

REQUIREMENTS ANALYSIS AND UML

INTERACTION DIAGRAMS AND STATE TRANSITION DIAGRAMS

UML PROVIDES A POWERFUL FRAMEWORK AND NOTATION FOR MODELLING BUSINESS PROCESSES AND OBJECTS

by Richard Vidgen

In part 1 of this article (April 2003 C&CE, p. 12) four arguments for conceptual modelling were advanced: to clarify our thinking about an area of concern; as an illustration of a concept; as an aid to defining structure and logic; and, as a prerequisite to design. The unified modelling language (UML) is a widely accepted object-oriented notation for modelling and specifying system requirements. In part 1, the UML use case diagram and class diagram were introduced and applied to the fictional Barchester Playhouse, an organisation investigating the implementation of an Internet theatre ticket booking system. Use cases represent the system from a functional, user-centred perspective, while the class diagram exposes the structure of the system. In this article, the behavioural and dynamic aspects will be explored using interaction diagrams and state transition diagrams.

INTERACTION DIAGRAMS

The use case diagram identifies the functionality of the system from the point of view of the user. Using an interaction diagram we can show how the instances of the structural classes will interact and collaborate to achieve the implementation of each of the use cases. There are two types of interaction diagram in UML, the sequence diagram and the collaboration diagram:

- *Sequence diagram*: The sequence diagram models the collaboration between objects and actors, showing the

exchange of messages needed to accomplish a specific purpose. Typically, a sequence diagram is prepared to represent a single use case. As with class diagrams, sequence diagrams are elaborated throughout the development process with design detail. At the conceptual level the sequence diagram should be consistent with the use case and the conceptual class model.

Fig. 1 is a sequence diagram for the use case "Internet ticket sale". The dashed vertical line represents an object's lifeline, with time moving from the top of the diagram to the bottom.

THE SEQUENCE DIAGRAM MODELS THE COLLABORATION BETWEEN OBJECTS AND ACTORS

Objects are spread out across the page. Note that the class name is shown after a colon — this indicates that we are referring to an instance of the class, a particular production or performance rather than the class itself. Arrows represent the passing of messages from object to object. When a message is sent to an object it invokes an operation supported by that object's interface. The period of time that the operation is active is shown by a rectangle on the object's lifeline.

In Fig. 1 the collaboration begins with the Internet user finding a production. The production responds to the user by returning the details of the selected production (the return message is not shown, although it can be where it aids understanding). The user then requests a list of performances for the production by invoking the `listPerformances()` operation of the →

production object. The production object satisfies the request by sending the message (for all of the production's performances) `getPerformanceDetails()`. The iteration is shown by an asterisk. The Internet user then finds the prices for the different parts of the theatre by sending a message to the theatre. The next stage of the process is to find available seats.

In the second part of the Internet ticket sale use case (Fig. 2) the sequence is to check for an existing customer (and add a new one if needed), to check if the customer is due a discount on the price as a member, take payment and write the transaction, and finally to change the seat at performance status to booked. Note that instances are created for new customers and for the transaction class. The creation is shown at the point in the timeline they are created.

Separating the use case into two parts is rather useful since the find seats and book seats parts of the use case could be reused. As well as creating new objects, a message might invoke the destruction of an object. In the use case "cancel performance" the performance object then sends messages to invoke the object destruction operation for the seats at

