Object-oriented systems development: survey of structured methods

A G Sutcliffe

Concepts of object-oriented system programming and system design are reviewed in the light of previous research on systems development methodologies. Key principles are identified and a selection of system development methods is then judged against these principles to determine their concordance with object-oriented design. The advantages of object-oriented system development are reviewed in the light of the study of structured system development methods.

object-oriented, object-oriented systems, structured methods, systems analysis and design

Object-oriented programming (OOP) has been the subject of several studies[1-3] that describe the principles of the object-oriented (OO) approach and their incorporation in the new generation of programming languages such as C++, Eiffel, and Smalltalk. In contrast, the object-oriented approach has received little attention in studies on system development methods. This paper aims to redress that balance and explore how OO concepts are being integrated into structured systems development methods.

Apart from the extensive interest in OOP languages, OO approaches have received some attention in office automation[4,5]. More recently, several methods have appeared claiming to be 'object-oriented' (OOSA (Object-Oriented Systems Analysis)[6], OOA (Object-Oriented Analysis)[7], and HOOD (Hierarchically Object-Oriented Design)[8]. As yet object-oriented system (OOS) development methods are not in widespread commercial practice, although interest in OO concepts continues to grow. One unanswered question is what are the essential differences between OO methods and those from the more classical 'structured camp', e.g. Structured Systems Analysis and Design Method (SSADM), Jackson System Development (JSD), and Structured Analysis/Structured Design (SA/SD). If OO methods are to become accepted, the advantages over and differences from previous methods have to be established and then the implications of migration paths from current techniques to OO methods should be made clear. This paper aims to throw some light on these questions by examining how current system development methods fit criteria for OO development.

First, OO concepts are described within the context of system development, then a selection of system development methods is reviewed.

OBJECT-ORIENTED CONCEPTS

OO development is claimed to improve software design for reliability and maintenance. Further claims are that the development process is made more efficient by reuse. The justification for these claims rests on three principles: abstraction, encapsulation, and inheritance.

Abstraction

OO approaches have been based on modelling structures in the real world. Programming languages that facilitate this modelling and support its implementation are said to create more maintainable and reliable systems with reusable program components[3].

Objects are an abstraction of parts of real-world systems and model composite units of structure and activity. Cook[3] points out that there are two roles that objects fulfil: an implementation role related to improving the maintainability of programs, and a modelling role, which addresses the problems of correct specification of system requirements. OOS development should emphasize the latter role, while supplying the necessary specifications to enhance maintainability in implementation.

Encapsulation

Encapsulation is the concept that objects should hide their internal contents from other system components to improve maintainability. By making part of the design local, objects limit the volatility of change in the system. The encapsulated parts of objects are hidden to insulate them from the effects of system modifications.

Inheritance

Objects should have generic properties, i.e., support reusability by property inheritance from superclass to subclass[3]. By organizing objects in class hierarchies, lower-level objects can receive properties from higher-level objects. This facilitates reuse of more general, higher-level objects by specialization.

Two forms of inheritance may be supported: hierarchi-

Department for Business Computing, School of Informatics, The City University, Northampton Square, London EC1V 0HB, UK

Reprinted from *Information and Software Technology*, Vol. 33, No. 6, July/August 1991.

cal, in which a child object can inherit only from its parent object, or multiple, when an object can inherit properties from several parent objects. Multiple inheritance may result in 'polymorphism', with one component having different properties in several new locations, as it is specialized in child objects.

These principles contribute to the OO model of systems, which is composed of a network of objects communicating by messages. Each object specifies both data and activity and may share properties according to a classification hierarchy. To enable comparison of methods, the basic principles of the OO approach need to be situated in a comparative framework that addresses not only OO concepts, but also more traditional models of structured methods. The ISO meta-schema (ISO TC97)[9] is taken as a starting point.

Evaluation of modelling components

The first question to resolve is what is an object, and what is the difference between objects and more traditional concepts such as entities and functions. The starting point may be taken from the entity definition given in the ISO TC97 report[9]:

> Any concrete or abstract thing of interest including association among things.

The ISO report makes distinctions about entities on three levels:

- Entity instances — the actual occurrence of one example of an entity type.
- Entity type — a type defined by a set of common properties to which all instances belong.
- Entity class — all possible entity types for which a proposition holds, i.e., the set of instances for a particular entity type.

These definitions accord with the OO approach. Besides entities, the other system components recognised by the ISO report are propositions (i.e., rules), constraints, which specify the behaviour of entities, and events, which are defined as 'The fact that something has happened in either the universe of discourse, or the environment or in the information system'. Events are modelled as messages in the OO approach, i.e., messages communicate events to which objects respond. Objects record states, i.e., an unchanging reality altered by transitions from one state to another, and react to events by changing state[10]. Events are modelled as messages passed within a network of objects, and thereby controlling their behaviour[3,10]. Rules, however, are more problematic.

The ISO separation of entities representing data structures from rules specifying control does not match the OO concept because objects specify a composite of data and activity. In the ISO meta-model, entities are not considered to possess attributes, instead attributes are regarded as entities in their own right. This is contrary to OO approaches in which attributes are components of objects. Furthermore, the ISO view of relationships does not fit the OO conceptualization of relationships between objects being either caused by events or specified in terms of a classification hierarchy.

