time the message carries information about a particular method that will be triggered by the target instance. Hence in the above diagram we now have an operation in the Theatre instance called operate_on which is triggered by the message.

Within the 2.4 superstructure specification this is called a asychCall sort of message.

6.3.1.3 Synchronous call message

This message sort is very similar to the asychCall sort except this time the message requires a response and therefore does not require the sending instance to continue until one is received. For example the Dinner_ready message sent from you, would stop you doing anything until the resulting activity of the instance who received the message completed the action evoked by that message (i.e. come and sit at the table). Synchronous messages are also called procedural messages. This type of message is associated with a reply message.

Similarly in the operating theatre scenario shown above if the message were a synchronous call message with the operate_on as the operation as part of the message this would mean that the Patient instance would be frozen until the target instance responded to the operate_on message via the operate_on operation.

6.3.1.4 Reply message

This sort of message informs the sending instance that the target instance returns control to the sending instance in a synchronous call message. Often it is omitted from the SD. A typical example is where the call message is get_something and the reply message is something.

Important:

Both the asynchronous and synchronous Call messages have operation names attached to them that are from the target's perspective. In contrast the asynchronous signal message name is from the sender's perspective.
6.3.1.5 Create message

The create message triggers the creation of a instance of the target instance, in the uml superstructure specification 2.4 the term only occurs twice and there is no discussion concerning the indicating of parameters on the message, in fact this is not possible in Magicdraw (but possible in the IBM case tool), once you add a operation/parameters to the message it changes to a call message. A workaround is to still use the create message and then also have a initialisation message (of one of the call sorts) after it.

6.3.1.6 Delete message

A delete message works much like a create message but this time destroys the target instance. In medical systems often instances are delete flagged (i.e. an attribute is set to deleted or archived instead of active) rather than destroyed.

6.3.2 Execution specification

The UML superstructure specification is difficult to follow to say the least and several people have tried to create a more user friendly interface to it, Schaum's UML by Bennett, Skelton and Lunn is a paper example whereas a web based example is Kirill Fakhrouddinov's excellent website (www.uml-diagrams.org). This is what he has to say about execution specifications.

Execution (full name - execution specification, informally called activation) is an interaction fragment which represents a period in the participant's lifetime when it is:

- executing a unit of behavior or action within the lifeline,
- sending a signal to another participant,
- waiting for a reply message from another participant.

Note, that the execution specification includes the cases when behavior is not active, but just waiting for reply. The duration of an execution is represented by two execution occurrences - the start occurrence and the finish occurrence.

Execution is represented as a thin grey or white rectangle on the lifeline. [end of quote]

I tend to ignore this specific semantic enrichment, so now let's move on to another aspect of the message in a SD.

6.3.3 Slanted Arrows

In a Sequence Diagram usually the message arrows are laid out horizontally; however if you want to indicate that a particular message may take some time to complete its task the arrow may be slanted downward so that the arrowhead is below the arrow tail.

The SD opposite (Tsang, Lau & Leung 2005 p.162 simplified) shows the use of such lines.
6.3.4 Parameters, Arguments and return values

The example a couple of pages back showing the operating theatre situation has the expression:
Operate_on(patient_ID, Theatre_ID) above the message. We will now concentrate on the expression in the brackets (parentheses) which can represent two distinct concepts:

- Parameters
- Arguments

**Parameters** – the patient_ID and Theatre_ID represent values based to the target instance operation Operate_on, however you may have noticed that these values are rather generic, they are equivalent to the class rather than instance level, yet aren’t we working at the instance level? This is an ambiguity in UML and it appears that it is legal, in UML SD’s to specify parameters. However I feel that whenever possible it is best to express them at the instance level using the argument concept.

**Arguments** - This is just the instance equivalent to the parameter value. Taking our example we show this:
(patient_ID=0293, Theatre_ID=003) 0293 and 003 are the arguments associated with the two parameters.

Often operations return valuable information and the point of the message is to interrogate the target instance in some way to find out something via the return message, this is indicated by reply message return values.

**Return values** - Over the various versions of UML the format for the reply message has changed many times. Now it seems to me to be rather tortuous, after all, all you want to show is the return value(s). Taking our example, I have now given the doctor class a method, start_examination that has a return value called message:

The argument for the reply message for the start_examination operation in the above is "no problems" this is saying that the message parameter value, has a argument equal to "no problems". I also provide an example of an operation that does not return anything other than a completion signal, you can either leave the reply message blank or add the operation name again omitting the parameter/argument values by using the dash – character.

