DIVISION OF COMPUTER SCIENCE

The Object Modeling Technique

David Levine and Carol Britton

Technical Report No.165

October 1993

CONTENTS

The Object Modeling Technique

Section 1 : Introduction

This report examines one object oriented development method, the Object Modeling Technique (OMT) of Rumbaugh et al [Rum91]. There are three main sections which summarise the technique, outline the key precursors and review the literature on the OMT. Knowledge of the basic concepts in object oriented systems, finite state machines and software design methodologies is assumed.

The OMT was chosen for practical use on a small case study in order to gain understanding of the concepts and issues in object oriented development. A personal account of using the OMT on the case study is given in a complementary technical report titled "An educational game of relationships : its modelling using the OMT".

There are several reasons why the OMT was picked from the many object oriented methodologies that are available. These reasons are :

(i) It is acknowledged as being one of the so called "state of the art" methods ([Aks92] , [Wal92]).

(ii) It is well documented in [Rum91] and by regular articles by Rumbaugh in the Journal Of Object Oriented Programming.

(iii) It covers the whole front portion of the software development process : analysis, design and implementation.

(iv) It has formed the basis for further research ([Hay91] , [Jer93]) and a number of CASE tools[1].

Section 2 of the report summarises the technique described by Rumbaugh et al [Rum91]. The summary aims to be an objective account with no evaluation or expression of viewpoints. The section is divided into subsections covering the four stages of the methodology. These stages are :

(i) Analysis: This stage details the building of the analysis model from a statement of the problem. This model [Rum91 page 5] is a 'concise, precise abstraction of what the desired system must do'.

(ii) System design: The target system is partitioned into subsystems based on the analysis model and decisions are made on the overall system architecture.

(iii) Object design: A design model is built based on the

[1] Select Software Tools' C++ Designer, Cadre's Paradigm Plus and Advanced Concepts Center's OMTool.

analysis model. Implementation details are added by focusing on the data structures and algorithms needed for each class.

(iv) Implementation: The design model is translated into a particular programming language, database, or hardware implementation.

Section 3 examines the question "Where did the technique come from?". It outlines the key papers, from which the technique was developed. It is divided into subsections covering the three kinds of model which the OMT uses to describe a system. The three kinds of model are :

(i) The object model describing the static structure of the objects in a system and their relationships.

(ii) The dynamic model describing the temporal behaviour of a system and in particular its control aspect.

(iii) The functional model describing the data value transformations in the system.

Section 4 looks at the responses to the OMT. Several papers have examined object oriented methods in general and included the OMT as one of the example methodologies ([Aks92], [Cha92], [Wal92]). Other papers concentrate solely on the OMT ([Bru92], [Hay91]). The aim of this section is to detail the major strengths and weaknesses of the OMT as seen by other authors.

Finally section 5 concludes by noting some wider issues regarding the choice of methodology.

Section 2 : The Object Modeling Technique

This section of the report summarises the technique described by Rumbaugh et al [Rum91]. The four stages of the methodology, analysis, system design, object design and implementation are outlined. The authors adopt an informal approach : " there are no proofs or formal definitions with Greek letters" [Rum91 Preface page x].

2.1 The Analysis Model

This subsection outlines the purpose of the analysis model, its inputs and outputs, its three constituent models (the object model, dynamic model and functional model) and the relationships between the three models.

The general aim of analysis with the OMT is to build a model of the real world system starting from a problem statement. During the analysis essential features of the application domain are abstracted without regard to implementation. The two main objectives are to obtain a clear understanding of the problem and to produce a model that will form a basis for the design stage.

The initial input to the analysis stage is a problem statement. "The problem statement should state what is to be done and not how it is to be done" [Rum91 p150]. It is a starting point for understanding the problem and is revised following analysis. The output from analysis consists of a revised problem statement and an object model, a dynamic model and a functional model.

2.1.1 Object Model

The object model captures the static data structure of a system and consists of an object model diagram and a data dictionary .In general it is the most important of the three models because as an object oriented approach the emphasis is placed on the identification of the objects with the fitting of procedures around the objects.