Object orientation, therefore, shares many of the ISO concepts, but by no means all. The main point of divergence is the separation of activity and data specification, a point that re-emerges when individual methods are considered. Within the perspective of systems development, the convergence of objects and traditional concepts may be summarized as:

- Objects are close to the entity concept, i.e., something of interest defined by a collection of attributes, although objects add activity to the entity.
- Objects are a type with one or more instances of the type, essentially the same as the entity-type concept.
- Objects instances may be changed by events in the outside world or within the system and record a state resulting from change.

Objects may have more or less activity associated with them. At one extreme are data-oriented objects, which undergo no operations other than simple updates to their attributes. In contrast, a task-oriented object may possess few data items and much complex algorithmic processing. An example of the latter is a mathematical calculation in an engineering system.

Given that objects may show variable structures and properties, a useful classification is given by Booch[10], who divides objects into actors, agents, and servers. Actors are objects that perform actions which influence other objects in the system, and have similarities with tasks and procedures; servers are the recipients of an actor's activity and are related to the database entity concept; and, finally, agents are an amalgam of both characteristics. In practice, the mix of object types within a system will reflect the application, e.g., real-time systems will have more actors, whereas data retrieval systems will have more servers.

So far the components of an OO model have been contrasted with more traditional concepts. However, conceptual models are only one facet of methods. The next section develops the comparison from modelling features into an evaluation framework.

EVALUATION PROCEDURE

A meta-model of OO development is illustrated in Figure 1, summarizing the components of OO conceptual models, the principles of the approach, and the OO conceptualization of the development life-cycle. Methods should advise practitioners how to proceed as well as giving them the tools with which to analyse and design systems. Four dimensions are used in the evaluation framework:

- Conceptual modelling: the method should contain a means of modelling applications, and in the perspective of this study, the model should meet OO criteria.

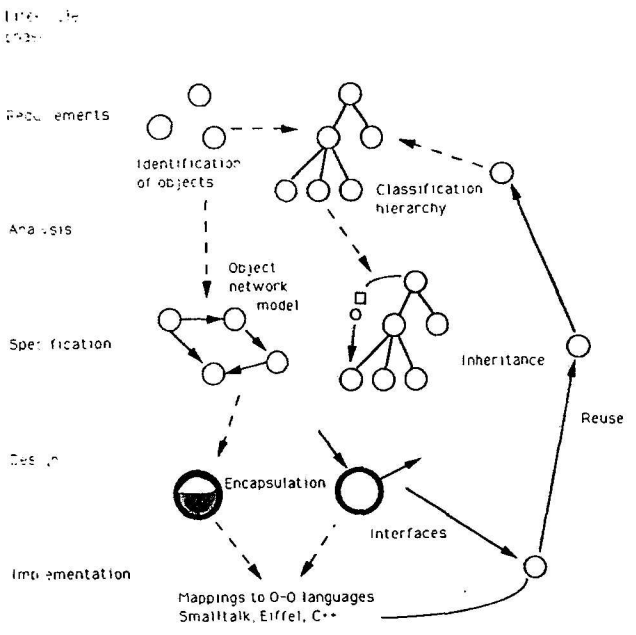

Figure 1. Summary of object-oriented meta-model

- Procedural guidance: a method should have clear steps telling the analyst how to conduct analysis, specification, and design.
- Transformations: methods should give heuristics, rules, and algorithms for changing specifications into designs. Ideally, these steps should be automatable.
- Design products: the results of specification and design should be clearly described, ideally delivering executable designs as code.

This schema was derived from previous studies[11] and shares many criteria with other evaluation frameworks[12]. Systems development methods may be classified into different groups that share some common approach or philosophical background[11]. Representative methods from different groups were selected for comparison against the following framework.

Conceptual modelling

- The data and processing control parts of a system are modelled in one unit rather than separately.
- The method produces a network system model of objects communicating by messages.
- The method explicitly models object types and instances.
- Classification of objects is supported with property inheritance.

Procedure and guidance

- The method should guide the analyst towards identifying and describing objects.
- Guidance should be available for analysis, specification, and design phases.

Transformations and products

- Design transformations should support change of OO specifications into designs implementable in OOP languages.

Table 1. Feature analysis of object-oriented methods

Method	Abstraction	Classification	Inheritance	Encapsulation	Coverage (R-A-S-D-I)
HOOD	Y	Y	Partial	Y	------
OOSD	Y	Y	Y	Y	------
OOSA	Y	Partial	-	-	-----
OOA	Y	Y	Y	-	------
ObjectOry	Y	Y	Y	Partial	-------

Key: Y = Yes.

R-A-S-D-I in coverage refers to Requirements Analysis, Analysis, Specification, Design, and Implementation. The measure of coverage is judged from the methods procedures and notations.

In the following sections, a selection of system development methods, chosen to cover diverse backgrounds from real-time to information-processing applications, is analysed to review how well they accord with OO concepts.

First, OO methods are reviewed for their support of OO principles, then traditional structured methods are surveyed in terms of their modelling perspective (data, process, or event)[12] and their potential fit to the OO approach. Selected methods are illustrated with specifications using the case study described in the Appendix. Space precludes illustration of all of the methods. Comparison of methods' specification is not the intention of this paper; instead, selected specifications are given to illuminate the differences between OO and non-OO methods.

OBJECT-ORIENTED METHODS

The claims of OO methods can now be evaluated using the OO meta-model. Each method is evaluated in terms of its fit with OO method criteria and its coverage in terms of analysis and design.

