Domain Models for User Interface Design

David Benyon

Computing Department, Open University,

Milton Keynes, MK7 6AA, UK.

+44 (0) 908 652679

D.R.Benyon open.ac.uk

ABSTRACT

Human-Computer Interaction (HCI) is concerned with developing computer-based systems which help people undertake their work or other activities and which demonstrate a high degree of usability. Both the process of development and the various products which are produced during development are critical to the success of the final human-computer system. Within HCI there is an increasing awareness of the importance of developing abstract representations - or models - of the area of activity which is to be the subject of the system. Such a representation is known as a domain model. However, there are a number of competing candidates for domain models. What are the strengths and weaknesses of these? In this paper, the philosophical and practical aspects of developing domain models of HCI are discussed.

1. INTRODUCTION

Human-Computer Interaction (HCI) is concerned with developing computer-based systems which help people undertake their work or other activities and which demonstrate a high degree of usability. Both the process of development and the various products which are produced during development are critical to the success of the final human-computer system (Hix and Hartson, 1993; Preece, et al., 1994). Within HCI there is an increasing awareness of the importance of developing abstract representations - or models - of the area of activity which is to be the subject of the system. Such a representation is known as a domain model, or an application model.

Any domain model must be built upon some conceptualisation of the domain which it is representing. Indeed the primitive constructs, or concepts, from which a model is built will reflect that conceptualisation. In HCI, however, there is no universally agreed conceptualisation on which we can base a model; there is no agreed theory of HCI. For example, there is the question of whether we interact *with* the computer or we interact *through* the computer; a debate which has been well-rehearsed in the pages of *Interacting with Computers* (Bench-Capon and McEnery, 1989a; Barlow, Rada and Diaper, 1989; Bench-Capon and McEnery, 1989b; Keeler and Denning, 1991). In (Bench-Capon and McEnery 1989b) the authors point out that both perspectives are important, but that the interacting through view should not be overlooked.

Another aspect of this debate concerns the importance of the concept of a 'task' in HCI. This concept has dominated the ontology of HCI in recent times as the wealth of literature on task analysis techniques will demonstrate (e.g. Diaper, 1989; Wilson, et al. 1988). Tasks are central to Carroll's (1990) conceptualisation of the task-artefact cycle. Long's (Long, 1989) conceptualisation of HCI includes the concept along with humans, computers and effectiveness. More recently however, the concept of 'task' has come in for criticism (Suchman, 1987; Benyon, 1992a; Benyon 1992b; Draper 1993).

Others (e.g. Fischer, 1989; Storrs 1989) challenge the dominance of tasks, arguing for a communication model of HCI. This model recognises the existence of shared knowledge in a particular problem domain. Humans and computers possess different but complementary knowledge and skills and should cooperate in the problem-solving activity. HCI should be viewed as the result of 'an interplay between [human] mental processes and external computational and memory aids'. (Fischer, 1989, p. 45).

It is likely that different conceptualisations of HCI have different strengths in different circumstances. In this paper, the philosophical and practical aspects of developing domain models of HCI are discussed. In section 2 a very general conceptualisation of HCI is provided. The following three sections examine three fundamental and contrasting modelling paradigms;

task-based approaches, object-oriented approaches and data-centred approaches. The discussion section 7 provides a comparison of these.

2. HCI AS A SYSTEM

When we sit in front of a VDU it is natural and easy to focus attention on the human-computer interaction and the tasks which we undertake. For example, to download a file from a remote computer I might have the tasks; log on to remote computer, enter a password, locate the directory and type 'get <filename>'. Yet when we take a picture with a modern, automatic camera or we navigate the globe using the internet it is equally easy to overlook the role of information technology (and increasingly intelligent information technology) in these activities. Using the world-wide web to get a file I click on an icon. A software system locates the remote computer, another logs on, another locates the directory and yet another sends me the file.

Software systems are now embedded in a multitude of devices. Consequently we need to consider HCI not so much as a human using a computer, but more as a network of interacting systems; some interacting with each other and some interacting with the end-users. Ultimately users are interacting with other people through the computer - with the person who put the file in the directory, or wrote the software - but there are so many software systems operating at different levels of the interaction that such a description of the interaction on its own is not very enlightening.

Instead of teasing HCI apart into humans, tasks, interfaces, computers and so on, it may be more fruitful to begin with a more basic concept; a system. A system is a more or less complex object which is recognised, from a particular perspective, to have a relatively stable, coherent structure (Checkland, 1981). For example, I might wish to consider a human interacting with a computer as a system. Another person might wish to consider the computer as a system in its own right. Another person might wish to consider the users of networked computers distributed across the globe as system.