In previous versions of uml you could simply put message="no problems" or return "no problems" this seems to make more sense to me and I will carry on using this old fashioned method.
6.3.5 Summary

This ends the description of the message as defined in UML2.4 as used in SD’s.

Ignoring the reply message, from the above you may have surmised that the actual name of the message is probably going to be either the operation to be called by the target instance or the message (i.e. signal) name, as is the case.

In effect the message can either invoke an operation or send a signal. Fowler 2004 p. 61 is again wary of all this complexity and suggests that people should use any sort of message other than the asynchronous signal variety with extreme care.

The diagram opposite shows the types of message that one can create using Magicdraw, interestingly the simple asynchronous signal message is the third one down and not the default message sort, probably because the uml 2.4 superstructure (p.507) says that the default sort of message is the synchronous variety.

Gradual refinement of messages

Tsang, Lau & Leung 2005 (p.152) along with other authors suggest that the first few models would use only simple asynchronous signal messages which in subsequent models some get transformed into call (synchronous and asynchronous), create and destroy messages. Obviously some may stay as simple asynchronous signal messages.

In the above I also presented details of parameters and arguments and return values, again I would only consider these once the model has gone through a number of iterations, but obviously there are exceptions to this as a particular scenario might bring the data (i.e. parameter/argument return values) aspect to the fore which screams out to be modeled.

The gradual changing of a SD from one that contains only simple signal messages to one containing a wide variety can be difficult, one problem is that to convert a signal message to a call message means changing a name from an originator to a target perspective, for example the signal message "enter password" between a user and a ATM might be converted to a operation within the ATM called check_password or get_password.

Exercise 3. Refining messages

Time: 30 minutes maximum

1. I have considered the various asynchronous signal messages in the GP/Patient/Nurse scenario on page 13 and come up with a list of operations that might be associated with them for just the GP and Patient. These are shown opposite. Using pen and paper try to develop an equivalent SD to the one on page 13 but using mainly the operations in the classes, that is using call messages. Do not worry about showing the reply messages or parameter values etc.

If you think there are any other operations necessary to mimic the original scenario between GP patient please add them as well.
6.4 Showing More Detail – Interaction Use

UML 2.4 introduced the concept of ‘InteractionUse’ – this indicates a number of interactions which occur in another diagram. This is indicated by using a frame which is drawn to cover the lifelines involved in the interactionUse and you can define parameters as well as a return value. This is excellent because it means that you can move detail to several smaller Sequence Diagrams, each representing an interaction fragment, and reuse these smaller components. Considering our example:

Such Interaction use constructs allow the modelling of complex scenarios in a number of clearly defined stages, for example if you wanted to model a whole project you would start by possibly creating a overview SD which just contained a number of Interaction uses called something like preliminary_setup, start, monitor, evaluate, foldup.

You will notice that in the above example the interaction use takes a parameter/argument, the UML 2.4 talks about arguments but the diagram example appears to show parameter values!

Decomposition is an important aspect of the modelling process – and interaction uses allow us to do this very effectively – a extremely complex scenario can be broken down into many interaction uses each of which can contain more interaction uses.

Interaction uses also have other uses including specifying how interactions are controlled, which we will now consider.
6.4.1 Gates

Quoting UML superstructure 2.4. page 494, "A Gate is a connection point for relating a Message outside an InteractionFragment with a Message inside the InteractionFragment." The ends of the messages going to and from the interaction Use (i.e. the ref box) are called **Formal gates** and the message ends within the interactionUse at its margin are called **actual gates**. Obviously the number and type of message must match up between the formal and actual gates.

Lets take an example, this time from the Magicdraw user manual version 17 (2011 p. 693) In the left hand SD, the getBalance goes to the interaction Use Balance lookup. Both the getBalance() synchronous call message along with its reply message therefore have a formal gate.

Considering the actual **Balance Lookup** interaction use the getBalance message has an actual gate and automatically repeats the data of the getBalance message from the untitled SD.

In our GP scenario gates might be used for the Nurse consultation interaction use with a message something like go_to_nurse with the parameter value patient_name. The sd opposite shows this and the sd referencing it on page 13. In some ways this produces, in terms of model equivalence the same as adding a parameter value to the actual interaction use.