Hierarchical Object-Oriented Design (HOOD)[8]

As may be expected, this method scores well on OO properties (see Table 1). HOOD encourages modelling of objects explicitly, although there is little guidance for early analysis stages and structured analysis and design techniques are even recommended for the purpose. Objects are modelled in a hierarchical manner, with inheritance of properties between parent and child objects. There is strong emphasis on the object interface specification and encapsulation. A system network of objects communicating by messages is created with control by event messages. HOOD uses Booch's conception of actor and server objects.

HOOD supports object classes, but inheritance specification is not detailed and reuse support is not explicit. The method is better developed in the design phase and gives explicit transformations into Ada. Overall, HOOD incorporates many OO properties, but it is a real-time design method, consequently data specification and associated inheritance mechanisms receive less attention.

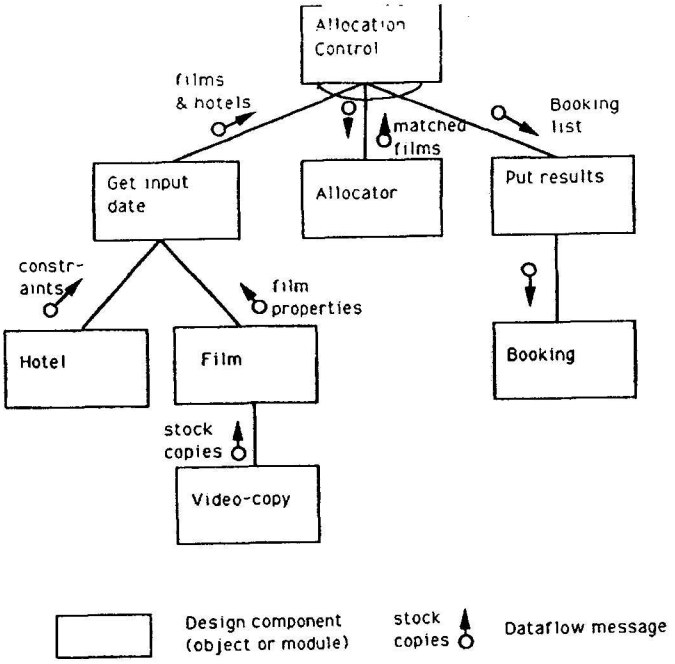

Figure 2. Object model of VI application produced by OOSD method

OOSD design showing structure chart notation. Some design components are shared with other methods, e.g., objects Film, Hotel, Video-copy, and Booking. Other components have been added by OOSD method, e.g., Allocation control, Put results

Object-Oriented System Design (OOSD)[13]

This method assumes that an analysis phase has identified and partially specified objects. OOSD provides a detailed notation for object classes and management of inheritance. Inter-object communication is also specified in terms of event/message types. The method supplies detailed notation for interface description and encapsulation, with local data and services. Part of an OOSD specification of the case study application is given in Figure 2. The system is modelled either as a sequentially executed hierarchy using the Yourdon structure chart notation or as an asynchronous network of processes with monitors.

No analysis advice is given, so coverage of OOSD is necessarily restricted to the design phase. The notation can become overcrowded and difficult to read.

Object-Oriented Systems Analysis (OOSA)[6]

Shaler and Mellor's method is described with a case study prototyping approach. It gives many heuristics for object identification and analysis, which help initial abstraction and object modelling. OOSA owes its ancestry to the data-modelling approach and many of its recommendations are indistinguishable from entity-relationship modelling.

The method models an object relationship network with subclasses. State-transition specifications are constructed for each object and functions are modelled with dataflow diagrams. The object relationship model is illustrated in Figure 3. The method does produce a composite activity-data model, but this is achieved by attachment of activity to the data model, essentially merging dataflow diagrams and state-transition models with entities. The procedure for achieving this synthesis is not explicit. The main criticism of OOSA is its lack of support for inheritance. Classes are supported, but only inheritance of object properties is modelled. Inheritance of services is not considered and reuse is not explicitly supported. In addition, the method is underspecified in the design phase.

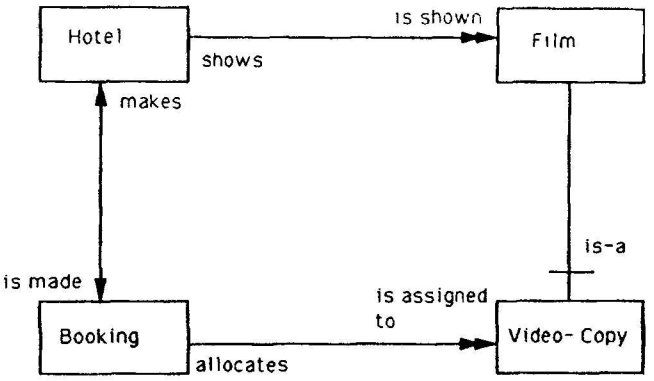

Figure 3. Object model of VI application produced by OOSA

Because OOSA takes data-modelling approach, more active objects, e.g., Clerk, Allocator, are not specified in object network. This functionality would be described in dataflow diagrams

Object-Oriented Analysis (OOA)[7]

OOA covers all OO concepts, although it is an analysis method, hence coverage of design issues is weak (see Table 1). Classification and inheritance are modelled and abstraction is helped by the structure layer, which gives an overview of object groupings for large systems. Objects are a composite data activity specification. Three links between objects are supported: relationship connections, which are modelled in the familiar data model crow's feet notation, classification hierarchies, and message passing. The resulting specification can appear overcrowded, although Coad and Yourdon separate the complexity into different layers (Subject, Structure, Attribute, Service) and build the specification incrementally. An OOA specification showing the object model in the service layer is depicted in Figure 4.