The concept of a system is a useful place to start because it can be applied at many different levels of abstraction and from many perspectives. However, it is not so general a concept that it has no foundation. Checkland's (1981) theory of systems is perhaps the clearest exposition. He stresses the need to declare explicitly as part of the system definition, the perspective (or *weltanschauung*) from which the phenomenon is being considered as a system and the aspects of the system which are considered to be stable and coherent. Systems theory also recognises that all systems are composed of other systems and exist within wider systems.

Another important aspect of systems theory is the recognition that systems interact with other systems. Systems interact with their sub-systems, with their super-systems and with systems at the same level of abstraction. The interaction of a systems' component subsystems results in the system having properties which *emerge* from the relationships existing between its sub-systems. In other words, systems possess properties which are not possessed by any of its subsystems. For example, if I consider water to be a system composed of the subsystems hydrogen and oxygen, then water clearly possesses properties not possessed by either hydrogen or oxygen. The properties of water emerge from the relationship between its subsystems. If I consider the Internet to be a system (defined as all the people, software, computers and communication mechanisms who are able to access the 'net), then its properties (e.g. being able to read today's *Daily Telegraph* from South America or participating in the virtual communities of computer conferences) are not possessed by any of the component systems. The properties emerge from the relationships between those component systems.

In order to illustrate the system concept, consider the systems illustrated in Figure 1. The environment under consideration in this figure is the banking system. The banks themselves exist within this environment (and indeed in other environments which are not under consideration). There will be other institutions which are part of the banking system which are not banks. My various accounts exist within the banks and stray outside (e.g. into Building Societies.) Automatic Teller Machines (ATMs) exist within the banks. Some are related to my accounts but others are not. The interfaces to ATMs exist within the ATM system. In Figure 1 we see a variety of systems, defined at a variety of levels of abstraction and from a variety of perspectives. The systems engage in a variety of interactions.

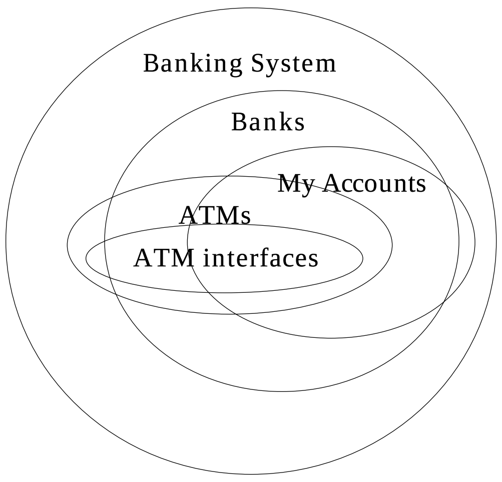


Figure 1 Some interacting systems in a Banking environment

Viewing HCI as a system focuses attention on the existence of multiple levels of networks of interacting systems. Observers, designers or participants in such a network can choose to identify and describe systems and their interactions at a level of abstraction appropriate for their purpose at hand. This purpose and the level of description needs to be openly declared as it is part of the modelling process.

Interacting Systems

In HCI we are primarily concerned with interaction, but how do systems interact? Theorists from many disciplines such as communication theory (e.g. Cherry, 1966), semiotics (e.g. Eco, 1976), systems theory (e.g. Laszlo, 1969), linguistics (Lyons, 1977) and information theory (Shannon and Weaver, 1949) recognise that systems interact through the exchange of signals which travel through a communication channel. Signals are the elementary particle of interaction (Benyon, 1993). Signals are binary in that they either exist or do not exist (Eco, 1976). However, they rarely travel alone and we refer to a structured collection of signals as a message. Like systems, however, signals and messages need to declared and viewed at a given level of abstraction. From my perspective pressing a button labelled 'withdraw cash' is the way I send signals to the ATM. From the perspective of the computer, the signals might be described as a series of bits.

In this conceptualisation of HCI as a network of interacting systems, it is useful to recognise two main classes of participants; agents and devices. Agents are intentional, autonomous systems in that they have beliefs and desires and can formulate their own goals (Laurel, 1990). People are agents and we are beginning to create artificial agents (e.g. Maes and Kozeriok, 1993; Maes, 1994). Devices are not autonomous. They may demonstrate sophisticated behaviour, but they do so without the sense of volition attached to agents.

Agents and devices are systems. They interact through the exchange of signals. Agents make use of devices in order to achieve goals within the actual and perceived constraints of an environment. The environment is itself a system which is interacting with other systems. Thus there are multiple levels of agent-device systems. In order to send and receive signals, both agents and devices require transmitter and receptor functions. These functions are themselves devices. Thus all interaction (at some level of abstraction) is mediated by devices.

