A proposed object-oriented development methodology

by LeRoy R. Hodge and Mary T. Mock

The object-oriented approach to software engineering is maturing and evolving as industry methodologists strive to clarify and promote its underlying principles. The object-oriented paradigm has the potential to increase consistency within the software development process compared with previous software engineering approaches. Much of the current work, however, tends to focus on a particular phase without addressing the transition and traceability between phases. The methodology presented in this paper is proposed for the full development life-cycle. It synthesises and enhances several emerging object-oriented techniques and notations into a consistent approach. This methodology was developed to provide a framework for using object-oriented techniques in the development of a large simulation and prototyping laboratory.

1 Introduction

Software system development and maintenance continued to be challenging. Unlike hardware engineering, where formal proven techniques have been successfully adopted, software engineering continues to struggle with a number of proposed techniques (functional, structured and object-oriented techniques), all of which claim to provide a systematic methodology for the software development process. Based on their practical application, each set of techniques has demonstrated advantages and disadvantages for software system development. The object-oriented approach has the potential to provide a more consistent approach for software system development. It builds on the strengths of traditional techniques, and emphasises data abstraction, encapsulation, information hiding, inheritance, polymorphism and reuse. Given such a foundation, the object-oriented approach can provide a consistent development model with which to formalise the software development process.

In an aggressive attempt to clarify and promote object-oriented techniques, industry methodologists are advocating diverse techniques and notations for performing the analysis, design and implementation phases of the development process. This diversity has made it difficult for software engineers to adopt a consistent set of steps for the object-oriented development life-cycle. A recent informal survey of emerging object-oriented techniques did not reveal any that adequately and consistently cover the complete development life-cycle. The survey did, however, reveal many useful techniques and notations, especially those of Ward [1], Coad and Yourdon [2], Booch [3] and Bailin [4]. Although not completely object-oriented, the techniques of McMenamin and Palmer [5] also influenced the methodology described in this report. The goal of this paper is to show how these techniques and notations can be combined into a unified approach for the software development process.

The primary motivation for this proposed object-oriented methodology is the need to reduce the development risks associated with applying this new approach to the development of a large-scale high-fidelity simulation and prototyping laboratory. The laboratory is being developed to support an advanced research and development (R&D) project investigating the feasibility of introducing higher levels of automation into the en route air traffic control (ATC) system. The laboratory will be used to refine the concepts for the automated system and assist in developing the eventual specification of this system. The development risks include lack of clear and consistent guidelines for performing each phase of the development life-cycle; lack of guidelines for transitioning and showing traceability between phases; lack of automated tools to support each phase; and lack of practical experience gained by the development staff in applying object-oriented techniques to the development process. The decision to adopt the object-oriented approach was based on the potential of this approach for integrating the development process, and for providing a flexible and maintainable system. The laboratory requires a high degree of flexibility and robustness in order to provide an experimental platform for the examination, demonstration, evaluation and specifications of advanced ATC automation concepts. It was therefore imperative that a clear and consistent understanding of object-oriented techniques for the development life-cycle was outlined before embarking on the development of the laboratory.

In order to gain a common understanding of object-oriented techniques and their application to the laboratory development, the development staff performed a short-term exercise; this exercise and its results are described in Reference 6. The methodology described in this paper resulted from this exercise, and the remaining figures in this paper are based on documents developed during the exercise.

The resultant methodology portrays the system from multiple perspectives, especially in the analysis and design phases. This provides greater understanding of the system content (an information view) and its behaviour (an interactive view). The methodology also emphasises the need to validate the various phases of the development process, and provides the flexibility to accommodate an iterative

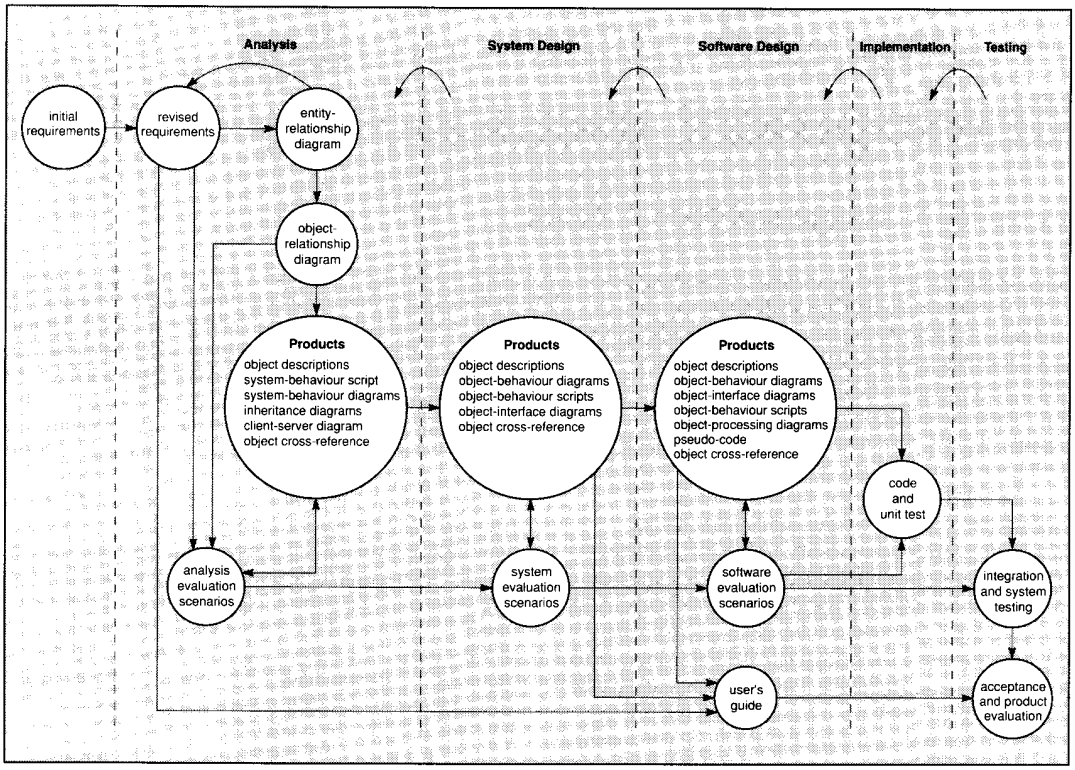

Fig. 1 Process and methodology overview

This approach defines five phases for object-oriented development: analysis, system design, software design, implementation and testing. The implementation and testing phases are not discussed in this paper; testing and implementation (object-oriented programming) are addressed elsewhere.

and cyclic development process. In addition, it highlights the need for a sophisticated CASE tool to facilitate the description and examination of the system from various points of views, and also to provide an infrastructure for maintaining consistency across these views. This methodology will be used for laboratory development and will be evaluated during this development. Modifications to the methodology, based on experience with laboratory development, are anticipated.