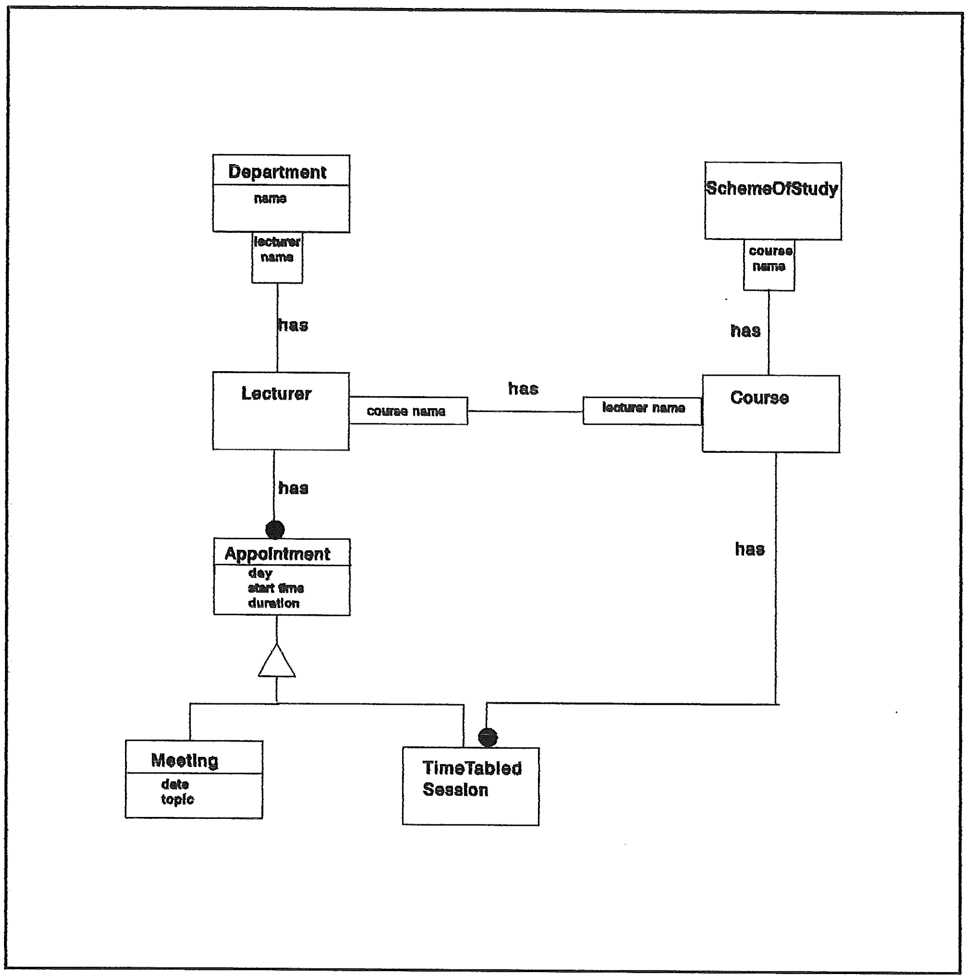
The object model notation is descended from the entity relationship (ER) model [Che76]. An object diagram describes the static data structure of objects, classes, and their relationships to one another. The notation includes constructs for the representation of a class and its attributes and operations and for the representation of the relationships between classes.

Emphasis is placed on associations. An association describes a group of links with common structure and semantics. A link is an instance of an association and connects two or more objects. Qualified associations are used for greater precision (see section 3).

Aggregation , the "is-part-of" relationship is regarded as a special form of association with additional restrictions such as transitivity and antisymmetry. Generalisation is the relationship between a class (the superclass) and one or more refined versions of it (each version being a subclass).

Some of the OMT notation is illustrated in figure 1 which is an object diagram for an appointment system for lecturers taken from a case study by Buchanan [Buc90].

Figure 1 : Appointment system - object diagram



In figure 1 qualified associations are used between Lecturer and Department and Course, and between Course and SchemeOfStudy. Appointment is an abstract class with subclasses Meeting and TimeTabled_Session.

The OMT lays down the following series of steps for building an object model. Iteration of the steps is a key feature of the methodology.

* the identification of object classes.
* the preparation of a data dictionary describing classes, associations, attributes and operations.
* the identification of associations.
* the addition of attributes.
* simplification using inheritance.
* the testing of access paths by asking questions.
* the grouping of classes into modules (logical subsets of classes).

2.1.2 Dynamic Model

The dynamic model describes temporal behaviour and consists of state diagrams for each class with important dynamic behaviour and a global event flow diagram for the system. Its most important use is to represent the control aspect of a system.

Harel's notation is used [Har87]. This is an extension of the standard state diagram notation that includes additional constructs that overcome problems with flat unstructured state diagrams for complex systems (see section 3).

Operations in response to events are classified by their duration relative to the timescales of the application. An activity is an operation that takes time to complete and is associated with a state. An action is an instantaneous operation associated with an event.

The steps involved in building a dynamic model are :
* the preparation of scenarios showing sequences of events.
* the drawing of event traces for each scenario showing the objects sending and receiving events.
* the preparation of a global event flow diagram showing the events between the classes in the system.
* the development of state diagrams.
* the checking of consistency and completeness between the state diagrams.

Figure 2 shows the statechart for the initial state of the appointments system [Buc90]. The dotted line separates two orthogonal states. The dot and arrow symbol shows the default states "Request Department Name" and "Error Message Off". Example of an activity and action are "do: request department name" and "verify department name" respectively. "Enter(department name)" is an event with the attribute department name and [invalid department] and [valid department] denote conditions.