The method uses hierarchical inheritance and masking rather than multiple inheritance, and specification of encapsulation and object interfaces is not as detailed as in OOSD or HOOD. Overall, however, it does meet many OO criteria.

ObjectOry[14]

This method supports OO concepts of classification, encapsulation, and inheritance. Abstraction is promoted by levels in design from higher-level system views to lower block and component levels. ObjectOry adds concepts of user-centred design 'uses cases' to the OO approach for specification of the user interfaces and tasks provided by object services. Use cases are specified

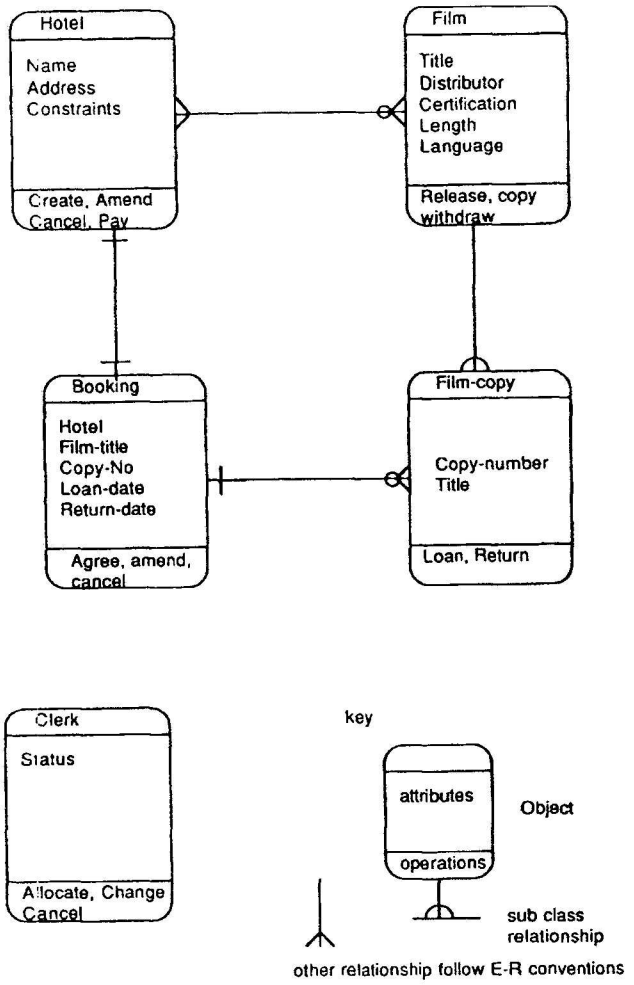

Figure 4. Object model for VI system produced by OOA method

with dataflow diagrams, and this functional specification is then mapped on to object services.

The composite data and activity definition of objects is not strongly enforced and services (described as processes) are also regarded as objects. Reuse is supported by component libraries, and design transformations to real-time languages are given (CHILL and Ada). Guidance for analysis is less comprehensive and the target applications of ObjectOry, like HOOD, appear to be real-time and engineering systems.

Summary of OO methods

The coverage of OO methods is variable and not all methods meet the necessary range of criteria. HOOD and OOSD give comprehensive design notations, but are weak on prescriptive guidance. Indeed, guidance in the analysis phase is totally absent. HOOD does fulfil most OO criteria, but does not completely support property inheritance, probably because its real-time orientation does not necessitate specification of complex data structures within objects. OOSA produces an object model with fewer components as a consequence of its data-modelling heritage, whereas OOA is more likely to identify actor as well as server objects. OOA meets many OO criteria and gives procedural advice, although its coverage of the design phase is not extensive. Consequently, no complete OO method exists, although all the issues are addressed separately in different methods.

Table 2. Summary of method specification models and approaches

Method	Functional process	Data relationship	Event sequence	Coverage (R-A-S-D-I)	Application
IE	Y	Y	Y	----------	IS
ISAC	Y	Y	N	------	IS
SASD	Y	N	Y	---------	IS
SSADM	Y	Y	Y	----------	IS
SADT	Y	Y	N	------	IS, RT
JSD	N	Y	Y	----------	IS, RT
NIAM	Y	Y	N	---------	IS (data intensive)
Mascot	Y	N	N	---------	RT

Key: Y = Yes, N = No.
Coverage of the life-cycle: Requirements (R), Analysis (A) Specification (S), Design (D), Implementation (I).
Application: IS = information systems, RT = real-time.

Table 3. Summary of structured methods' object-oriented features

	Object model	Data + activity	Encapsulation	Types + instances	Classification
IE	Poss	N	N	Y	N
ISAC	Y	N	N	N	N
SASD	Y	N	N	N	N
SSADM	Y	N	N	Y	N
SADT	Y	N	N	N	N
JSD	Y	Y	Y	Y	N
NIAM	Poss	Poss	N	Y	Y
Mascot	Y	Y	Y	Y	N

Notes:
(1) For the object model, Poss means an object model could possibly be constructed from the data model in these methods.
(2) To score Y for the object model, methods have to specify a concurrent network of message-passing processes, however these processes may be functional or data-oriented. This can be cross-checked on column two, which records whether data and processing are modelled together in an object.