Reflecting on the bank balance example above it seems rather confused with both the same message parameter and interaction use parameter and would certainly be refined in subsequent models.
6.5 Control

Let’s think how the flow of control in a SD might be affected:

- We might want a particular fragment to be repeated until a certain condition is achieved = **loop**
- We might want a particular fragment to take place only if a specific condition is met = **opt**
- We might want alternative fragments to take place depending upon a particular condition = **alt**
- We might wish several fragments to go on at once = **par**

Each of the highlighted words in the above sentences indicates the keyword that UML uses to specify the particular condition you are modelling. After uml2.0 this is modeled using a concept called a **Combined Fragment**. Let’s look at each of these in turn.

6.5.1 Iteration

In a old style SD we might have the message *[for all samples]:supply_sample. Indicating that we want something to repeat until all the samples had been supplied. This is an example of an iteration; this type of Combined Fragment repeats until a certain condition is achieved.

Iteration is just another name for repetition / looping; it indicates that a particular message repeats a certain number of times. The number may be specified or be related to some condition or even initially be unspecified.

How we show this in a SD has changed since UML 2.0. Because the two methods are so different and most UML practitioners occasionally still use the older method, I will provide details of both.

Pre UML 2.0

An iteration is represented by an asterisk (*) followed by an expression in square brackets followed by the message name. For example:

* [While items in stock] : buy
* [For each item] : catalogue
* [While not unconscious] : drink
* [For all samples] : supply sample (as in the above Sequence Diagrams)

*[While hospital not full] : admit
*i := 1..n : send message (taken from the UML specification v1.4 p3.133)

If you are not sure of the iteration condition you can miss it out and just have the asterisk followed by the message name e.g. * admit, * update etc. The UML v1.4 specification is unclear about the use of the asterisk, mentioning it in the narrative v1.4 p3-133 and expression example on p2-134 but not showing it in the example Sequence Diagram on p3-104.

UML 2.0 onwards

This uses the Combined Fragment concept basically a number of messages and lifelines enclosed in a box, showing the keyword (called a interactionOperator in uml 2.4) **loop** in this instance along with the condition at the top. An example is given opposite.

Obviously this example is trivial with just two messages, but you can easily see how you can specify more.
6.5.2 Single Branching

Suppose in our example we wish to show that given a particular condition (called a guard condition or constraint), a certain group of messages is executed. Again there is the old and new way of doing it. To be more specific, we only want the interaction fragment to be executed if the guard condition evaluates to true.

Pre UML 2.0

The guard condition is represented as an expression within square brackets followed by the message name that will take place when the expression is true. As in the case with iteration, the expression and message would appear on the appropriate message.

Here are some examples:

- \[[\text{patient weepy}]\] : give reassurance
- \[[\text{product available=yes}]\] : buy it
- \[[\text{Finished your vegetables}]\] : leave the table
- \[[\text{balance>0}]\] : borrow money

In UML pre 2.0, message lines could divide to indicate conditional branching.

UML 2.0

This time the opt keyword is used and all the other aspects mentioned above for the loop situation also apply here. Imagine that you feel you are giving some patients too much time and really only want to reassure those that you feel really need it, such as those who appear a bit weepy:

6.5.3 Multiple Branching

Another more complex type of branching only exists in UML 2.0. This is where you want to consider the possibility of several options at once. Taking the above example (the consultation scenario) one step further assume we have replaced the opt Combined Fragment above, instead you want to consider three alternative possibilities for your patients: if very weepy, if anxious and those who are no bother to you whom you can just give a paper leaflet explaining what to do.

The frame now has three compartments, each with an expression called an interaction operand presenting a choice (i.e. a type of guard condition), the frame is again called a combined fragment but this time with the alt keyword which means there is also a else compartment and as many other choices (i.e. operands) as you want. As usual it’s more important to be able to use these constructs correctly rather than just know the name of them.

6.5.4 Concurrent Messages

Concurrent messages are the last type of message I want to describe. In this situation you want to be able to model the situation where more than one thing is happening at once. UML 2.0+ allows you to do this by once again using a combined fragment but this time with the par keyword. An example should suffice. Assume now that holographic nurse Derek can carry out several activities at once with the patient.

It should be noted that the SD tells us nothing of the timing of each of these operands, only that they will run concurrently; we do not know which, if any, may finish first etc. Uml has added a large number of additional key words to model more complex concurrent situations but beyond the material covered in this chapter.
6.6 Nesting

So far we have not seen one interaction fragment within another, but this is quite possible. See the SD opposite.

I’m sure you will be pleased to know that you have now learned the basics of Sequence Diagrams, but the important thing is to get as much practice as possible. To help you achieve this I have provided a number of exercises on the next few pages. Just a quick word of advice first.