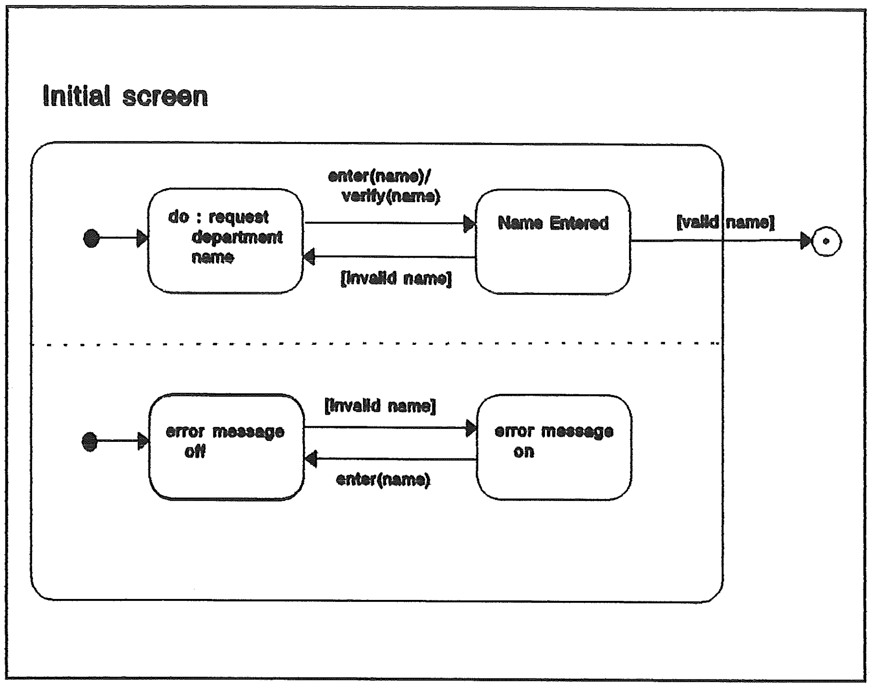


Figure 2 : Appointment system - initial state.

2.1.3 Functional Model

The functional model describes data value transformations and consists of data flow diagrams (DFD's) and constraints. The OMT's use of DFD's follows closely the approach of traditional methodologies such as Yourdon [You89] including the use of control concepts with the addition of a new construct, a hollow arrow, representing a data flow that results in a data store. A constraint shows the relationship between two objects at the same time or between values of the same object at different times.

Building a functional model involves :
* the identification of input and output values.
* the preparation data flow diagrams .
* the description of each function (bottom level processes in the DFD).
* the identification of constraints.
* the specification of optimisation criteria.

2.1.4 Conclusion

Each model focuses on a particular aspect. The relationship between the models is summarised in table 1 overleaf. Although for most systems the object model is the most important, the balance of importance varies according to the kind of application. For example, the dynamic model is important for interactive systems whereas the functional model is the main model for systems such as compilers.

Table 1 : Relationship between the models. (adapted from [Rum91] p139)

	Object Model	Dynamic model	Functional model
Relative to Object Model	--------------	* shows the states of each object with important dynamic behaviour. * shows the operations on events (actions) and associated with each state (activities).	* defines the operations.
Relative to Dynamic Model	* shows what changes state and is operated on.	--------------	* defines the actions and activities.
Relative to Functional Model	* shows the structure of the actors, data stores and data flows.	* shows the control sequence of the processes	--------------

Following the initial construction of the three models, operations from the dynamic and functional models are added to the object model. The overall analysis model is refined by removing inconsistencies and by testing using scenarios including error conditions. The final model is verified with the client and used to revise the problem statement.

2.2 System Design

Design in the OMT is split into a high level stage called system design and a more detailed stage called object design.

The OMT's treatment of system design is aimed at projects with up to ten developers. System design gives an overall strategy for constructing the software system. The strategy is produced by answering a set of high level decisions. The decisions that *must be made* (our italics) are listed below. The high level structure formed by making the decisions is called the system architecture.

2.2.1 High-level decisions

* *division into subsystems.*

Division into subsystems is the first step in system design. A subsystem is a package of classes that are interrelated and have a small well defined interface. The internal implementation is not visible to other subsystems enabling object design and implementation to be divided between more than one developer.

A system can be divided into both vertical partitions and horizontal layers. Partitions are ideally independent subsystems. A layer either uses the services of the layer immediately below (forming a closed architecture) or can use services from any of the layers below (forming an open architecture).

The relationship between two dependent subsystems is either that of a client-supplier in which the supplier does not know about the client interfaces or a peer-to-peer relationship in which there is mutual knowledge of the interfaces. Client-supplier relationships are simpler and therefore more desirable.

* *identification of concurrency.*

The dynamic model is used to identify concurrent and mutually exclusive objects.

* *allocation of concurrent subsystems to hardware units.*

Performance needs are used to determine the hardware units to be used.

* *management of data stores.*

A decision is made on the form of permanent data storage. The two main choices are between the use of a DBMS and the use of files.

* *handling of global resources.*

Global resources such as processors, disk space and logical names are identified. Guardian objects are used to control access to these resources.

* *implementing control*

Choices about implementing both external and internal control flows are made.

"External control is the flow of externally-visible events among the objects in the system [Rum91 p207]." Procedure-driven, event driven and concurrent systems are three kinds of control for external events. Event driven systems should be used in preference to procedure driven systems.

"Internal control is the flow of control within a process [Rum91 p208]." A process may be split into several tasks. They are generated by objects as part of the implementation algorithm, so their response patterns are predictable. Most internal operations can therefore be thought of as procedure calls, in which the caller issues a request and waits for a response.

* *handling of boundary conditions.*

Decisions are made on initialisation, termination and failure (unplanned termination).

* *setting of trade-off priorities.*

Trade-off priorities for incompatible goals such as frequency of disk accesses and execution speed are made. These are used to guide the design process.

2.2.2 Common architectural styles

In addition to making the high level decisions above, the authors identify a number of common architectural frameworks that can be used to guide the design. These include interactive interfaces, dynamic simulation, real-time systems and transaction managers. Most applications are a hybrid of these common frameworks.