REVIEW OF OBJECT ORIENTEDNESS OF SYSTEMS DEVELOPMENT METHODS

A summary feature analysis of the methods investigated is given in Table 2. The types of model employed by methods are categorized as functional/process (typically represented by dataflow diagrams), data relationship (entity-relationship diagrams), or event (entity life histories). The feature analysis also includes the approximate life-cycle coverage of each method. For further details of method comparisons, see Loucopoulos *et al.*[11] and Olle *et al.*[12]. A summary of the OO features is illustrated in Table 3 and described in more detail in the following sections.

Information Engineering (IE)[15]

Data modelling is an important component of IE, which encourages object modelling of the data components of a system. Functional specification uses process dependency and action diagrams, separated from data modelling, thereby discouraging common data and control specification. Cross-referencing of functions to entities is provided for and state-transition diagrams explicitly associate event-creating operations with entities, giving a partial OO specification.

Concepts of type-instance are supported; also IE encourages conceptual modelling of business processes leading towards object orientation. A data model composed of entities and relationships gives a network specification for the static part of systems, but separation during analysis of processing from data and the emphasis on functional decomposition means that IE cannot be regarded as truly object-oriented.

Information systems activity and change analysis (ISAC)[16]

This method advocates top-down functional decomposition of processing and data in separate specifications as activity and data diagrams. Emphasis is placed on analysis of change, and processes are viewed as transforming data, which encourages a partial OO approach. Type-instance and classification concepts are not supported. Even though a network model of processes and data structures is produced, the separation of data from system control makes ISAC more functionally oriented than object-oriented.

Structured Analysis/Structured Design (SASD)[17–19]

SASD uses top-down functional decomposition to analyse systems in terms of a network of processes connected by dataflow messages (see Figure 5). The method is based on principles of functional cohesion, which groups actions pertaining to a single goal in processing units, and coupling, which aims for low interdependence between system components. Dataflow diagrams specify the system as a network of communicating functions, which is transformed into a hierarchical design. The method does not support any OO concepts, separates data and process specification, and encourages specification of functionally based system components. More recent versions have added state-transition diagrams and bottom-up analysis driven by event identification[19]. This creates more potential for expressing OO specifications.

Structured Systems Analysis and Design Method (SSADM)[20]

SSADM is a composite method derived from structured analysis, structured design and data analysis. Process analysis is by dataflow diagramming and separated from data analysis, which employs an entity-relationship

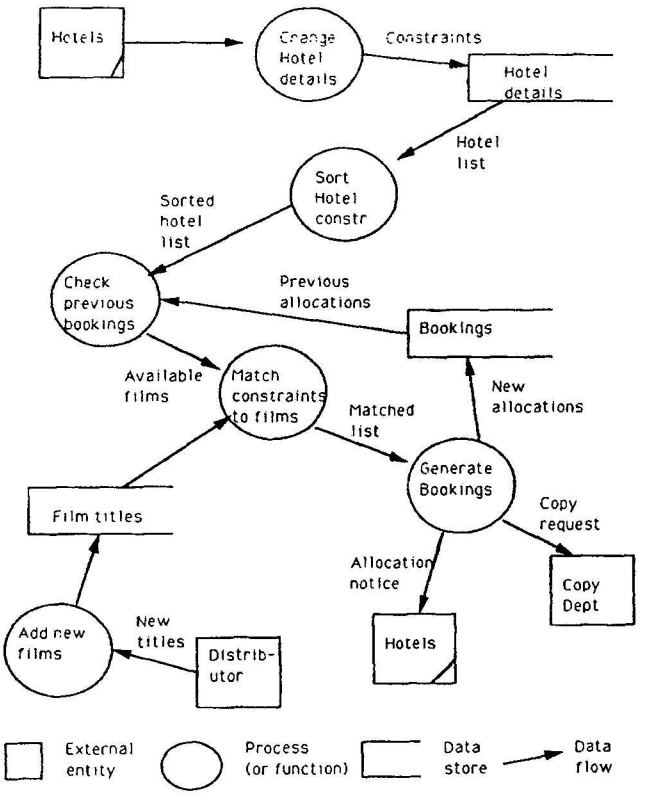

Figure 5. Dataflow diagram specification of VI application using SSA

approach. As with IE, data analysis encourages object orientation, but the separation of processing from data specification and use of top-down functional decomposition results in specification of functionally related processing structures. As a result most of the views expressed about IE also apply to SSADM. Entity life histories do associate processing events with data objects, but this is just one modelling view within the method. In version 4 of SSADM it forms a major theme within the overall specification and hence encourages OO specifications. Although SSADM does encourage data abstraction by conceptual modelling, functional modelling is also supported and hence it cannot be said to be truly object-oriented.

Structured Analysis and Design Technique (SADT)[21]

SADT uses top-down decomposition to analyse systems in successively increasing levels of detail. Specification uses network diagrams of processes connected by data flows, control messages, and mechanisms. The method does encourage modelling of real-world problems, but constructs separate activity and data models using the same box and arrow notation. More emphasis is placed on activity modelling. SADT does not support type-instance concepts, although some classification is possible in the hierarchical decomposition of data. The separation of process specification from data makes this method unsuitable for an OO approach.