2 Background

The MITRE Corporation is currently conducting an intensive R&D effort investigating the feasibility of introducing higher levels of automation into the en route ATC system. This multi-phased project is sponsored by the Federal Aviation Administration (FAA) and is called the Automatic En Route Air Traffic Control (AERA) program. The most advanced phase of this programme, Advanced AERA Concepts, is sponsored by the Research and Development Service of the FAA. This programme seeks to determine the limits to which automation can be usefully extended in the en route ATC process. A primary motivation for the AERA programme is the growing need to manage more effectively, efficiently and safely the increasing traffic levels and diversity of aircraft capabilities. The primary goals of the AERA programme are to increase airspace user benefits, and to make significant improvements in ATC system safety, throughput, economy and productivity.

Advanced AERA proposes three major changes to the ATC system. The first is automation of the aircraft separation function currently performed by the human ATC specialist. Second is the utilisation of expected improvements to aircraft avionics, surveillance and communications equipment. The third change is that the role of the human ATC specialist will change, from being primarily responsible for aircraft separation to the management of airspace capacity and throughput in the en route airspace.

As part of this investigation, the AERA Protocenter, a large-scale high-fidelity laboratory, is being developed to prototype and simulate key functional and operational characteristics of the proposed ATC system and the future environment in which it will operate. Key components of the envisioned ATC system and the anticipated ATC environment will be modelled in the Protocenter, alternative automation approachs will be explored, the proposed role of the human ATC specialist will be defined and evaluated, and the integration of human and automation functions will be demonstrated and reviewed. The Protocenter software will be developed over several years as a series of releases, with each release adding a significant increase in functionality. The requirements for each release will be based on the evaluation of the previous release and the goals and objectives for that release. The evaluation of each release will also contribute to the preparation of the specification of the end-state Advanced AERA system. This specification, as it is refined, will also influence the formulation of the goals and objectives for each release.

The Protocenter will comprise a mix of hardware and software components operating in a distributed multi-

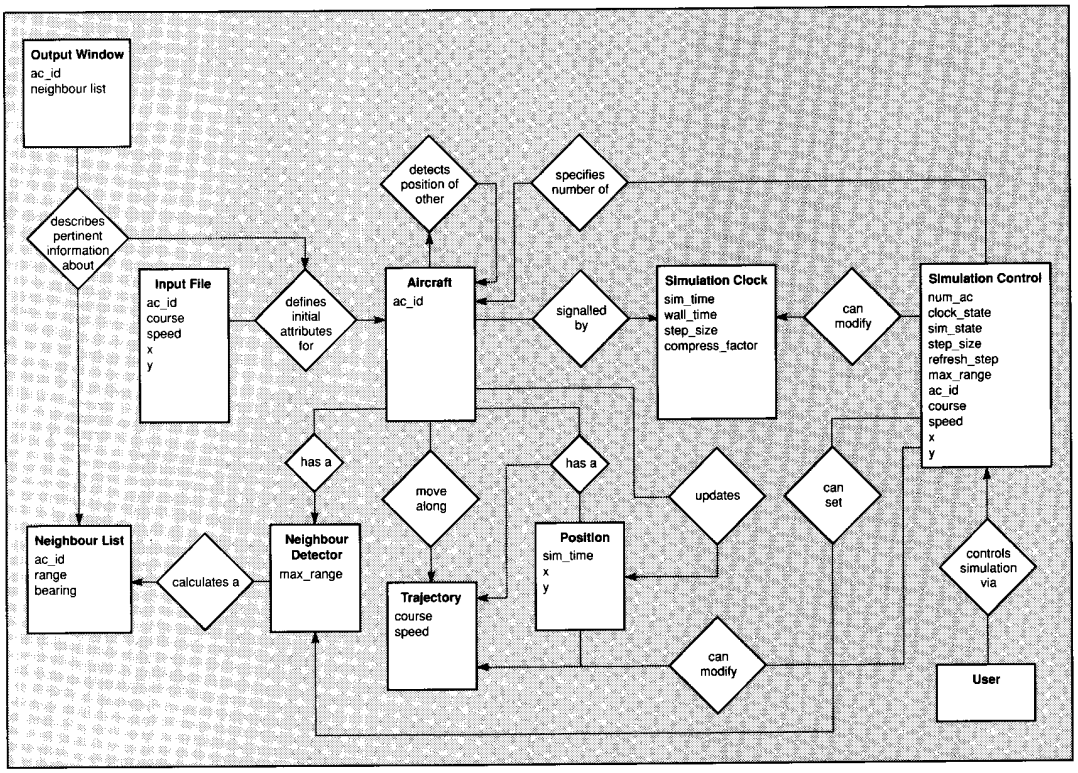

Fig. 2 Entity-relationship diagram (ERD)

The ERD is a working model and is used in conjunction with both the requirements analysis and event analysis activities to determine what is the problem domain. The ERD is a flat model; no attempt is made to show any hierarchical structure.

tasking environment. The development of such a sophisticated software system represents a significant software engineering challenge, and mandates the adoption of a disciplined, yet flexible, software development process and methodology. The object-oriented approach offers many advantages over traditional software engineering approaches and was therefore chosen as the development model for the AERA Protocenter.