2.3 Object Design

During Object Design ([Rum91] Chapter 10) full definitions for the classes, associations and operations are added to the analysis models . Object Design is analogous to the *detailed*[2] design stage in a traditional lifecycle.

The product of Object Design is a Design Document which is a revision of the Analysis Document. It includes an extensive revision of both the object diagrams and class descriptions of the Object Model . The Functional Model is also revised and kept current. Additions to the Dynamic Model are made if a procedure-driven approach is chosen. Although the Design Document uses the same names and follows on from the Analysis Document it is advisable to keep the two documents separate in order to retain the user's view of the system.

Object Design is broken down into a series of steps whose importance is dependent on the nature of the subsystem being examined. The steps concern operations and their algorithms, optimisation, control, inheritance, associations, object representation and physical packaging.

2.3.1 Operations and their algorithms

Combining the three analysis models to obtain operations on classes is one of the end stages of the analysis. For simple operations such as accessing the value of a variable the specification in the functional model is sufficient for implementation. For complex operations algorithms and data structures are chosen. This can involve the use of class libraries and the introduction of internal implementation classes that were not part of the analysis model and do not affect the functionality of the system.

2.3.2 Design optimisation

The importance of the trade-off between efficiency achieved by design optimisation and clarity is a decision which will have been made during System Design. Associations paths are analysed to determine frequent inefficient operations for which indexes can be added. Saving derived data in order to avoid recomputation involves another trade-off between the cost in execution time and the addition of extra code for updating the derived data.

2.3.3 Implementation of Control

Three approaches are described in system design for implementation of the control aspect of the system as represented by the dynamic model. This aspect is only relevant for those

[2] Rumbaugh states that object design is analogous to the *preliminary* design phase [Rum91 page 227]. Our understanding is that preliminary design more accurately describes system design.

subsystems with important dynamic behaviour such as the user interface. For the procedure-driven option pseudocode is written showing the flow of control. For the event-driven system a state machine class with the state diagram represented as a table can be used.

2.3.4 Inheritance

Increasing the amount of inheritance entails examining the classes and operations in order to abstract out common behaviour into superclasses. Operations that are taken into a superclass must have the same signature and semantics and may need to be adjusted to enable the "abstracting out" to be done.

Any subclass should be a form of its superclass. "Inheritance of implementation" is a technique in which an existing class is used as the superclass for a subclass for which not all the operations in the superclass are semantically appropriate. Delegation and not inheritance is recommended for such cases.

2.3.5 Design of associations

The analysis of association traversal is the first step in their design.

When the traversal is just one direction it is easiest to implement as a buried pointer with the pointer being added to just one of the classes. The term buried pointer is appropriate because the pointers are additional attributes which are implementation constructs and are not visible in the Analysis Model. Dictionary objects, which are available in class libraries, can be used for the implementation of qualified associations.

The weakness of assuming one way traversal is that if the requirements change then the associations may become two-way. Pointers may be added to both classes or an independent association class may be introduced consisting of a set of pairs of pointers.

2.3.6 Physical packaging

Guidelines are given on information hiding, the coherence of entities and the construction of modules.

At the design stage information hiding involves separating the interfaces of the classes into public and private domains and limiting the scope of the methods.

Entities such as classes and methods should have a single major theme. A method's theme should focus either on making decisions (policy theme) or on the execution of an algorithm (implementation theme). This increases the possibility of reusability as methods containing an implementation theme are less likely to be application dependent.

In the OMT, modules are logical constructs for grouping classes and their associations. Modules should be cohesive with inter-module coupling minimised.

2.4 Implementation

The crucial decisions about implementation are made during the design stages. This subsection gives a brief outline of the final part of the OMT which contains a set of wide ranging guidelines for implementing the design to achieve goals such as maintainability and extensibility. These guidelines cover object oriented programming style and the features provided by C++, Eiffel and Smalltalk for implementing classes, creating objects, calling operations and handling inheritance and associations. The mapping to non object oriented languages C, Ada and Fortran is covered as well as the design and implementation of relational databases from the object model.

Section 3 : Background to the OMT

The section will draw out the most significant features of the primary sources for the OMT.

The three models which are the basis of the methodology have independent origins. The object model is introduced in the original paper on the OMT by Loomis et al [Loo87] while its emphasis on associations originates from [Rum87]. The dynamic model uses Harel's extension of the state diagram notation [Har87], [Har88]. The functional model uses traditional data flow diagrams (e.g. [You89]).

3.1 Original paper

The original paper on the OMT [Loo87] is titled "An Object Modeling Technique for Conceptual Design". Conceptual design refers to high-level design without any internal detail.

The product of the technique is the object model only, consisting of a diagram, descriptions of method behaviour and class definitions.

The three types of relationships, generalisation aggregation and association are identified. However in conflict with [Rum91], "there is direct interchangeability between associations and attributes [Loo87]". Qualified relationships are introduced to reduce the multiplicity for many to one or many to many relationships.

The technique's starting point is the production of a problem statement by a discussion with subject experts . Classes are identified followed by relationships and their cardinalities. The model is then reviewed with subject experts before listing methods and checking their access paths. Attributes are then listed and relationships are collapsed. Finally the model is again reviewed with subject experts. Feedback and iteration of the steps is a key feature of the technique.