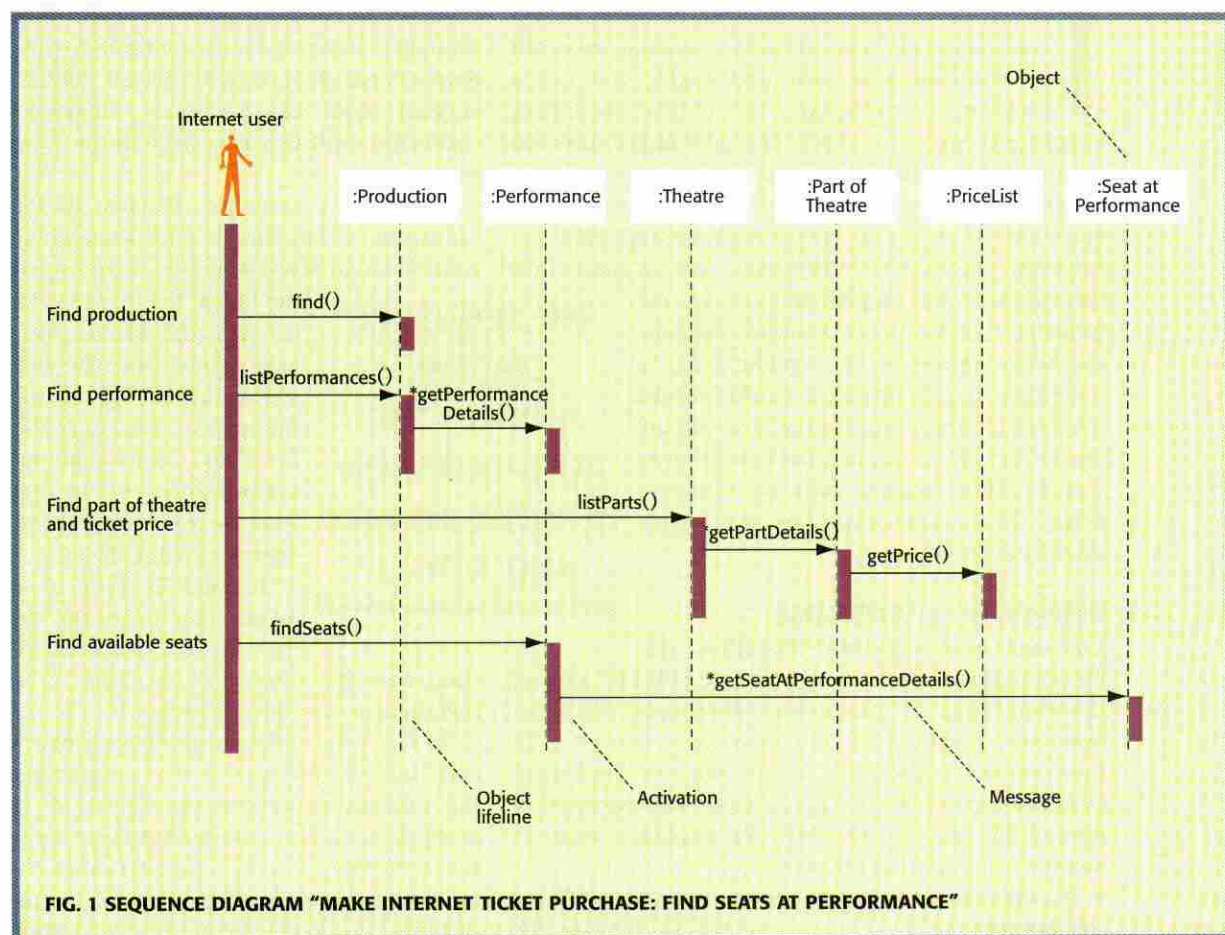
performance associated with that performance object (Fig. 3). The performance then sends a message to itself (reflexive) to destroy the performance instance.