6.7 Incremental Development

As discussed in the previous section concerning message refinement, incremental development in SD’s is essential and also applies to the whole process of modelling, the modeller gradually develops a number of scenarios considering first how the system would behave in normal conditions and then when something untoward might happen (e.g. the patient passes out or the nurse is not available). Finally the modeller may consolidate them by comparing and contrasting each scenario. UML provides a special diagram to allow you to amalgamate them without getting too much clutter: the Interaction Overview Diagram.

However I feel that you can achieve a similar result by making use of the ref interaction use as we did earlier. Now onto some practice for you.
6.8 Exercises - Sequence Diagrams

Exercise 4. Improving Sequence Diagrams

Below are given some draft Sequence Diagrams. See how many errors you can spot and make as many improvements as you can to make them into UML 2.0+ compliant Sequence Diagrams.

A. A Restaurant

```
<table>
<thead>
<tr>
<th>waiter</th>
<th>customer</th>
<th>kitchen</th>
</tr>
</thead>
<tbody>
<tr>
<td>bring menu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discuss options/specials</td>
<td></td>
<td></td>
</tr>
<tr>
<td>read menu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place order verbally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write down order and numbers of items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>place order list and number of items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prepare food</td>
<td></td>
<td></td>
</tr>
<tr>
<td>food ready to be served</td>
<td></td>
<td></td>
</tr>
<tr>
<td>food is served</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
### B. Organising a Local Medical Conference

Another rough draft for you to work on.

<table>
<thead>
<tr>
<th>Delegates</th>
<th>Speakers</th>
<th>Organisers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form organising committee</td>
<td>Select possible speakers</td>
<td>Select and reserve venue</td>
</tr>
<tr>
<td>Decide on topics</td>
<td>Invite speakers</td>
<td></td>
</tr>
<tr>
<td>Send out first announcement</td>
<td>Send out second announcement and registration</td>
<td></td>
</tr>
<tr>
<td>Collect replies of interest from public</td>
<td>Collect registrations</td>
<td></td>
</tr>
<tr>
<td>Make travel accommodation arrangements</td>
<td>Hire venue and AV equipment</td>
<td></td>
</tr>
</tbody>
</table>
C. Carrying Out Cerebral Angiography (only attempt this one if you are a medical doctor or nurse who might want to annotate it significantly!)

<table>
<thead>
<tr>
<th>Action</th>
<th>Radiologist</th>
<th>Neurosurgeon</th>
<th>Patient</th>
<th>Nurse</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive request for angiography</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check indication with neurosurgeon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>review patient’s CT/MR scan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>introduce self to patient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explain procedure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reassure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>obtained informed consent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check BP &lt;160 systolic &lt;120 diastolic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check INR is 1.2 or&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV drip</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>order diazepam 10mg orally 2hrs prior</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>book anaesthetist</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>book angiography suite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check on patient in ward 2hrs prior to procedure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perform 4 vessel angiogram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read angiogram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>report angiogram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discuss report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check puncture site for 6 hours in ward</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check BP and pulse for 6 hours post procedure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many thanks to past students for providing the basis for the above examples.
D. A Published Example (see how many errors you can spot and think of ways to possibly improve it)

From: http://www.visual-paradigm.com/product/vpuml/htmltutorial.jsp
Exercise 5. Developing your Own Sequence Diagrams

Time: 60 minutes maximum

1. Devise a Sequence Diagram for two situations (scenarios) involving yourself at work.

2. Devise a Sequence Diagram for one of the following situations:
   1. Planning a 6 month holiday
   2. Building or renovating something
   3. Items of service claims forms for GPs
   4. Clinical audit within a hospital department
   5. Multidisciplinary assessment in the community
   6. Commissioning negotiation
   7. Accident and Emergency assessment
   8. Care programme approach review meeting in psychiatry

3. Consider one of the scenarios in the modelling scenarios document at http://www.robin-beaumont.co.uk/virtualclassroom/chap11/s0/modelling_scenarios.pdf

7. Business Process Re-engineering (BPR)