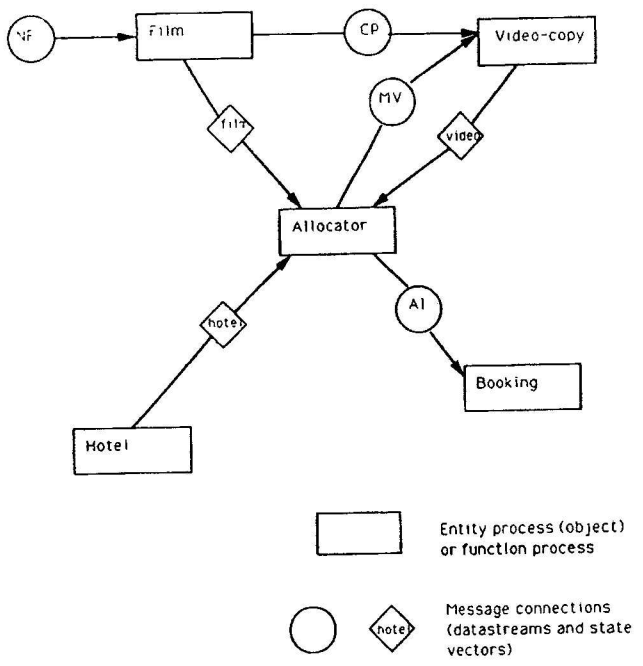

Figure 6. System network diagram of VI system produced by JSD

Jackson System Development (JSD)[22,23]

JSD produces system models based on networks of concurrent communicating processes, with a type-instance concept, although classification and property inheritance is not supported. System control is modelled in terms of time-ordering of actions associated with entities, and more recent versions have placed more emphasis on data analysis, resulting in an object model that combines data and operations. A JSD system specification diagram (see Figure 6) shows a network of communicating processes similar to an object model. Because of its emphasis on an entity-life-history approach, JSD has much in common with OO methods, although it does not explicitly support all OO concepts. Even though object classification is not supported, JSD does advocate alternative views on an object, called entity roles.

Nijssen's Information Analysis Method (NIAM)[24]

NIAM is a conceptual-modelling method that concentrates on data specification during the early parts of the analysis life-cycle. Based on the ANSI/SPARC schema, it supports data abstraction with conceptual modelling, thereby encouraging object orientation. Process analysis is by addition of semantic constraints to the data model and by specification of transactions for data input and output using a rule-based approach. Type-instance concepts are supported, as is classification by entity subtypes, so NIAM can be said to possess some OO properties, although it does not support inheritance. However, emphasis on constraint-based processing tightly coupled to relationship roles in the data model does detract from the OO approach.

Mascot-3[25]

Mascot advocates functional decomposition of systems, however, recent versions have introduced modular concepts of encapsulation and clearly defined interfaces for system components. Mascot system specifications consist of a network of communicating processes, and hierarchical abstraction is supported. Mascot has a type-instance concept for implementing many instances of software modules from one template 'type'. However, it does not explicitly support classification of objects, although some inheritance of communication procedures between modules is provided for by the access interface. Encapsulation is encouraged by the strongly typed interface specification of modules.

Mascot gives little guidance during early analysis, and other functional methods such as structured analysis and CORE[26] are recommended. Overall, Mascot encourages the analyst to produce a functionally oriented specification because of its imprecise early stages and emphasis on functional decomposition, although its implementation does incorporate OO features.

Summary of method evaluation

Methods using functional decomposition (e.g., SASD) encourage identification of goal-related components in systems (see Figure 5), in contrast to the OO approach (see Figures 3 and 4), which promotes system components more compatible with data models. SASD encourages specification of a hierarchy of task/procedural units that are unrelated to the objects on which the tasks act.

Although it may be argued that functions are essentially objects containing only activity, a method's viewpoint will influence modelling. An analyst trained in the functional approach will naturally identify goal-related modules using the principles of cohesion and functional decomposition[17,18]. In contrast an analyst using an OO viewpoint will identify modules that relate to a model of the real world without prejudice to processing goals. However, OO methods such as OOSD and HOOD do not encourage a specific view on object identity, so it is possible to argue that structured analysis and design modules are equivalent to actor objects in Booch's sense. Resolution of this dichotomy may depend on the fit of method and application, with real-time methods (e.g., HOOD, ObjectOry) tending towards functional, actor-type objects. For information systems, data-oriented objects may be more suitable.

Consequently for information systems, structured methods with a data-modelling heritage (e.g., IE, SSADM) are closer to the OO approach. Data modelling encourages specification of the static aspects of object structures. Unfortunately, data modelling ignores dynamic system components, and as a result these methods generally borrow functional specification for the dynamic parts of the system from methods such as structured analysis. Process specification that relies on functional decomposition will bias implementation towards functionally based structures. Another method in this group

is NIAM, which emphasizes semantic data modelling, combining entities and rules in one model. In spite of this, NIAM does not explicitly attach all the system activity specified as rules to objects.

JSD views entities as being active and creates a system model explicitly based on real-world objects, combining data and control within one structure. JSD, however, does not support object classification. Instead, it advocates multiple views of an object in terms of roles that could be used to specify property sharing. Mascot cannot be regarded as truly object-oriented because it uses functional decomposition to identify modules. However, Mascot, in common with other real-time methods, does include OO concepts such as encapsulation.