3 The proposed object-oriented development methodology

The proposed development process and methodology is a synthesis of techniques and notations from several sources. This methodology, however, offers more than just the sum of its parts. It provides traceability and continuity through the development life-cycle, primarily since documents and diagrams produced at early phases are refined and enhanced in later phases. Another benefit is that it provides the basis for a smooth transition from one phase to the next. It also provides multiple perspectives of the system under development, increasing understanding on the part of analysts, developers and implementers. Finally, it provides a mechanism for validating the models developed at each phase to verify that phase's products.

As shown in Fig. 1, this approach defines five phases for object-oriented development: analysis, system design, software design, implementation and testing. The implementation and testing phases are not discussed in this report; testing and implementation (object-oriented programming) are addressed elsewhere [6, 7]. These phases are similar to those defined in traditional development approaches. The right-to-left arrows in Fig. 1 indicate that the development process involves cycling between phases; returning to previous phases for refinement or reorganisation is expected.

The analysis phase considers the system as a solution to a problem in its environment or *domain*. During this phase, fundamental objects in the system and its domain are identified. The relationships between these objects, as well as the attributes and fundamental behaviour of these objects, are identified. Implementation considerations are not discussed; unlimited processing and storage are assumed at this phase. As the analysis phase nears completion, *analysis evaluation scenarios* are produced to validate the model prepared during this phase. The analysis evaluation scenarios are simple situations prepared to determine if the analysis model adequately covers all the fundamental requirements of the system.

The system design phase focuses on how the system works. During this phase, two major activities occur. The first is specifying details of the behaviour of system objects and the interaction of these objects. This is a refinement of objects identified during analysis. The second activity is identifying additional objects that are required to make the system work. Like analysis, system design is performed without considering implementation. In addition, evaluation scenarios are prepared to validate the system design. *System evaluation scenarios* are refined versions of the scenarios prepared for the analysis phase, showing that all

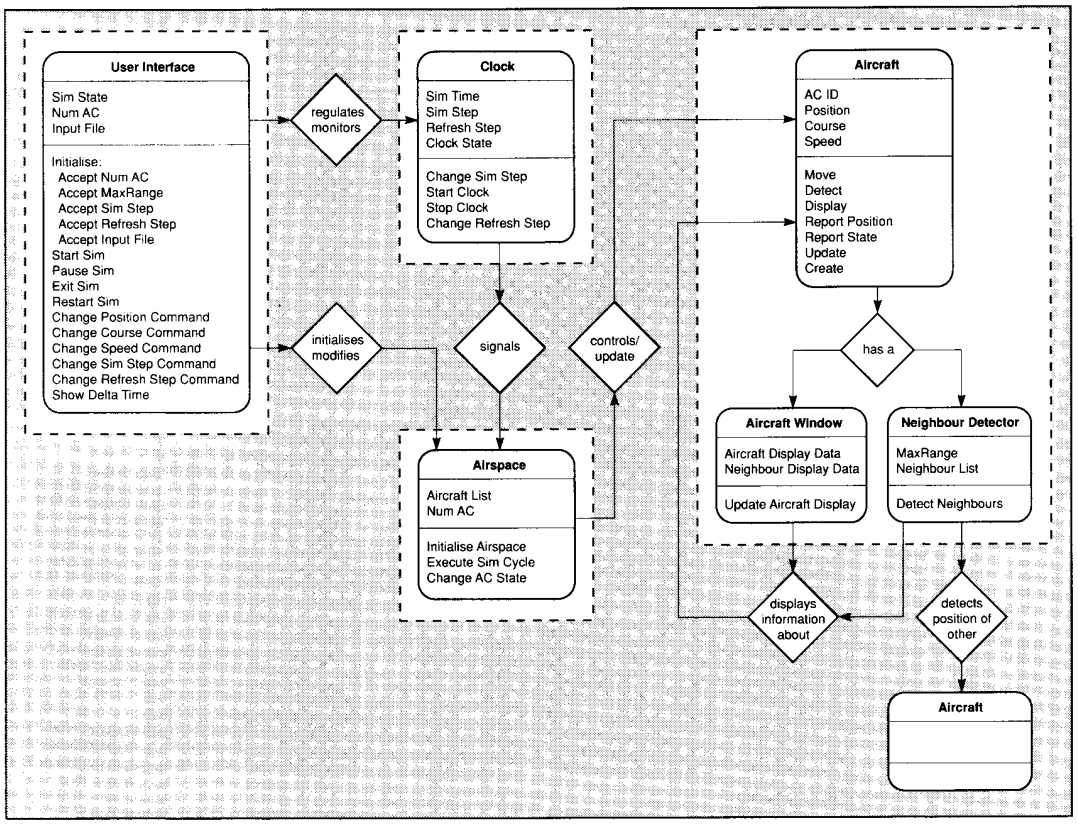

Fig. 3 Object-relationship diagram (ORD)

Data elements in the ERD will become either an object or an attribute of an object in the ORD. This model shows the objects, their attributes, sub-objects, their behaviour (also called *services* or *methods*) and the relationships between the objects. The objects shown in the ORD can be consolidated into *subjects*. Subjects are groups of objects that work together to provide related services within the system.

mechanisms necessary to perform the system requirements are present in the system design model.

The software design phase addresses how this system will be implemented with a specific programming language on a particular hardware platform and software suite. The activity in this phase is primarily adding implementation information to the objects defined in the analysis and system design phases. As before, evaluation scenarios are prepared to validate the software design. These *software evaluation scenarios* are refinements of the system evaluation scenarios and provide a complete set of system testing data. Implementation is expected to be straightforward from the software design products.

Frequent reviews are essential to any system development effort, with any approach and methodology. As a minimum, a review is needed at the completion of the analysis, system design and software design phases. The evaluation scenarios play a major role in these reviews.

The following describes the phases and the corresponding documentation. The documentation is in the form of textual descriptions and diagrams. Samples are contained as figures; these figures are based on those produced during the development exercise.