This very impressive term is really nothing more than analysing an organisation using – amongst other techniques – Sequence Diagrams. But as would be expected, they are given numerous other names in BPR. BPR uses the Sequence Diagram concept to consider how the events in an organisation might be re-organised for more efficient and/or effective behaviour. I feel an example coming on:

A particular health authority are running a project called 'heartrisk'. This is a GP community based project which provides data concerning individual patient risk of coronary heart disease based upon measuring various risk factors (smoking, drinking, weight etc.) to patients in the form of a hand-held sheet. At the present time a representative visits practices they feel might be interested in participating in the project. If they are interested, a representative from the external software company is informed who in turn informs a colleague who visits the GPs and sets up the GP system to collect the relevant data. A representative of the software company or the health authority then visits the site to train them. A further visit by a member of the software firm collects a copy of the GP system backup tapes to obtain the necessary data after a suitable time period. Following initial data analysis by the software firm in consultation with the practice, a ranked list of each patient's CHD risk score is printed out and sent to the practice. The practice then decides who would most benefit from the leaflets and requests them from the software firm. These are then posted out to the practice which contacts the patients to offer them advice if they haven't died already!

The options available to improve this scenario are numerous. Let's consider the one simple strategy of allowing the GPs' own system to do what the external software firm now do to a limited extent. The new event trace is given below. It immediately looks much less complex. Event traces are extremely useful for analysing situations such as the one above.
Dynamic Modelling and Process Re-engineering using UML
Part 1 – Use case and Sequence diagrams

(Business) Process Re-engineering is a very important aspect of modelling which is frequently ignored by people not involved in management. This is a shame as such people can bring a degree of analytical rigor which is often lacking. For more information about BPR see the links section in this chapter.

Interestingly the NHS some while ago produced a document about BPR: “Management of Change: An Overview of Management Change Methodologies” (NHS E 1997 ref no.D4054). Unfortunately the document is very thin on information and basically just provides a list of methodologies.

Many management companies have developed methods that make use of BPR, Lean and six sigma are two popular approaches I have included below a abstract from the Wikipedia entry for six sigma.

Beginning of extract
Six Sigma projects follow two project methodologies inspired by Deming’s Plan-Do-Check-Act Cycle. These methodologies, composed of five phases each, bear the acronyms DMAIC and DMADV.

DMAIC is used for projects aimed at improving an existing business process. DMAIC is pronounced as "duh-may-ick".

DMADV is used for projects aimed at creating new product or process designs. DMADV is pronounced as "duh-mad-vee".

DMAIC
The DMAIC project methodology has five phases:

- **Define** the problem, the voice of the customer, and the project goals, specifically.
- **Measure** key aspects of the current process and collect relevant data.
- **Analyze** the data to investigate and verify cause-and-effect relationships. Determine what the relationships are, and attempt to ensure that all factors have been considered. Seek out root cause of the defect under investigation.
- **Improve** or optimize the current process based upon data analysis using techniques such as design of experiments, poka yoke or mistake proofing, and standard work to create a new, future state process. Set up pilot runs to establish process capability.
- **Control** the future state process to ensure that any deviations from target are corrected before they result in defects. Implement control systems such as statistical process control, production boards, and visual workplaces, and continuously monitor the process.

DMADV or DFSS
The DMADV project methodology, also known as DFSS ("Design For Six Sigma"), features five phases:

- **Define** design goals that are consistent with customer demands and the enterprise strategy.
- **Measure** and identify CTQs (characteristics that are Critical To Quality), product capabilities, production process capability, and risks.
- **Analyze** to develop and design alternatives, create a high-level design and evaluate design capability to select the best design.
- **Design** details, optimize the design, and plan for design verification. This phase may require simulations.
- **Verify** the design, set up pilot runs, implement the production process and hand it over to the process owner(s)

Often people develop sequence diagrams in isolation but it is extremely important to realise that they are just one aspect of the overall model as they relate to all the other UML diagrams and this must always be in the modellers mind we will now consider this in more detail.

End of extract
8. Relationship between Sequence Diagrams, Use Cases and Class Diagrams

We have mentioned that there is a correspondence between the instances shown in the Sequence Diagram and the classes in the class diagram. Furthermore concerning call messages they map directly to particular operations in the target class (instance – in the SD).