In summary, current structured methods using an entity-modelling and/or entity-life-history approach have potential to evolve towards object orientation. Classification and encapsulation are supported, but separately in different methods. Inheritance is not supported, although data-oriented methods could incorporate these features, as illustrated by the evolution of OOSA and OOA.

DISCUSSION

The first part of the discussion reviews OO concepts proposed by previous studies, followed by discussion of the object orientedness of system development methods.

Object-oriented concepts

Objects are close relatives of abstract data types[27], which first brought specification of data structures and operations together. Objects, however, go beyond abstract data types, which emphasize control from a viewpoint of constraints on data structures, to encompass a wide range of system components. Booch[10] defines objects as entities characterized by their actions, essentially composite specifications of the active, processing and the static, data-related components of systems. Reviewers of OOP have also defined objects as being composite specifications of data and control/actions[1,3], combined with properties to enhance program maintenance and reusability of modules.

The importance of modelling systems that can respond to change is discussed by Maclennan[27], who points out that there is a dichotomy between valued-oriented and OO programming. The former being based on mathematics is concerned with unchanging definitions and alias values; object orientation, however, is about change and the tasks of recording and responding to it. Maclennan[27] develops this point to demonstrate that many current programming languages are value- rather than object-oriented.

While current programming languages rarely support OO principles, a new generation of languages has been developed to support object programming, some of which (e.g., C++, Smalltalk) have gained widespread acceptance. To reap the rewards of improved maintainability and reusability which these languages offer, system development methods need an OO approach, otherwise procedural specifications will continue to be implemented, failing to reap the advantages of OO design.

General conclusions

Principles of OO development have been devised to tackle problems of poor specification, the lack of maintainability, and the need for software reusability. It may be argued that use of a particular system development method will not bias implementation of OO systems and that OO designs may be derived from any specification. This view is unrealistic, as demonstrated in this study by the different specifications produced by application of OO and non-OO methods. However, data model and OO specifications show considerable convergence, suggesting a feasible migration path from structured methods such as JSD, IE, and SSADM towards further object orientation.

Functionally based development methods (e.g., Structured Analysis) are less well suited to development of OO systems. If functionally based methods are used, the designer would have to map functional components on to objects, a difficult task that may require re-specification of large parts of the system. Some attempts have tried to graft functionality on to objects in an *ad hoc* manner[28], resulting in muddled specification of objects without a clear modelling basis. More recent developments have taken the entity model as the starting point for object definitions and then used dataflow design to model services, alias functionality[6,7].

The functional bias problem arises with OO real-time methods (e.g., HOOD), which either leave the analytic phase underspecified or recommend use of methods based on functional decomposition and procedural dependency (e.g., SADT[21] and CORE[26]) as front-ends for requirements analysis and early specification stages. The OOSD method[14] builds on structured design concepts and develops a notation and design procedure for object-like modules. The method, however, does not cover requirements analysis and specification. OO analysis methods offer coverage of the early life-cycle phases[6,7], by integrating object specification with dataflow diagram specification and entity-relationship analysis, although only the Coad and Yourdon method meets all the OO modelling requirements. OO analysis does not offer good coverage in early life-cycle phases, but no design transformations are included. All of these methods have yet to be proven in practice and have little computer-aided software engineering (CASE) tool support, but they do lend support to the importance of the data model in OO concepts.

Within the current generation of structured system development methods only JSD has a truly OO approach to modelling, even though it does not support classification. However, data-modelling approaches using rules applied to data structures, as found in NIAM's semantic data model, may also provide a promising way forward. The derivation of OO specification as created by the Coad and Yourdon method demonstrates that method

evolution is possible and practical. Further evidence of evolution moves may be the importance attached to entity life histories, essentially Jackson techniques, in version 4 of SSADM.

Migration to object orientation, however, will largely depend on system developers being convinced of the benefits of the approach. Thorough evaluations of OO claims for improved maintainability and reuse have not been published, if they exist at all. Object models alone are unlikely to be sufficient to promote extensive reuse as none of the OO methods contains procedures or explicit modelling techniques for reusable system development. Initial studies of this problem suggest considerable problems exist in specifying generic objects[29]. Furthermore, because much information about domains is contained in the relationships between objects and in propositional statements object models alone may be insufficient for specification of applications. OO methods may need to move in the direction of semantic data modelling e.g., TAXIS[30] and CML[31], to augment the data/activity specification of objects with richer semantics. The inter-relationship between objects and system control could also present problems for OO methods, as recognised by Nierstrasz[4]. Modelling techniques to specify inter-object communication and message-passing control will have to progress beyond concepts of client-server objects as found in HOOD.

If, to paraphrase Rentsch's[1] prediction, 'object oriented systems development will be in the 1990's what structured design was in the 1970's', system development methods will have to pay more attention to OO concepts and approaches. On the other hand, proponents of the OO approach will have to demonstrate the validity of their claims by evaluation in industrial-scale applications.

ACKNOWLEDGEMENTS

The author is grateful to colleagues at City University, Alwyn Jones and John Crinnon, for their comments and suggestions.

This work was based on research within the AMADEUS project 1229(1252), partially funded by the Esprit programme of the Commission of the European Communities.