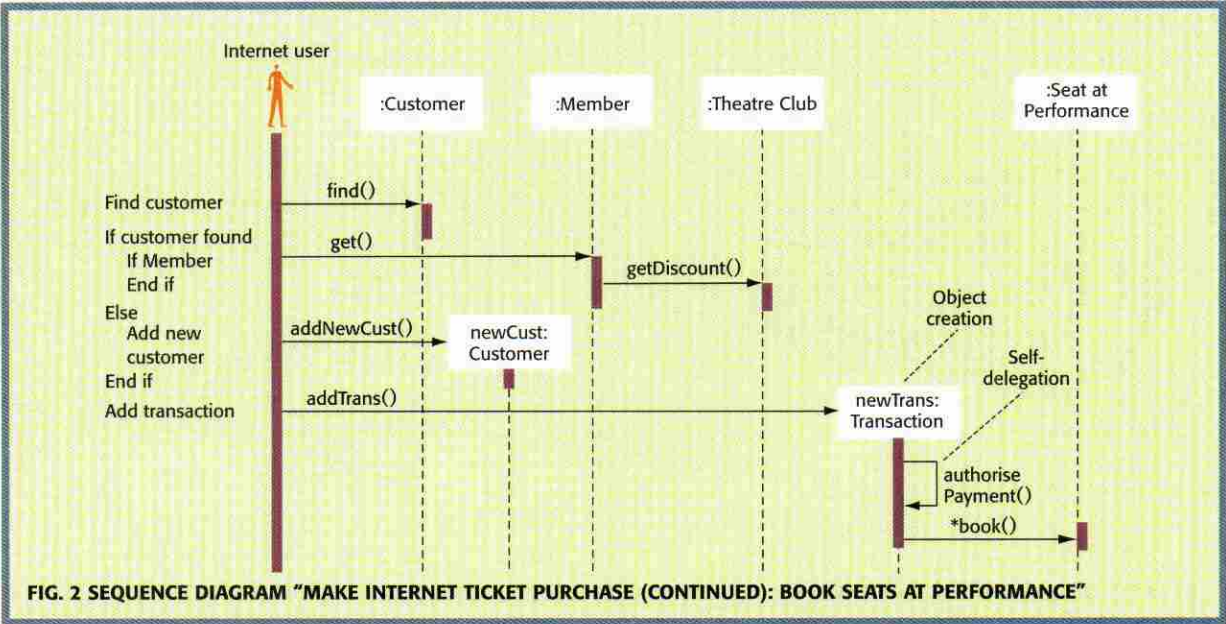
• *Collaboration diagram:* Collaboration diagrams are the second interaction modelling technique in UML. They do the same job as a sequence diagram — they show how objects work together to achieve a use case. In the collaboration diagram arrows show messages and boxes represent objects, with numbers being used to show the sequence of messaging (Fig. 4). With the collaboration diagram it is easier to see how the objects work together to achieve a use case, while the vertical lines presentation of the

sequence diagram makes the ordering clearer.

Both forms of interaction diagram, therefore, show collaboration between objects. From an analysis and conceptual modelling perspective it is probably sufficient to prepare a sequence diagram. As the diagrams are refined for design purposes then the collaboration diagram becomes more useful, allowing packages of processing to be identified from clusters of interactions. We have introduced some conditional

THE COLLABORATION DIAGRAM MAKES IT EASY TO SEE HOW OBJECTS WORK TOGETHER





behaviour into the second part of the sequence diagram (Fig. 2) to branch for customers who are members. As more conditional processing is added the interaction diagrams get more and more convoluted and it is wise to consider developing different diagrams for different scenarios, e.g. "Internet ticket purchase by member" and "Internet ticket purchase by non-member".

STATE TRANSITION DIAGRAMS

The class model has defined the structural aspects of the system and interaction diagrams show how objects interact to achieve a specific purpose, as defined by a use case. As we have seen already, an object can take on a number of states at different times — is the seat at performance available, reserved or booked? More formally, a state is a "condition or situation in the life of an object during which it satisfies some condition, performs some activity, or waits for some event" (Booch *et al.* Addison-Wesley, 1999).

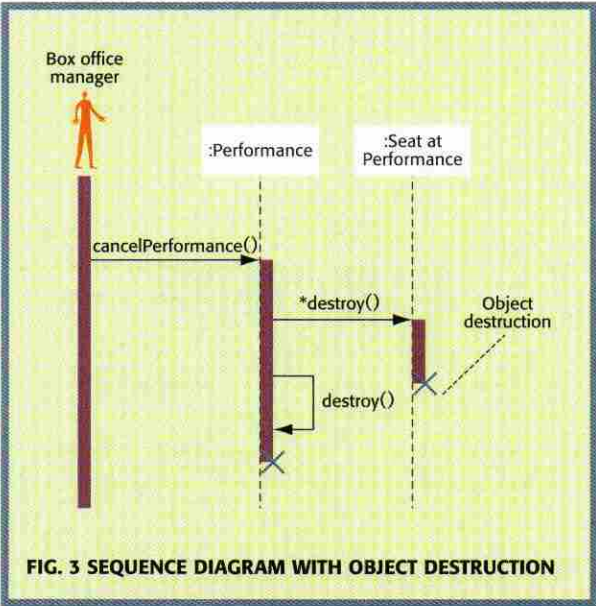
An event is something that happens in space and time: "user clicks on button", "it begins to rain", "I've been waiting for five minutes in a telephone queue". It is events that trigger changes in the state of objects — a state transition. When a user clicks on a button the result might be to maximise a window on the screen, i.e. a state change for the window from minimised to maximised. If it starts to rain then person changes to person with open umbrella; if I have been waiting for ten minutes I will cancel my call and become a dissatisfied customer.