However, let’s now consider the simple asynchronous trigger messages, although these do not directly relate to operations they may suggest operations in subsequent more refined SD’s where some of the messages may be remodelled into call messages.

8.1 Asynchronous trigger Messages

You will notice that the possible operations I suggest in the above classes relate to the arrow source (the opposite to call messages).

On the next page I attempt to consolidate the asynchronous signal messages above along with the operations I have suggested.\side
8.2 Call Messages

Having considered the SD on the previous page I have now refined it as shown below with call messages and gates.

8.3 Use Cases

The relationship between Actors and aspects of the SD are a bit more problematic. Frequently they represent several instances of one particular class, for example various roles a nurse might undertake. This indicates that several actors may become one class. The individual use cases themselves occasionally become operations, but they are usually too high level and require further refinement to map to low level operations. Tsang, Lau and Leung 2005 p.191 provide a little more detail, indicating that the single use case specification (i.e. the list of steps within a single use case) can be mapped to a series of messages between instances in a Sequence Diagram which then themselves become operations in various classes. Obviously there is nothing stopping you defining operations in a class and then making use of them in a SD – which way round you do it is up to you. I tend to ignore the Use Case diagram after producing a very simple overview. Basically we have:

Actors -> candidate Classes
Steps within a use case or sometimes actual individual use case -> call Message in SD -> operation in target class.
The diagram below indicates the relationship I have described above. Now you possibly can see why it is advantageous to use a case tool which keeps all the diagrams synchronised rather than a simple drawing tool.

---

**Exercise 6. Relationship between Sequence and Class Diagrams**

**Time: 20 minutes maximum**

Consider one of the Sequence Diagrams you have drawn and develop a class diagram from it. Make sure you include operations for the various classes derived from the messages in the SD.
9. Instance scenarios

As with other UML diagrams you can develop the SD from a narrative, but because the SD is at the instance level and often the initial text is usually at the class level it needs to be transformed. An example should help.

Original narrative (implicitly at the class level)
A doctor calls in a patient who then welcomes the patient when they arrive at the consulting room. The patient acknowledges the welcome then the doctor begins asking questions which are answered by the doctor. The doctor then usually reassures the patient and then sends them off for the nurse consultation. Etc.

Transformation into an instance scenario
Amy Smith (instance of doctor) calls for AbdulHussein (instance of patient).
Amy Smith (instance of doctor) welcomes AbdulHussein (instance of patient).
AbdulHussein (instance of patient) acknowledges the welcome from Amy Smith (instance of doctor).
Amy Smith (instance of doctor) requests information from AbdulHussein (instance of patient).
AbdulHussein (instance of patient) provides information to Amy Smith (instance of doctor).
Amy Smith (instance of doctor) reassures AbdulHussein (instance of patient).
Amy Smith (instance of doctor) sends AbdulHussein (instance of patient) off to see . . . . . etc..

Often it is a good idea to show the instance scenario in the form of a table as shown below:

<table>
<thead>
<tr>
<th>Message id</th>
<th>Message type</th>
<th>originator</th>
<th>target</th>
<th>Name/operation call</th>
<th>comment</th>
<th>etc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>signal</td>
<td>Amy Smith (instance of doctor)</td>
<td>AbdulHussein (instance of patient)</td>
<td>Calls_in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>signal</td>
<td>Amy Smith (instance of doctor)</td>
<td>AbdulHussein (instance of patient)</td>
<td>welcomes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>signal</td>
<td>AbdulHussein (instance of patient)</td>
<td>Amy Smith (instance of doctor)</td>
<td>acknowledges</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Some UML case tools allow you to create the SD via a table. The example opposite, taken from the VP UML case online tutorial, is typical.
10. Importance of Dynamic Modelling and testing

In the above sections you have learned a lot about dynamic modelling and specifically the sequence diagram and I’m sure now you must have a headache!

What is so amazing is that dynamic modelling has often been ignored in health situations in the past. However it is becoming more and more important as the realisation that understanding how people work and how this will change with the greater use of computers is still poorly understood in the health service.

One area where dynamic modelling is being applied with more and more success is that of specifying 'business processes' within an organisation. The use of this technique was clearly demonstrated in the example concerning the 'heartrisk' project. As mentioned at the beginning of this chapter, in 1997 the NHS Executive produced a book on Business Process Reengineering in the NHS. Unfortunately the book contains little in the way of practical advice. In fact, one feels it is trying to encourage the use of external consultants rather than encouraging people to learn the skills themselves.