Its extension of the entity relationship model [Che76] and the emphasis on relationships, places the origins of the OMT in the "database camp".

3.2 Use of associations

The ideas on associations are detailed in [Rum87] where the term relation used is synonymous with the term association. The key arguments are that associations should have the same semantic importance as generalisation and secondly that object oriented languages should support the association (relation) construct. The author argues that for large systems associations are more important than generalisation because they are more likely to affect the partitioning of the system. The object-relation model is introduced (the term object-relation model is dropped in [Rum91]) which combines class hierarchy and methods with associations taken from semantic data modeling. Unlike the relational data model, objects can appear directly in associations.

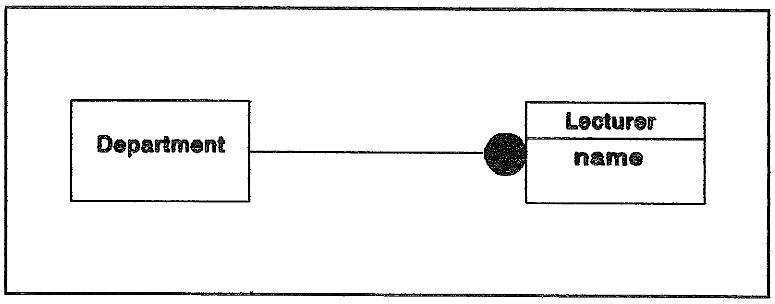
Objections to the use of associations because of the violation of information hiding are anticipated. Two classes in a binary association "know" about one another. For example the update of an association element affects more than one class. It is argued that information hiding in separate classes hides semantic information if the classes are in an association relationship. The information is shared among several classes. The use of associations as both a logical and implementation construct means that the computer world can better model the real world.

The use of qualified associations, which are used at the analysis stage, is introduced. Qualified associations often arise when names are used as qualifiers on a set of objects. The example below illustrates this idea which according to [Rum93c] is unique to the OMT.

Example : use of qualified associations.

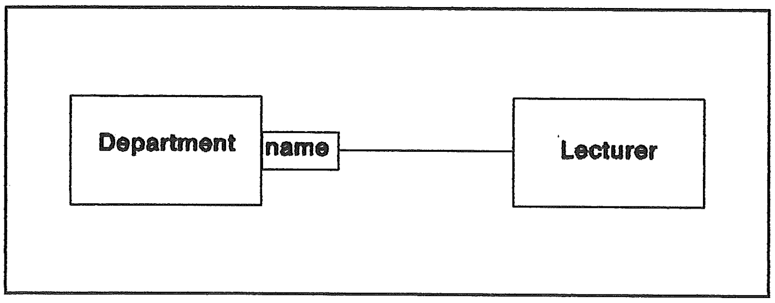
A department has many lecturers with an individual lecturer being assigned to just one department (figure 3).

Figure 3



The names (including initials) of the lecturers are used to distinguish among the set of lecturers for a particular department. Each set of names is local to a department. A name qualifies a department to identify a unique lecturer. There is a one-one association between a (department, name) pair and a lecturer (figure 4).

Figure 4



Rumbaugh argues that it is better to use qualified associations to represent names rather than to assume unique names. This use makes a system modelled using qualified associations more resistant to change. Also, by reducing the many multiplicity to a qualified one multiplicity navigation through the object model is facilitated. At the design stage qualified associations can be represented by lookup tables. "The selector values in the lookup table are the qualifiers and the target values are the objects in the other class." [Rum93c].

3.3 Use of statecharts

Statecharts are one of a series of specification techniques used to specify behavioural requirements [Dav88], [Dav90]. The behavioural requirements for a particular piece of software are concerned with the description of the interfaces between the software and its environment. The primary reason for the use of specification techniques is to overcome the inherent ambiguities of natural language descriptions [Dav90].

The motivation for statecharts arose from the problems with describing large and complex reactive systems. Reactive systems such as operating systems are event driven, reacting continually to external and internal stimuli. The traditional state diagram notation used to define a finite state machine is unsuitable for these type of systems for a number of reasons [Har88] :

* Because of the flat nature of the notation, the number of states grows exponentially as a function of the number of independent attributes affecting control. Also there is no support for nesting diagrams.

* An event which causes the same transition from a large number of states, is represented by an arrow for each state. This can result in spaghetti like diagrams.

* The notation does not support concurrency.

Harel's statechart notation solves these problems by providing a number of extensions. The basic extensions are described below. Figure 5 shows a conditional transition from state A to state B. When event g occurs in state A, provided condition c is true, the system transfers to state B.

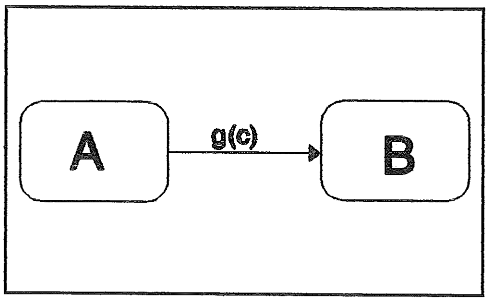


Figure 5 - conditional transition

The superstate extension is illustrated by figure 6. When the system is the superstate D , then it is in either state A OR state B (exclusive OR). Event f in state D, causes the transition to state C. State D is a superstate because the arrow f captures a common property of states A and B.