A state transition diagram shows the different life cycles that an object of a class can undergo. The state of an instance is given by the attributes and associations it has. For example, a seat at a performance that has been booked will have an association with a transaction object. States are shown

as rounded rectangles and transitions as arrows. The labels on the transitions are of the form:

event [guard]/action

In Fig. 5 the transitions are labelled with events, such as "Die", "Marry" and "Divorce". In the UK it is necessary to get divorced before remarrying. If polygamy is allowed then a married person can get married again without being divorced. This is shown as a self-transition in Fig. 5 and a guard has been added to ensure that this transition can only take place if "Polygamy OK" (of course, although our model is accurate from a legal standpoint, it does not stop people committing bigamy). The state transition diagram is a template. It describes the possible →



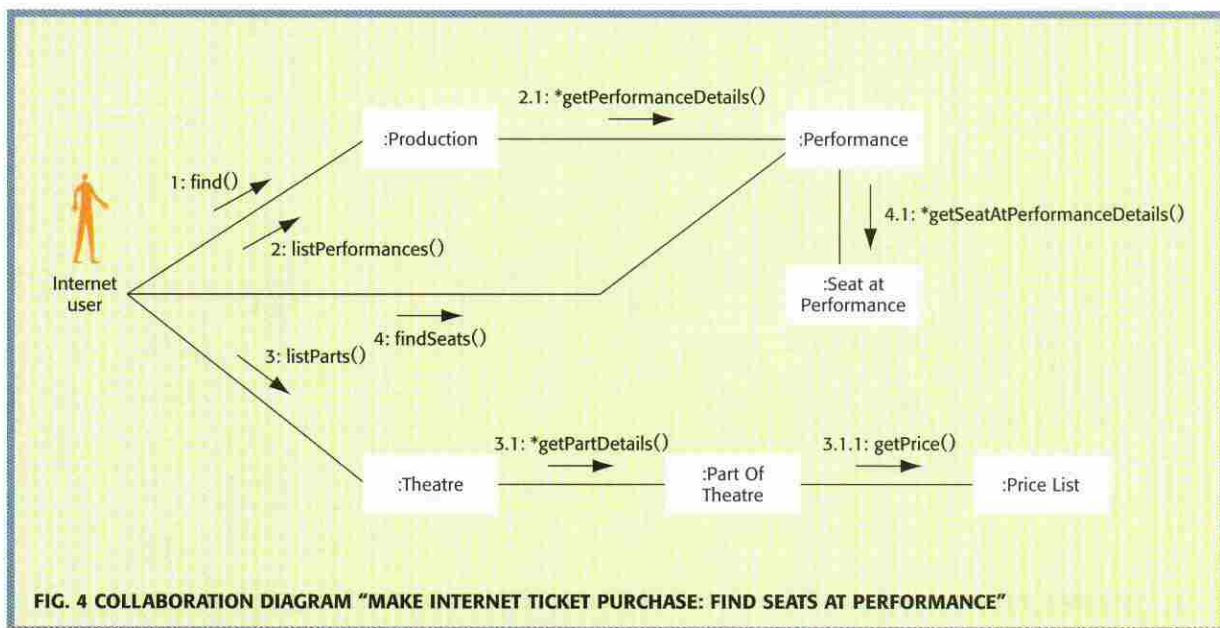


FIG. 4 COLLABORATION DIAGRAM "MAKE INTERNET TICKET PURCHASE: FIND SEATS AT PERFORMANCE"

paths that an object might take. For example, some people will be born and die without getting married. Others will marry once and others might marry multiple times. By definition, an object must have a state and can only be in one state at any one time.