Dynamic models are also used to describe computer interfaces. Each screen or window is represented as a state while the event between each state is the various mouse clicks or key presses required. This aspect is covered in a separate document which you will find at http://www.robin-beaumont.co.uk/virtualclassroom/contents.htm

Finally and possibly most importantly dynamic models require some type of testing procedure, and even the simple room thermostat presents a complex problem when all possible states and transitions are considered. However, luckily various approaches have been developed to help reduce the number of required test scenarios and one company, Testcover.com, provide software along with a process to help, you can see the state machine for the room thermostat at http://testcover.com/pub/thermo.php which is well worth a look.

Exercise 7. Developing Dynamic Models

Time: 120 minutes maximum

1. Consider the DopeHead scenario and draw a Sequence Diagram for a typical blood sample. Details of how to obtain the modelling scenarios document is given in the “Before You Start” section at the beginning of this chapter.

2. Select a particular aspect of your work and draw a Sequence Diagram. Some possibilities might include:
   - Patient admission
   - Patient discharge
   - Transferral to ITU
   - Preparation for an organ transplant
   - An outpatient clinic
   - If you do not work in the health field think up some of your own

   You should then present your findings to your colleagues with suggestions for any enhancements that you think are appropriate. Take note of any comments they make about these proposed enhancements. Did you forget some important characteristics of the system when suggesting them?

3. Ask a friend about a particular scenario. Preferably, choose a profession unrelated to your own.

4. Looking at the modelling scenarios document, choosing one of the scenarios attempt to model its dynamic aspect.
11. **Multiple Choice Questions (MCQs)**

I have included a number of multiple choice questions below to help you revise the material in this chapter. I would advise you to mark beside each one where you found the answer in the chapter.

1. Which of the following best describes the dynamic aspect of system modelling (one correct answer):
   
   a. Investigation of data aspects  
   b. Investigation of temporal (time) aspects  
   c. Investigation of functional aspects  
   d. Investigation of security aspects  
   e. Investigation of general modelling aspects

2. The dynamic aspect of classes/instances is reflected in their (one correct answer):
   
   a. Data values  
   b. Operations  
   c. Attributes  
   d. Class names  
   e. Relationships

3. Messages can be considered to be equivalent to (one correct answer):
   
   a. Warnings  
   b. Activities  
   c. Responses  
   d. Triggers  
   e. Communications

4. The process of developing dynamic models involves, amongst other things (one correct answer):
   
   a. Development of Sequence Diagrams based upon identified classes  
   b. Development of Sequence Diagrams from event traces  
   c. Development of classes from event traces  
   d. Development of work improvement plans from BPR  
   e. Development of table definitions from identified classes

5. What is the focus of a sequence diagram (one correct answer):
   
   a. interactions between classes  
   b. interactions between instances  
   c. interactions within an instance  
   d. interactions between instances of a single class  
   e. interactions between classes in the model and external systems

6. A Sequence Diagram is said to represent (one correct answer):
   
   a. A lifeline  
   b. A scenario  
   c. A message board  
   d. A state sequence  
   e. Instance relations
7. What is the name of the above diagram (one correct answer):
   a. Event matrix
   b. Class inter-relationship diagram
   c. Sequence diagram
   d. Message board
   e. State diagram

8. What type of thing does "Label 1" point to in the above diagram? Note that this may be different from the standard approach taken to this type of diagram (one correct answer).
   a. Class name
   b. Data sources
   c. Instance instance
   d. Event source
   e. Actor

9. What is the correct title for 'Label 2' above (one correct answer):
   a. Activity
   b. Message
   c. Warning
   d. Information arrow
   e. Data flow

10. What is the correct title for 'Label 3' above (one correct answer):
    a. Time
    b. Time in hours
    c. Time in days
    d. Scenario boundary
    e. Does not possess a sensible label
11. What is the correct title for 'Label 4' above (one correct answer):

a. Lifeline  
   b. Timeline  
   c. Instanceexistence boundary  
   d. Dataline  
   e. Does not possess a sensible label

12. What is the correct title for 'Label 5' above (one correct answer):

a. Lifeline  
   b. Time period  
   c. Existence constraint  
   d. Association  
   e. Does not possess a sensible label

13. Which statement is true (one correct answer):

a. UML are basically sets of diagramming standards.  
   b. UML are sets of diagramming standards along with rules for software development.  
   c. UML consist of a set of diagramming tools, software development rules and suggested management structures.  
   d. UML focuses on systems management techniques.  
   e. UML provide a small number of diagramming tools but a detailed method for software development.

14. When developing scenarios (one correct answer):

a. Scenarios are developed gradually, encompassing more complex/exceptional circumstances. Eventually an attempt is sometimes made to consolidate them.  