3.1 Analysis phase

The analysis phase focuses on identifying the fundamental objects that are contained within the system and its problem domain, and the relationships between those objects. Analysis answers the question: 'what is present within the system and its domain?' In answering this question, three techniques common to traditional development are used: requirements analysis, information analysis and event analysis. Requirements analysis identifies ambiguities and deficiencies within the requirements statement, and results in a revised requirements statement. Information analysis identifies the information (or data) contained within a system. In object-oriented development, this means identifying objects and their attributes. Event analysis identifies the behaviour of objects within the system. Requirements, information and event analysis are highly inter-related, and their output can provide the basis for gaining agreement with the system client on the scope and contents of the system. Although these activities may be performed sequentially, it is expected that they will be performed concurrently and that the results of each activity will confirm the results of the other activities. The result of the analysis phase is a model of the system within its domain, where that model consists of objects, their attributes and behaviour, and relationships among objects.

3.1.1 Requirements analysis: the goal of requirements analysis is to develop a clear statement of the system purpose, scope and required functionality. Ideally, this is provided by the client. In reality, this statement is often not

Purpose:	The Aircraft Object simulates the motion of the aircraft along a trajectory and provides operations that can modify the non-identifying attributes. The public services provided initialise the object, report an aircraft's position, move the aircraft, detect neighbours, display aircraft information and neighbour list, update attribute values and return attribute information. The private services perform the updating of aircraft attributes.
Sub-object of:	{ }
Sub-objects (*has_a relationship*)	Neighbour Detector Aircraft Window
Inheritance/Superclass (*is_a relationship*)	{ }
Attributes	AC ID Position Course Speed *Neighbour_List *AC_List
Services Provided	Create( ) Move( ) Detect( ) Display( ) Report_Position( ) Report_State( ) Update( )
Services Required	Detect ⇒ (*Neighbour Detector Object*) Display ⇒ (*Aircraft Window Object*)
Service Defintions	*Will be added during System Design phase*

Fig. 4 Object description (OD)

The OD is a textual description of each object. It serves as a repository for the detailed characteristics of each object in the ORD. The ODs are core documents that are progressively updated and expanded through the development life-cycle.

provided, or if it is, it is ambiguous and inconsistent. In addition to required functionality, the requirements statement must include all client concerns and constraints, such as performance requirements.

Two key questions that the analyst must ask in performing requirements analysis are

- what do I need to know about the problem domain?
- what and who are sources of information on the problem domain?

In answering the first question, the analyst identifies the scope of the system. Answers to the second question may include the client, the analyst's own experience and other sources unique to the problem domain. The product of requirements analysis is a statement of purpose, scope and functionality, and typically will be a refinement of the original requirements statement. (More detailed discussions of requirements analysis can be found in Reference 8.)

3.1.2 Information analysis: the goal of information analysis is to model entities in the problem domain and the relationships among these entities. Entities are the underlying data elements that describe what is in the problem domain. The initial goal is simply to identify which data elements are in the problem domain and to establish relationships between these elements. Since information analysis is an investigative process, the analyst should use all available sources to identify entities with the problem domain.

This model is represented graphically in an entity-relationship diagram (ERD) (Fig. 2). The ERD is also called an information model. It is primarily a flat model; no attempt is made to show any hierarchical structure. The ERD is a working model and is used in conjunction with both the requirements analysis and event analysis activities to determine what is in the problem domain.

Once entities have been identified and represented in the ERD, the next step is to translate them into objects. This is done by determining what object each entity best describes and ascribing the entity to that object. The question to be asked is: what object does this data element best describe? The data element becomes either an object or an attribute of an object. This is represented in the object-relationship diagram (ORD), shown in Fig. 3. This model shows the objects, their attributes, sub-objects, the behaviour that they perform (also called *services* or *methods*) and the relationships between the objects.

The objects shown in the ORD can be consolidated into *subjects*. Subjects are groups of objects that work together to provide related services within the system. This higher level grouping further decomposes the problem domain into manageable units, and thereby provides a hierarchy. Subjects are indicated by the dashed lines around groups of objects (Fig. 3).

A textual description of each object, called the object description (OD), details the characteristics of each object in the ORD, as shown in Fig. 4. The ODs are core documents that are progressively updated and expanded through the development life-cycle. Another document to be updated and expanded through the development life-cycle is the object cross-reference (OCR), as shown in Table 1. The OCR is a tabular form of all objects, the services they perform and the services they utilise from other objects. As a concise representation of the objects, the OCR is expected to be most valuable in subsequent main-

tenance activities. The OCR is derivable from the ODs and ideally would be automatically, rather than manually, generated.

Finally, as objects and their attributes and services are identified, inheritance relationships between object classes may be observed. Inheritance is a generalisation–specialisation relationship, where the superclass is the generalised version of the more specialised subclass. These relationships are shown in an inheritance diagram, as shown in Fig. 5. (A more detailed discussion of information analysis and information modelling can be found in References 1 and 9.)

3.1.3 Event analysis: like information analysis, event analysis serves to identify data elements in the problem domain and then consolidates them into objects. In performing event analysis, the analyst views the system as a stimulus–response machine that responds to a series of external stimuli. The goal is to capture system behaviour from an external viewpoint. Event analysis can be performed in conjunction with information analysis or separately as a validation exercise for the analysis phase. In this methodology, event analysis is expected to be used primarily as validation.

Given a statement of the system's primary purpose, all activities that the system will perform in response to external stimuli can be determined. From this list of activities, the analyst must identify the activities that are fundamental to the system and that the system must perform in direct support of its stated purpose. These activities are classified as *fundamental activities* because without them the need for the system ceases to fulfil its primary purpose. All other activities that are performed in support of the fundamental activities are classified as *custodial*. These stimuli and response activities are listed in the system-behaviour scripts (SBS), as shown in Table 2.