State transition diagrams are prepared for those classes that have sufficiently complex states and transitions to warrant modelling. Some classes will have simple states and transitions going directly from creation to destruction. Others will have more complex behaviour, such as objects in the class SeatAtPerformance (Fig. 6). Note that the action is specified as an operation, such as "release", which is supported by a class, in this case SeatAtPerformance (see the theatre system class diagram in part 1 of this article for details of the classes and operations).

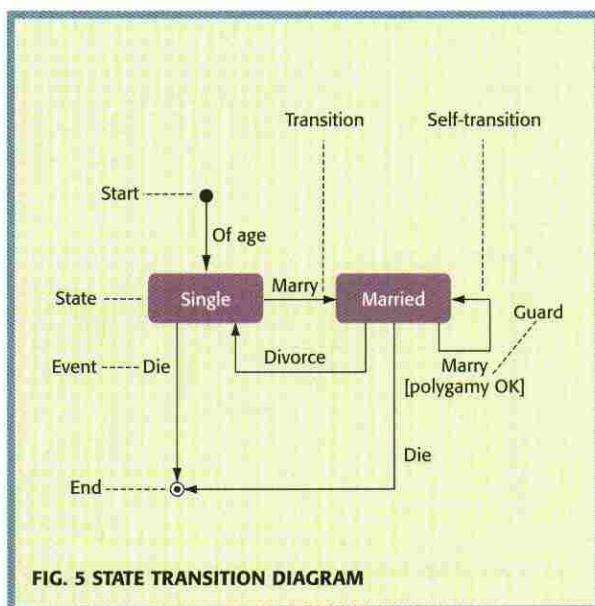


FIG. 5 STATE TRANSITION DIAGRAM

The state transition diagram therefore represents the possible life cycle of objects of a given class. It is closely related to the sequence diagram, which shows the message passing that will result in state changes in objects.

MOVING TO SOFTWARE SYSTEM DESIGN

The aim of this two-part article has been to use UML to model business requirements in conceptual terms. However, as noted earlier, UML is used throughout analysis and design and provides a basis for software implementation. We will therefore, look in outline terms at a suitable software system architecture for the ticket booking system. Any large and complex system needs to be divided into layers if it is to be comprehended and managed. The system design for the theatre booking application has been implemented in software using a three-tier client-server architecture. A client-server architecture links computers such that some computers — servers — perform functions for other computers — clients (usually end-user PCs). Indeed, this is how the worldwide web works. The user browses the web using a client PC that can request web pages (a service) from Internet web servers.

The three layers of the application can be configured in different ways, ranging from server-intensive (distributed presentation) to client-intensive (distributed data management). For the development of a demonstration theatre booking system a straightforward approach was taken: remote presentation. The presentation layer is handled by the user's web browser. The business logic layer resides on the server and is written in a scripting language (Macromedia's ColdFusion), but some of it could be distributed to the client (distributed function) to reduce network traffic and server load. For example,

form validation and simple calculations, such as VAT (sales tax), could be handled using JavaScript or a Java applet downloaded by the client. Data management also resides on the server. Although cookies can be used to store data on the client, for example, to identify returning customers, client-side cookies are only used for transient data. All permanent data are stored on the central server since cookies are ephemeral and specific to individual client PCs.

The architecture of the theatre booking system consists of three layers: business data services, business logic services, and user presentation services (Fig. 7). In UML notation, each of these layers is represented as a package. The user presentation services are the responsibility of the client, while business logic and business data services reside on the server. The business logic services handle business functions such as “find available seats at performance”, and the business data services layer is responsible for maintaining details of productions, performances, ticket transactions etc.

The UML stereotype <<import>> in Fig. 7 indicates that one layer has access to the contents of another layer, for example the user presentation layer can access the business logic services, but not the other way around. This architecture will make the application easier to maintain and extend in the future; changes to the databases or the addition of new business logic can be localised to individual layers such that they are transparent to the invoking layer. So, if the way the seat allocation algorithm works was re-specified, then the change is localised to the business logic layer — no changes are required to the