REFERENCES

1 **Rentsch, T** 'Object oriented programming' *SIGPLAN Notices* Vol 17 No 9 (1982) pp 51–61

2 **Cohen, A T** 'Data abstraction, data encapsulation and object oriented programming' *SIGPLAN Notices* Vol 17 No 1 (1984) pp 31–35

3 **Cook, S** 'Languages and object oriented programming' *Soft. Eng. J.* Vol 1 No 2 (1986) pp 73–80

4 **Nierstrasz, O M** 'An object-oriented system' in **Tsichritzis, D (ed)** *Office automation* Springer-Verlag (1985)

5 **Tsichritzis, D** 'Objectworld' in **Tsichritzis, D (ed)** *Office automation* Springer-Verlag (1985)

6 **Shaler, S and Mellor, S J** *Object oriented systems analysis* Yourdon Press (1988)

7 **Coad, P and Yourdon, E** *Object oriented analysis* Yourdon Press (1990)

8 **Robinson, P J (ed)** *The HOOD manual, issue 2.1* European Space Agency, Noordwijk, The Netherlands (1987)

9 **van Griethyuysen (ed)** 'Concepts and terminology for the conceptual schema and the information base, computers and information processing' *ISO/TC97/SC5/WG3* International Organization for Standardization, Geneva, Switzerland (1982)

10 **Booch, G** 'Object oriented development' *IEEE Trans. Soft. Eng.* Vol 12 No 2 (1986) pp 211–221

11 **Loucopoulos, P, Black, W J, Sutcliffe, A G and Layzell, P J** 'Towards a unified view of system development methods' *Int. J. Inf. Manage.* Vol 7 No 4 (1987) pp 205–218

12 **Olle, T W *et al.*** *A framework for the comparative evaluation of information systems methodologies* Addison-Wesley (1989)

13 **Wasserman, A, Pircher, P A and Muller, R J** 'Concepts of object oriented design' *Technical report* Interactive Development Environments, San Francisco, CA, USA (1989)

14 **Jacobsen, I** 'Object oriented development in an industrial environment' in *Proc. OOPSLA-87* ACM Press (1987) pp 183–191

15 **Macdonald, I G** 'Information engineering — an improved, automatable methodology for the design of data sharing systems' in **Olle, T W, Sol, H G and Verrijn-Stuart, A A (eds)** *Information systems design methodologies: improving the practice* North-Holland (1986)

16 **Lundeberg, M, Goldkuhl, G and Nilsson, A** *Information systems development: a systematic approach* Prentice Hall (1981)

17 **DeMarco, T** *Structured analysis and system specification* Yourdon Press (1978)

18 **Yourdon, E and Constantine, L** *Structured design* Yourdon Press (1977)

19 **Yourdon, E** *Modern systems analysis* Prentice Hall (1990)

20 **Longworth, P G and Nicholls, D** *SSADM—Structured Systems Analysis and Design Method* NCC Publications (1986)

21 **Ross, D T and Schoman, K G** 'Structured analysis for requirements definition' *IEEE Trans. Soft. Eng.* Vol 3 No 1 (1977) pp 1–65

22 **Jackson, M A** *System development* Prentice Hall (1983)

23 **Sutcliffe, A G** *Jackson System Development* Prentice Hall (1988)

24 **Nijssen, G M** *A conceptual framework for organisational aspects of future data bases* Control Data Corporation, Brussels, Belgium (1978)

25 **Simpson, H** 'The Mascot method' *Soft. Eng. J.* Vol 1 No 3 (1986) pp 103–120

26 **Mullery, G** 'CORE — a method for controlled requirements specification' in *Proc. 4th Int. Conf. Software Engineering* IEEE (1979)

27 **Maclennan, B** 'Values and objects in programming languages' *SIGPLAN Notices* Vol 17 No 12 (1982) pp 75–81

28 **Balin, S C** 'An object oriented requirements specification method' *Commun. ACM* Vol 32 No 5 (1989) pp 608–620

29 **Sutcliffe, A G** 'Towards a theory of abstraction: some investigations into the object oriented paradigm' *Technical report* City University, London, UK (1991)

30 **Greenspan, S J and Mylopoulos, J** 'A knowledge representation approach to software engineering: the TAXIS project' in *Proc. Canadian Information Processing Society* Ontario, Canada (1983) pp 163–174

31 **Jarke, M** 'DAIDA: conceptual modelling and knowledge based support for information systems development process' in *Software engineering in Esprit (Techniques et Science Informatiques)* Vol 9 No 2 Dunod-AFCET (1990)

APPENDIX: CASE STUDY

A complete description of this case study can be obtained from the author. A summary is presented here.

Video International hires video tapes of films to hotels, who

then transmit videos to guests via internal cable TV networks. Films are hired from distributors, who charge a rental fee based on the popularity of the film and the duration hired. Video International has contracts with hotels to supply a set number of films as specified by the hotel. Films are hired in blocks of one or more weeks and it is usual for hotels to offer guests a choice of four to five films. Hotels impose constraints on the type of film they wish to accept. Some hotels have a policy on non-violent films, some films may offend religious values, while other hotels accept films with specific running lengths. In addition, all hotels do not wish to be allocated the same film twice. Hotels may also change their film preferences from time to time.

The problem is to satisfy the demand for films from the available titles within constraints imposed by individual hotels. The hiring history of each hotel has to be examined to determine which films they have not received. Films are allocated to hotels and the appropriate number of copies are made for the demand. Video copies are delivered to hotels. Sometimes video tapes break and the copy has to be replaced. Records of the hotel video booking log have to be updated, showing which film copies have been allocated to each hotel for each week. Revenue is calculated from these logs, however, billing is not within the remit of the investigation.