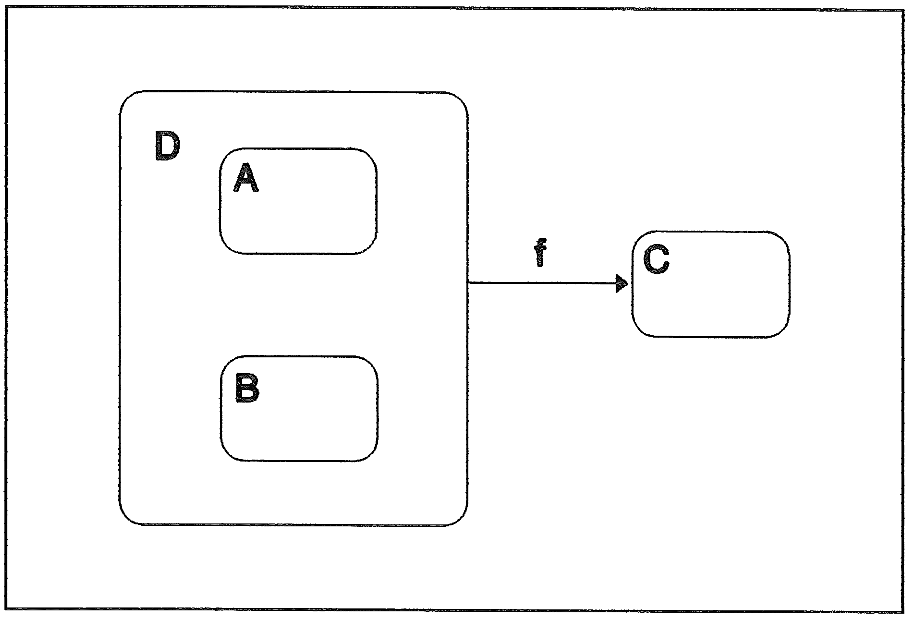


Figure 6 - superstate extension

The notation for supporting concurrency and default states is shown in figure 7. State Y is in states A AND D. This orthogonality is denoted by the dashed line. The dot and arrow symbol is used to denote default states. In figure 7 the initial states are B and G.

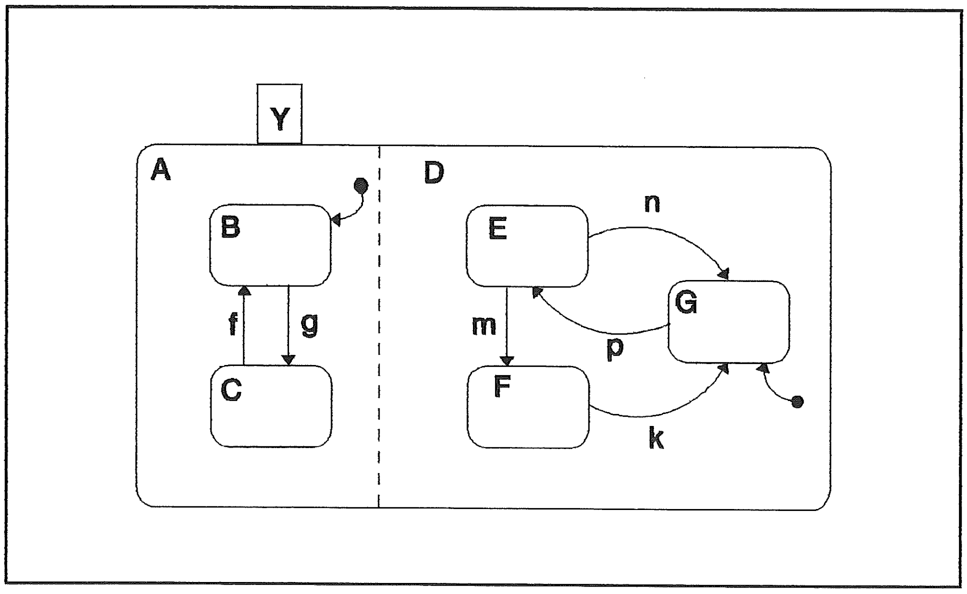


Figure 7 - orthogonal states

In addition to the basic features above, use is made by the OMT of the concepts of actions and activities. Actions and activities give statecharts the ability to model the generation of events. An action is an event that is an instantaneous occurrence. These are expressed by the label "…/s" attached to a transition where s denotes an action. For example, in figure 8 the event n in state F generates the action g. This can affect the behaviour of an orthogonal part of the statechart. If the system is also in state E then the transition from state E to state B will also occur. This is a simple broadcast mechanism.