   b. Scenarios are developed for mutually exclusive situations which are eventually joined together.  
   c. Scenarios of simple situations are developed and then consolidated.  
   d. Scenarios are developed gradually, encompassing more complex/exceptional circumstances. Finally a single, all embracing one is produced.  
   e. Scenarios are developed by the system developers to propose enhancements to manual systems.

15. BPR stands for (one correct answer):

a. Business Purpose Realignment  
   b. Business Purpose Redevelopment  
   c. Business Process Reengineering  
   d. Business Process Redevelopment  
   e. Business Process Realignment

16. BPR makes extensive use of which type of diagram (one correct answer):

a. Sequence  
   b. Event  
   c. Dataflow  
   d. Class  
   e. Higraph

17. What is commonly the main purpose of BPR (one correct answer)?

a. To consider how proposed events in an organisation might be organised for maximum efficiency and effectiveness  
   b. To radically reorganise an organisation for maximum efficiency and effectiveness  
   c. To consider how the events in an organisation might be reorganised for more efficient/effective behaviour  
   d. To consider how present or proposed events in an organisation might be reorganised for more efficient/effective behaviour  
   e. To monitor the current efficiency and effectiveness against a theoretical optimal level
Dynamic Modelling and Process Re-engineering using UML
Part 1 – Use case and Sequence diagrams

12. Summary

Dynamic modelling is gaining in importance as it becomes clear that often the problem with computer, as well as human, information systems is how they work and not just the data they produce.

The techniques of dynamic modelling are still relatively in their infancy compared to the traditional entity relationship (ER) modelling and its more modern equivalent, class/instance modelling. This is probably due to a number of factors. Dynamic modelling has been traditionally less used by modellers so has not been so well developed, resulting in a catch 22 situation with fewer people using dynamic modelling because the tools are less useful because few people use them. There is also perhaps the fact that dynamic modelling is conceptually more difficult for the modeller compared to the sometimes almost somnambulant process of defining entities for ERD's or classes for uml.

In this chapter we have barely scratched the surface of dynamic modelling but probably have still gone further than most traditional courses, a worrying prospect given the dire consequences of not considering the dynamic aspect of a system.

The diagram below once again shows the main stages in developing a dynamic model.

Now return to the learning outcomes at the beginning of this chapter and see how many you can tick off. Remember that at the start you were warned that this subject is difficult!

The next chapter looks at the dynamic aspect within a single class.

Possible sequence of stages in developing a Dynamic Model

13. References


Fowler M Scott K 1998 UML distilled: Applying the standard object language. Addison Wesley


NHS Executive 1997 Business Process Reengineering in the NHS. NHS executive UK
Open University 1993 Relational Database Systems M866 (Five books; Introduction to database technology, The Relational Modal, Normalisation, Using SQL, SQL database management) [These are excellent resources with many exercises and clear concise explanations].

Pender T 2003 UML bible. Wiley [uses uml 2]
Pender, A Thomas 2002 UML Weekend crash course: 15 hours. Wiley
Samek M 2009 (2nd ed.) Practical UML statecharts in C/C++ Elsevier [Although this book focuses on programming the dynamic aspect is covered in detail].

14. Links

An excellent set of web pages giving examples and discussion concerning UML: http://www.uml-diagrams.org/use-case-diagrams-examples.html

A Spanish course on uml by Cleidson de Souza http://www.ufpa.br/cdesouza/teaching.html

BPR Links:

A special issue of the Journal of Management Information Systems (Jmis) year 2000 volume 16 no.4: discusses the Impact of Information Technology Investment on Organizational Performance. These articles provide chilling reading concerning the actual lack of evidence for any advantages of automation.

Warwick University did provide a large number of useful links concerning Business Process Reengineering (BPR) Research, Tools, and Practice but unfortunately this has now all gone but one small page provides details of a comparison between BPR, score cards and iSixSigma approaches: http://blogs.warwick.ac.uk/huangh/entry/module_review-le_part A PHd dissertation concerning BPR contains a interesting chapter (3) about the history of BPR available from http://www2.warwick.ac.uk/fac/sci/dcs/research/em/publications/phd/ychen/files/.

Business Process Redesign - An Overview (Article) This is an excellent introductory article which also explodes the myths concerning BPR. Well worth a read. http://www.brint.com/papers/bpr.htm

Business process Reengineering: from the perspective of competitive advantage by Edwin B dean Humanistic approach to BPR (+ bibliography from Nasa) A well referenced article about the managerial problems with BPR and how to cope with them. http://spartan.ac.brocku.ca/~pscarbrough/dfca1stmods/dfc/bpre.html

Add your own:
15. Appendix A – pre uml 2 message syntax

The UML 1.3 complete message specification

As you can see you can make a message rather complex. Note that the message name can be the same as one of the operations of the sending instance. I have taken the above from Pender 2002 p181 (adapted).