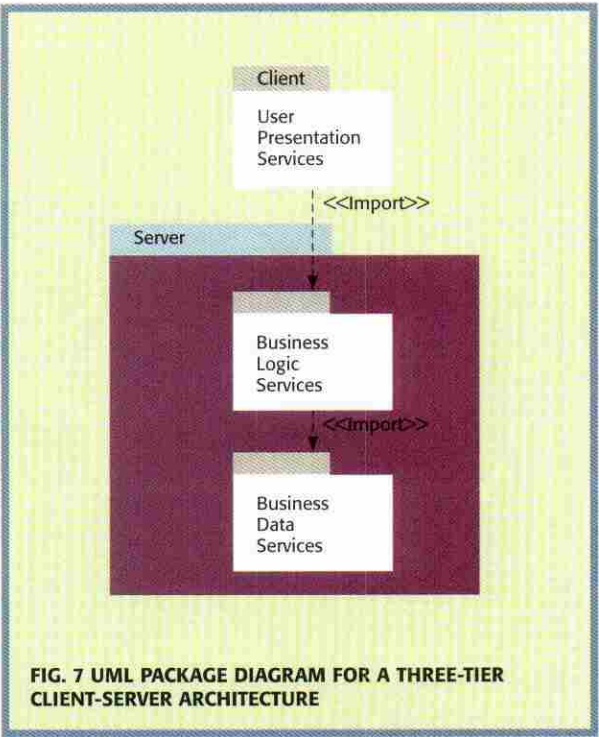


FIG. 7 UML PACKAGE DIAGRAM FOR A THREE-TIER CLIENT-SERVER ARCHITECTURE

user presentation services, assuming that the interface exported by the seat allocation business logic is unchanged.

BASIS FOR DESIGN

The UML notation can be used to model business and organisational requirements from a conceptual perspective with aims that include gaining understanding of the current situation, redesigning business processes, and providing a structural and behavioural model suitable for developing a software system. In the first part of this article use cases were used to model functions from a user perspective and class diagrams to lay out the structure of the situation. In this part the behavioural aspects were modelled using interaction diagrams and state transition diagrams. Collectively, these models of the business domain form a basis for the design and construction of a software system to support the ticket booking process.

ACKNOWLEDGMENTS

This article is reproduced with the permission of the publisher and is an abridged extract from: Vidgen, R. T., Avison, D. E., Wood, J. R. G. and Wood-Harper, A. T.: “Developing Web Information Systems” (Butterworth-Heinemann, 2003). Further details of WISDM are available at www.wisdm.net together with a demonstration theatre ticket booking system developed using ColdFusion MX.

Richard Vidgen is with the School of Management, University of Bath, Bath BA2 7AY, UK, e-mail: mnsrtv@management.bath.ac.uk

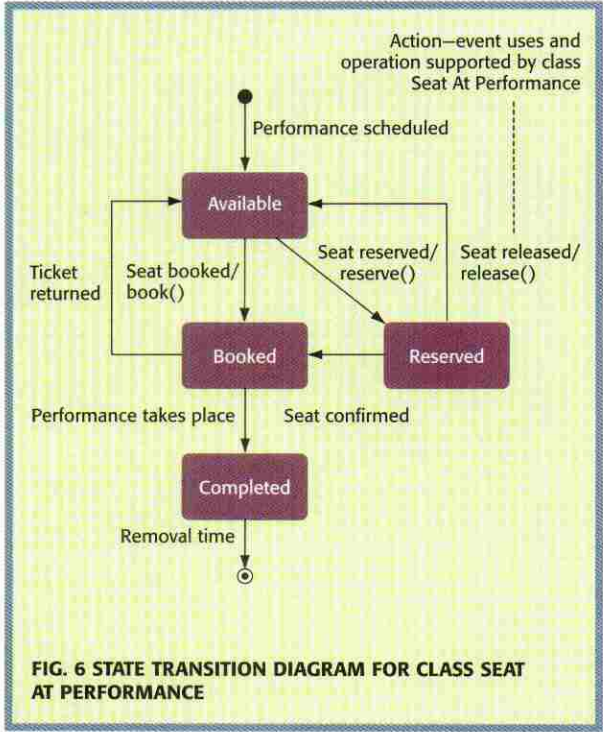


FIG. 6 STATE TRANSITION DIAGRAM FOR CLASS SEAT AT PERFORMANCE