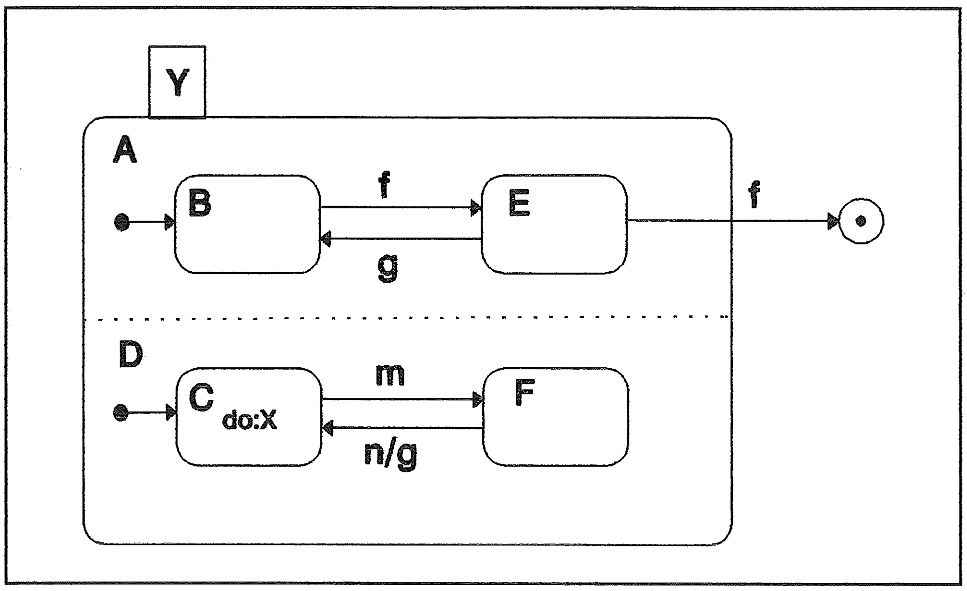


Figure 8 - actions and activities

In contrast to an action "an activity always takes a nonzero amount of time" [Har87] p256. An activity X is associated with the two actions start(X), stop(X) and the condition active(X). Rumbaugh uses a different notation do: X associated with a state (see figure 8) which encompasses the condition active(X), the action start(X), carried out on entering the state, and the action stop(X) carried out on leaving the state.

3.4 Functional model

The functional model uses traditional data flow diagrams with the addition of control flows e.g. [You89]. The only new notation is the use of a hollow triangle at the end of a data flow. This indicates the generation of an object for use by another operation.

3.5 Conclusion

The OMT is the first object oriented methodology to combine the three views of a system represented by the object, dynamic and functional models. It provides a notation and detailed guidelines for the construction of the analysis models and for their elaboration during system and object design.

Section 4 : Responses to the OMT

This section examines the responses to the OMT by noting the strengths and weaknesses of the methodology that are identified in the literature. The responses are concentrated on the analysis, the most important of the phases. Design, implementation and the methodology as a whole are briefly covered. Our own responses to the OMT are detailed in the complimentary technical report "An educational game of relationships : its modelling using the OMT".

4.1 Analysis

In the object model [Wal92] supports the OMT's assertion about the importance of associations and welcomes the explicit approach to qualifications and constraints. [Cha92] notes the treatment of attributes as pure data values, a direct consequence of the OMT's use of associations. [Bru92] writes in the context of the use of the OMT for a large group project in an undergraduate course. The object diagrams were a particularly good vehicle for communication throughout the analysis and design stages of the project.

[Aks92 p345] focusing on obstacles to object oriented development notes that separation into subsystems is left late in the methodology. This can result in an excess number of objects being considered making the analysis unmanageable . (These problems are not isolated to the OMT and early separation brings in problems with loss of commonality between classes).

In the dynamic model [Cha92] notes the use of nested state diagrams and the classification of operations into actions and activities. [Wal92] feels that the relationship between the object model and the dynamic model is carefully thought out. In particular he notes the thought given to the relationship between the state diagrams of a superclass and its subclasses. State diagrams of a superclass are inherited by a its subclasses. [Aks92] also records that the OMT alone, addresses this issue of generalisation of state specifications.

[Mon92] distinguishes micro from macro level structure in the use of the dynamic model. The macro layer deals with a system's control and is at a higher level of abstraction than the micro layer dealing with an object's control. The OMT's dynamic model only deals with micro level states and transitions and neglects higher level interaction between objects. This is also pointed out by [Jer93] in their work on building a new methodology partly based on the OMT. [Hay91][3] which only considers analysis, is another paper criticising the weakness of the OMT on object interaction. The paper points out that the lack of communication primitives means that overall system behaviour cannot be deduced

[3] Both [Hay91] and [Jer93] are produced by the same research team at Hewlett Packard.

from the behaviour of individual objects.

The use of the dynamic model in modelling the user interface is seen as a problem by [Mon92] because its approach is too low level. Identification of interface classes is a related activity that is neglected in the OMT.

The functional model with its use of data flow diagrams is seen as the least successful part of the methodology [Wal92]. [Hay91] emphasises the use of natural language for defining basic concepts in the functional and dynamic models. This use "makes it impossible to check that the dynamic model is consistent with the functional model." [Hay91 p175]

4.2 Design and implementation

The treatment of system decomposition is a weakness noted by [Jer93] and implied by [Aks92]. [Arn91] infers that the OMT is weak on design compared to other the four methods evaluated (Booch, Buhr, HOOD and Wirfs-Brock).

Rumbaugh is noted as being strong on its discussion of implementation issues by [Arn91].

4.3 The methodology as a whole

[Arn91] notes that the process is well defined with useful heuristics to guide development. The iteration between processes (the seamless transition) is one of three features noted by [Bru92] necessary to support the successful completion of a student project. (the strength of its notation and the availability of a CASE tool are the other two features). The method's ease of use and learning is another strongpoint.