It is important to identify the overall system behaviour from an external viewpoint. This is represented by a system-behaviour diagram (SBD) (Fig. 6). The SBD is a state-transition diagram that captures the externally visible states of the system, as well as the possible transitions

Table 1 Object cross-reference (OCR)

Object	Services Provided	Services Required	Object Providing Service
Aircraft	Move		
	Create		
	Report Position		
	Report State		
	Display	Display	Aircraft Window
	Detect	Detect	Neighbour Detector
	Update		
Aircraft Window	Create		
	Display		
Neighbour Detector	Create		
	Detect	Report Position	Aircraft
Clock	Change Sim Step		
	Change Refresh Step		
	Start Clock	Report Clock State	Clock
	Stop Clock	OK to Tick	Clock
User Interface	Create	Update Clock	Clock
	Setup	Update State	Airspace
	Exit		
	Start		
	Pause	Execute Sim Cycle	Airspace
	Resume	Display	Sim Perf Window
Clock	Create		
	Update Clock	Display	Sim Perf Window
	OK to Tick		
	Report Clock State		
Sim Perf Window	Display		
Airspace	Create	Move	Aircraft
	Execute Sim Cycle	Create	Aircraft
	Update State	Report State	Aircraft
	Report	Display	Aircraft
		Detect	Aircraft
		Update	Aircraft

The OCR is a tabular form of all objects; it shows the services they perform and the services they utilise from other objects. As a concise representation of the objects, the OCR is expected to be most valuable in subsequent maintenance activities. The OCR is derivable from the ODs and would ideally be automatically, rather than manually, generated. The OCR is updated and expanded through the development life-cycle.

between those states. The stimuli that trigger state transitions may be the same as the stimuli listed in SBS.

The next step is to identify the data elements that arrive with each stimulus and that comprise each response. Most of these data elements should have already been identified during information analysis, and thus will confirm the validity of that set of entities. Newly identified entities should be added to the ERD as identified.

As in information analysis, once the data elements have been identified, the next step is to consolidate these data elements into objects. This process is the same as in information modelling and again should complement the results of information analysis. Thus, the ORD should be modified to include any newly identified objects.

In identifying the objects from a stimulus–response perspective, the analyst has identified which object performs which activities in the system. This associates system behaviours with specific objects. Object behaviour is listed in the OD. (A more detailed description of event analysis can be found in Reference 5.)

3.1.4 Transition to system design: the final product of domain analysis is the client-server diagram (CSD) (Fig. 7). The CSD provides a more detailed view of the interaction between objects in the system than that shown in the ORD. As such, the CSD provides a transition between analysis and system design. The direction arrows point from client to server, the multiplicity (or n-ary) relationships between objects are shown, subject groupings are shown and messages between objects are shown.

It is important to note that this development approach is a cyclic process; returning to previous phases for refinement or reorganisation is expected. It is also important to note that analysis should be a rigorous and complete

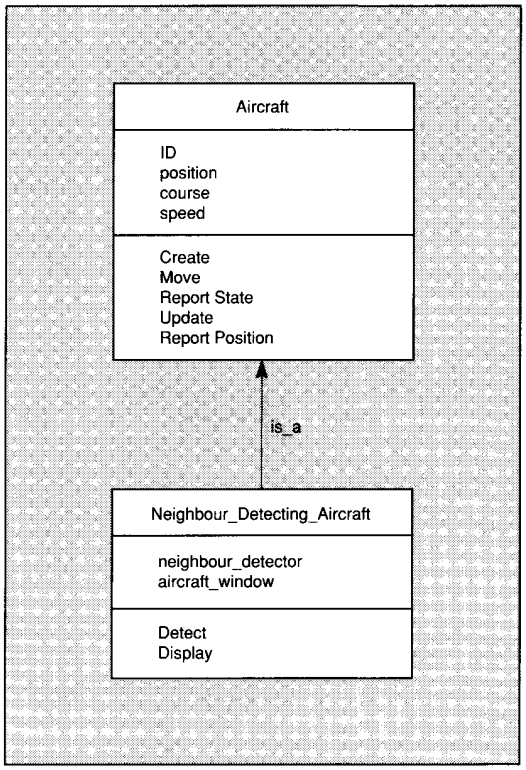

Fig. 5 Inheritance diagram (ID)

The ID shows the generalisation–specialisation (inheritance) relationship between object classes, where the superclass is the generalised version of the more specialised subclass.

Table 2 System-behaviour script (SBS)

Purpose of the System (Raison d'être)

The purpose of the system is to serve as a simulation system to demonstrate an airborne air traffic control (ATC) concept, in which individual aircraft have the capability of detecting neighbouring aircraft and displaying relevant information.

Activities

Stimulus	**Response**
Fundamental Activity	
Next Simulation_Cycle Signal	Perform work for Simulation Cycle – Update AC position – Detect neighbouring aircraft – Update Display
Custodial Activities	
Next Refresh Cycle Signal	Update Display with AC information
User Command – Setup – Update – Start – Pause – Restart – Exit	Perform appropriate action(s) for User Command – Setup initial simulation characteristics – Modify simulation characteristics

The SBS depicts the system as a stimulus-response machine that responds to a series of external stimuli. The SBS describes, at a high-level, the system's response to each stimulus. Given a statement of the system's primary purpose, all activities that the system performs in response to external stimuli can be determined. Activities that are fundamental to the system and that the system must perform in direct support of its stated purpose are classified as *fundamental activities*. All other activities that are performed in support of the fundamental activities are classified as *custodial*.

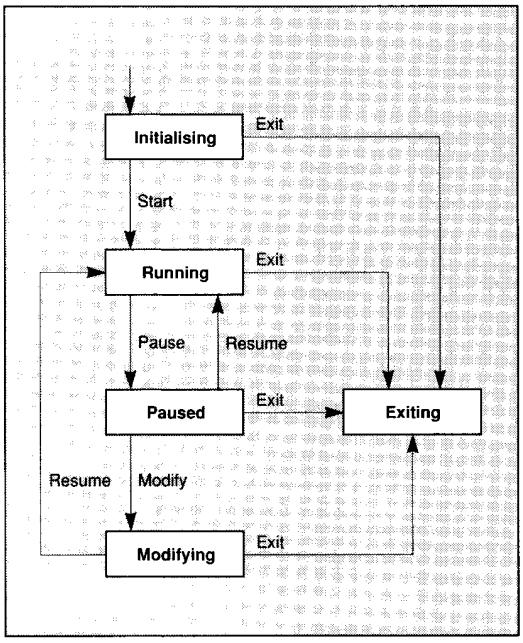

Fig. 6 System-behaviour diagram (SBD)

The SBD is a state-transition diagram that captures the externally visible states of the system, as well as the possible transitions between those states.

process. By viewing the system from both information and event perspectives, the analyst is more confident that the problem domain is being properly decomposed. This is especially important because analysis sets the framework for design and implementation. This is a strategic point at which to hold a review with the customer.

3.2 System design

The system design phase focuses on defining how the system works. During this phase, unlimited computing and storage are assumed, and so implementation details are not addressed. Two primary activities occur during this phase; identifying additional objects that are specific to the system design, and refining the objects that were identified during analysis. A linked list is an example of an object that is specific to system design, but would not have been identified during analysis. In the simple Figures, an aircraft object is a fundamental object in the system and was identified during analysis. A linked list object could be used to manipulate a collection of aircraft instances. In this situation, the linked list is not a fundamental object, but serves a purpose in defining how the system works, and thus is specific to system design.

In object-oriented development, the way that the system works is determined, to a large degree, by the interaction of the objects. Defining this interaction clearly and completely is a major goal for the system design phase, and it is the primary way in which the objects identified during

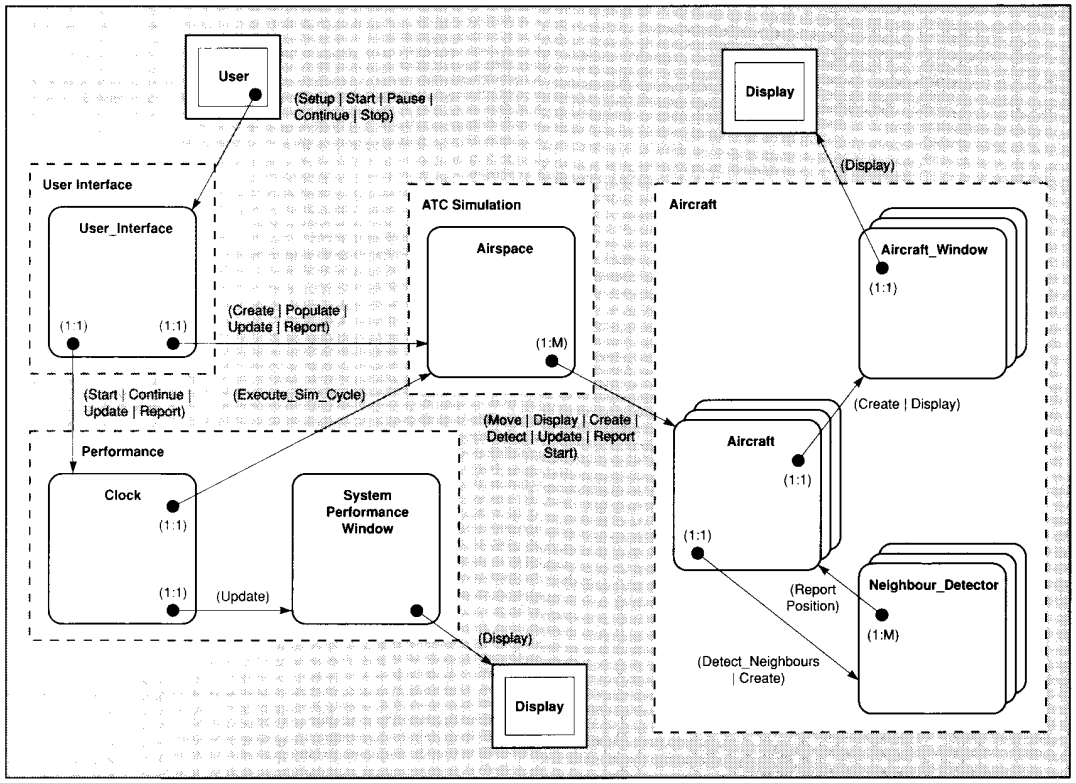

Fig. 7 Client-server diagram (CSD)

The CSD provides a more detailed view of the interaction between objects in the system than that shown in the ORD. As such, the CSD provides a transition between analysis and system design.

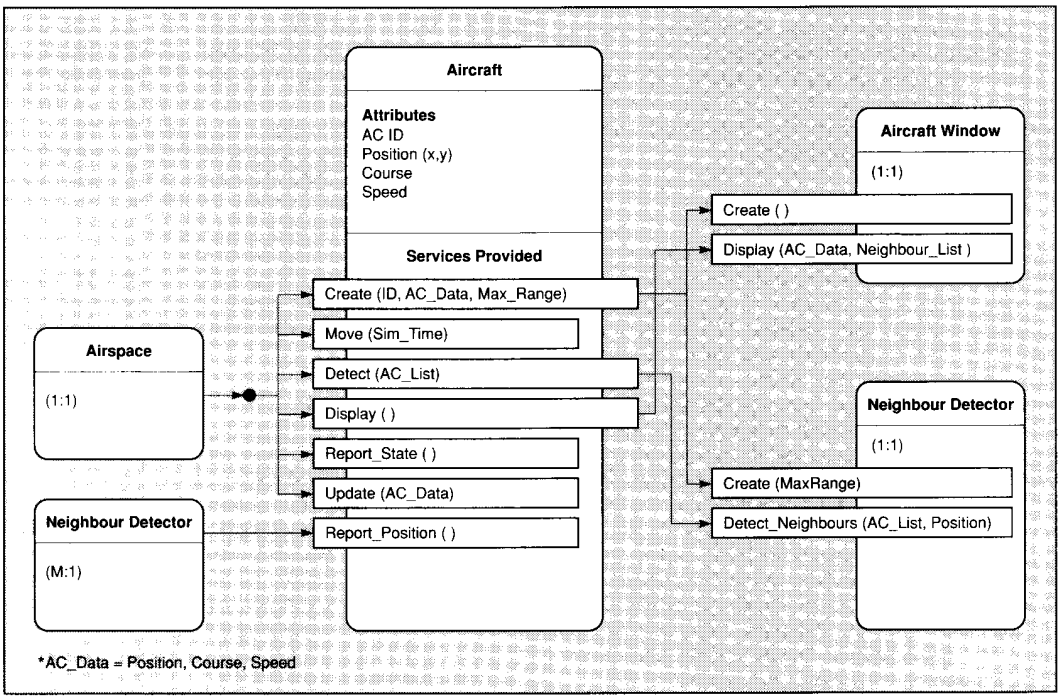

Fig. 8 Object-inferface diagram (OID)