[Wal92] and [Arn91] note the weakness of the OMT in its failure to address the reuse of both software and design components. The OMT assumes greenfield development. Taking a wider view of the methodology [Wal92] points out that there is almost a total neglect of issues such as testing, quality and the management of the software engineering process. In fairness [Rum91] only claims to cover the front portion of the software development process. [Mon92] refers to the different levels of granularity between the three analysis models. The object and dynamic models are at a micro level (see above) whereas the functional model is at a macro level. These different levels make it difficult to integrate the models.

Section 5 : Conclusion

A reading of the literature ranks the OMT as being one of the leading "first generation[4]" object oriented methodologies. Its status is also vindicated by the availability of CASE tools by several companies and the production of training courses (e.g. by QA Training Limited). In addition a revised version of the OMT is to be published in 1994 and this version will hopefully address the problems raised in the literature. This support and revision should strengthen the OMT's position in the market place.

The number of methodologies being developed continues to grow. This profusion has caused confusion in the industry and moves towards a common base model have been made by the Object Management's Group special interest group on analysis and design. This work on a common base model has been attacked in a letter [Mel93] by some of the leading exponents of object oriented methodologies, including Rumbaugh. They argue that the change in methodologies is currently so rapid that any standardisation of methods will discourage innovation and prevent maturation. "We need to deploy many large systems with many different methods and maintain them for several years to establish effectiveness. This will yield a base model for standardisation"

If the authors of the letter are correct then the OMT may well provide many of the elements of the base model of the year 2000.

[4] those methodologies published between 1988 and 1991.

References

[Aks92] Mehmet Aksit & Lodewijk Bergmans ,
"Obstacles in Object-Oriented Software Development",
OOPSLA, 341-358, 1992

[Arn91] P.Arnold, S.Bodoff, D.Coleman, H.Gilchrist, F.Hayes,
"An evaluation of Five Object-Oriented Development Methods",
Hewlett-Packard Report HPL-91-51, June 1991

[Bru92] B. Bruegge, J.Blythe, J.Jackson and J.Shufelt ,
" Object-Oriented System Modeling with OMT",
OOPSLA, 359-376, 1992

[Buc90] M.Buchanan, "Phantom of the Object",
M.Sc Thesis , Hatfield Polytechnic, 1990

[Cha92] D. de Champeaux, P.Faure, "A comparative study of object-oriented analysis methods", JOOP, Vol 5, Issue 1, 21-33, March/April 1992

[Che76] P.P.S Chen, "The Entity-Relationship model - towards a unified view of data"
ACM Transactions on Database Systems 1, 1(March 1976)

[Dav88] Alan M. Davis, "A comparison of techniques for the specification of external system behavior",
Communications of ACM 31,9 (Sep. 1988), 1098-1115

[Dav90] Alan M. Davis, "Software Requirements Analysis and Specification", Prentice- Hall , 1990

[Har87] David Harel,
"Statecharts : a visual formalism for complex systems",
Science of Computer Programming
8, (1987), 231-274

[Har88] David Harel, "On visual formalisms",
Communications of ACM 31,5 (May 1988),514-530

[Jer93] P.Jeremaes, D.Coleman, "Fusion: A second generation object-oriented analysis and design method",
IEE Colloquium on Object-Oriented Development, January 1993

[Hay91] F.Hayes, D.Coleman, "Coherent models for object oriented analysis", OOPSLA , 171-183 ,1991

[Loo87] M.Loomis ,A.Shah and J.Rumbaugh,"An Object modeling technique for conceptual design",
Proceedings of ECOOP Paris, France
Lecture Notes in Computer Science, 276 Springer-Verlag, New York 1987

[Mel93] S.J.Mellor, S.Sclaer, G.Booch, J.Rumbaugh, J.Salmons, T.Babitsky, S.Adams, R.J.Wirfs-Brock,
"Premature methods standardisation considered harmful",
Open Letter to the Industry , JOOP July/August 1993

[Mon92] D.E.Monarchi and G.I.Puhr, " A Research Typology for Object Oriented Analysis and Design."
Communications of ACM, 35,9 (September 1992), 35-47

[Rum87] J.Rumbaugh, "Relations as semantic constructs in an object-oriented language",
OOPSLA' 87 as ACM SIGPLAN 22, 12 (Dec. 1987), 466-481

[Rum91] J.Rumbaugh, M.Blaha, W. Premerlani, F.Eddy, W.Lorenson
"Object-Oriented Modeling and Design", Prentice Hall 1991

[Wal92] I.J.Walker, "Requirements of an object-oriented design method",
Software Engineering Journal, March 1992, 102- 113

[You89] E.Yourdon, "Modern structured analysis",
Prentice Hall 1989

Articles in the Journal of Object Oriented Programming by J.Rumbaugh covering key concepts in the OMT.

<u>Associations</u>
[Rum92a] Horsing around with associations, Feb 1992

<u>Object identity</u>
[Rum92b] A national identity crisis, Oct 1992

<u>Inheritance</u>
[Rum93a] "Disinherited! Examples of misuse of inheritance", Feb 1993

<u>The dynamic model</u>
[Rum93b] "Controlling code - How to implement dynamic models", May 1993

<u>Qualified associations</u>
[Rum93c] "What's in a name - A qualified answer", Jul/Aug 1993