The OID depicts the interaction of objects from the perspective of an individual object. The OID would be typically derived from the CSD, which shows object interaction for the system. The CSD shows object relationships at a system level; the OID increases the level of detail and focuses on a single object, rather than the full system.

analysis are refined during this phase.

The object-interface diagram (OID) represents this interaction of objects from the perspective of an individual object. The OID, shown in Fig. 8, would typically be derived from the CSD, which shows object interaction for the system. The CSD shows object relationships at a system level; the OID increases the level of detail and focuses on a single object, rather than the full system. Objects that access the target object's services are shown, as well as other objects' services that are accessed by the target object. The OID provides a complete view of an object's interaction with other objects. During analysis, system behaviour was represented by the system-behaviour diagram and system-behaviour script. Analogous representations of individual object behaviour are shown in the object-behaviour diagram (OBD) and object-behaviour scripts (OBS). Since not every object has interesting states, an OBD will not be necessary for every object. The ODS is shown in Table 3. Since the OBD is similar to the SBD (Fig. 6), its system-level counterpart produced during analysis (a sample OBD) is not shown here.

The OD and OCR produced during analysis are refined during system analysis, both to include more information about object behaviour. The OD is expanded to include a *service definition* for each service provided by the object. The service definition includes input and output parameter names and types. As in analysis, the OCR is a condensed tabular representation of information about all objects; at system design, it now includes columns for services' parameters. In addition, the OCR is expected to be automatically generated.

3.3 Software design

The software design phase refines the products of the system design phase to address the implementation-specific details of the proposed system. The target hardware platform and software suite are addressed, although specific implementation language details are not.

The OD, OBD, OID, OBS and OCR may be expanded from their counterparts from system design. However, which are needed for software design depends largely on the specifics of the system and the development platform. In general, if implementation details are relevant for a particular diagram, that diagram is refined; otherwise, the diagram produced during system design is used without modification. For many objects, the OBD, OID and OBS do not need to be refined for software design. The OD is expected to be refined for all objects and expanded to include information such as operating system calls to be used, and which methods are *public* and *private*. A private method is intended for use only by that object, whereas a public method is available for use by other objects. Identification of public and private methods is useful for software design, even when the implementation language does not support this distinction. Units of measure for the object attributes should also be included in the refined OD and reviewed for consistency within groups of communicating objects.

The object-processing diagram (OPD) is used to represent the internal processing required for an object. It shows public and private methods, as well as which methods read and set the object's attribute values. Input

and output parameters are also shown for each method. An example of an OPD is shown in Fig. 9.

Pseudo-code may be used for objects with complex algorithms or those that have especially complicated aspects to their implementation. This would again be subject to the designers' judgment.

4 Conclusions

This methodology provides a straightforward approach to object-oriented system development with four major benefits. First, effort is concentrated in the analysis phase, decreasing through latter phases. This is expected to reduce the cost of identifying and fixing the inevitable errors in the system, and thereby reduce the overall system development effort. Secondly, the system is viewed and represented from multiple perspectives; as the overall system, as individual objects and interacting objects. This results in documentation that is readily understood by the appropriate person, whether that is the customer, analysts, designers, programmers or testers. Thirdly, the effort and documentation in latter stages are based largely on those in earlier stages. This results in traceability throughout the complete development cycle. Finally, by relying on the analysts' and designers' judgment, especially in latter stages, the methodology is flexible in order to accommodate different problem domains and systems.

The multiple perspectives of the system under development can provide a thorough understanding of both the system content and its behaviour. It is important to point out that not all views (and supporting documentation) are required for all objects in the system. These views should be used as appropriate. However, by developing these various views early in the development process, a cleaner and more consistent design and implementation can be achieved. The need for a sophisticated CASE tool to facilitate the description and examination of the system from various points of view, and also provide an infrastructure for maintaining consistency across these views, is highlighted by this proposed methodology.

The methodology also emphasises the need to validate the various phases of the development process, and provides the flexibility to accommodate an iterative and cyclic development process.

5 Acknowledgments

The authors would like to thank others who participated in developing this methodology: Randy Crawford, Tom Hsiao, Mark Krause, Pete Lane, John McCarron, Art McClinton, Jim Reierson, Dwight Shank, Ghina Siddiqui and Frank Sprague. We would also like to thank Robert N. Leggett and Anne Deslattes for their thorough review and excellent suggestions, and our management and sponsor for encouraging and supporting this work.

6 References

[1] WARD, P.T.: 'The CASE real-time curriculum'. Software Development Concepts, New York, 1989

Table 3 Object-behaviour script (OBS): aircraft

Message	Behaviour Script
Create (ID, Position, Course, Speed, Max_Range)	Instantiate aircraft and initialise values: – set initial position (x, y) – set initial course – set initial speed Instantiate sub-objects – aircraft window – Neighbour Detector (Max_Range)
Move (Sim_Time)	Calculate new aircraft position based on current Sim_Time/Sim_Step
Update (Position, Course Speed)	Reset appropriate AC state data: – save new AC position (x, y) – save new AC course – save new AC speed
Report_State( )	Return the values of the AC state data
Display( )	Get neighbour data from my neighbour detector (send Report_State message) Forward Display message to sub-object Aircraft Window: – i.e., display (Position, Course, Speed, Neighbour List)
Detect (AC_List)	Forward Detect message to sub-object Neighbour Detector – i.e., call my neighbour detector passing it a list of all aircraft in the airspace. It will, in turn, query each AC for position and compare with MaxRange to determine if AC is my neighbour
Report_Position( )	Return AC position (x, y)

The OBS depicts an object as a stimulus–response machine that responds to a series of external stimuli (invocation of its methods). The OBS presents, at a high-level, an object's response to each stimulus.

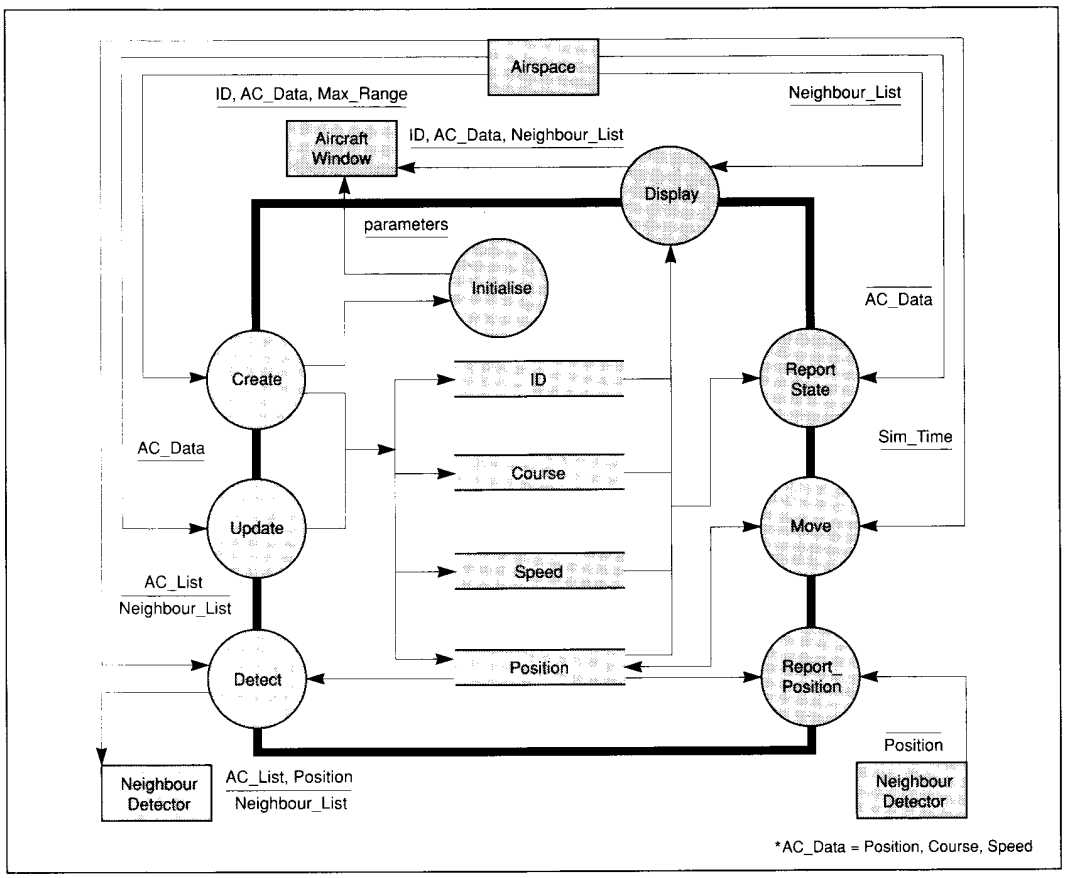

Fig. 9 Object-processing diagram (OPD)

The OPD is used to represent the internal processing required for an object. It shows public and private methods, as well as which methods read and set the object's attribute values. Input and output parameters are also shown for each method.

[2] COAD, P., and YOURDON, E.: 'Object-oriented analysis' (Prentice Hall, Englewood Cliffs, New Jersey, 1990)

[3] BOOCH, G.: 'Object-oriented design with applications' (Benjamin/Cummings, Redwood City, California, 1990)

[4] BAILIN, S.: 'An object-oriented requirements specification method', *Commun. ACM*, 1989, **32,** (5), pp. 608–623

[5] MCMENAMIN, S.M., and PALMER, J.F.: 'Essential systems analysis' (Yourdon Press, New York, 1984)

[6] MOCK, M.T., and HODGE, L.R.: 'An exercise to prototype the object-oriented development process'. The MITRE Corporation, McLean, Virginia, 1991

[7] MYERS, G.J.: 'The art of software testing' (John Wiley & Sons, New York, 1979)

[8] COX, B.J.: 'Object oriented programming: an evolutionary approach' (Addison-Wesley, Reading, Massachusetts, 1986)

[9] GAUSE, D.C., and WEINBERG, G.M.: 'Exploring requirements: quality before design' (Dorset House, New York, 1986)

[10] SCHLAER, S., and MELLOR, S.J.: 'Object-oriented systems analysis: modeling the world in data' (Yourdon Press, Englewood Cliffs, New Jersey, 1988)

7 List of abbreviations

AERA = Automated En Route Air Traffic Control
ADT = attribute definition table
ATC = air traffic control
CASE = computer-aided software engineering
CSD = client-server diagram
ERD = entity-relationship diagram
FAA = Federal Aviation Administration
OBD = object-behaviour diagram
OBS = object-behaviour script
OCR = object cross-reference
OD = object-description
OID = object-interface diagram
OPD = object-processing diagram
ORD = object-relationship diagram
R&D = research and development
SBD = system-behaviour diagram
SBS = system-behaviour script
SRM = stimulus-response machine

The paper was first received on 4 March and in revised form on 26 September 1991.

The authors are with the Center for Advanced Aviation System Development, The MITRE Corporation, 7525 Colshire Drive, McLean, Virginia 22102-3481, USA.