Focusing on the interaction of systems allows us to distinguish two perspectives on those systems. One perspective attends to the syntax, semantics and pragmatics (Stamper, 1977; Eco, 1976) of the transmission and reception of signals. This is called different things by different authors such as the 'presentation' view, the 'view', the interface, the perceptual view and so on. Coutaz, *et al.*, (this volume) provide a comparison of several nomenclatures. Some authors (e.g. Goedicke and Sucrow, this volume) like to distinguish the internal presentation from the external presentation. The other perspective concerns the mechanisms by which systems structure and manipulate the signals. This is the 'abstract' view, also called the 'model', the abstraction view or the conceptual view.

There may be several presentation views on a system according to the other systems with which it interacts. For example, the ATM has one presentation view in order to interact with the human user of the ATM, another to interact with the bank card and another in order to communicate with the central computer. Similarly we may view the user of an ATM from the presentation view of pressing keys on the keypad, from inserting the card into the machine and so on. Clearly there are many other presentation views of humans; when they interact with other humans or with other devices.

This systems perspective of HCI shifts attention from the characteristics of humans and computers. Instead it focuses on interactions and the exchange of messages between the components of a human-computer system. Different views of those systems and their interaction will reveal different aspects of the interaction. This is illustrated schematically in Figure 2.

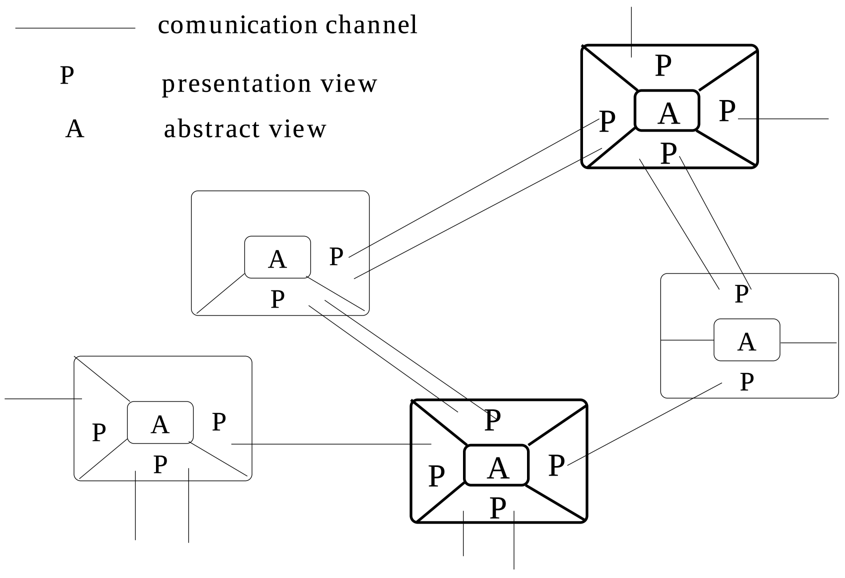


Figure 2 Systems have a variety of presentation views and a variety of interactions

3. MODELLING THE DOMAIN

The domain of HCI is a network of interacting systems some of which are agents and some of which are devices. If we wish to think about HCI, and to design human-computer systems, we need to conceptualise the domain using appropriate representations - or conceptual models. Conceptual models are devices for understanding, communicating, testing or predicting some aspects of some system. They are 'professional' languages which both constrain and focus a discourse by limiting the range of concepts which can be expressed in the language (Kangassollo, 1983, Lyttinen, 1983). Conceptual models provide a certain perspective of the domain by employing abstraction mechanisms which are reflected in the content and the structure of the concepts employed by the model.

A conceptual model will be more or less effective depending on the characteristics of the model. First we need to consider the purpose and use of the model. Second we need to consider whether the model possesses the necessary structure and processing capability to fulfil its purpose and third we need to consider whether the physical characteristics of the model - its notation and its usability. The analytic, explanatory and communicative power of a conceptual model arises from the structure, operations and constraints which that model is able to capture (Kangassalo, 1983).

For example, consider maps as conceptual models of a terrain. Maps are abstractions from the domain (the terrain) which emphasise some features whilst suppressing others. If one wants to find a path from one village to another then a relief map (one that shows only the height of the land) is unlikely to be of much help. It's purpose is not appropriate for the purpose at hand. If we have a map of the scale 1:50000 this may be more suitable for the purpose of finding a path. The map may have been designed to show fields, fences, rivers and paths and so contains the required functions (the ability to follow a path on the map) and the necessary structure (the concepts of path, etc.) to fulfil its purpose. However, the map may be so poorly designed at the physical level (e.g. it uses black lines to show boundaries and paths and rivers) that it is not usable for its purpose.

Our purpose is to develop human-computer systems. It follows that any model which we use must be orientated towards that purpose. Even with this declared purpose there are many levels at which we can view human-computer systems. At the organisation level we might wish to consider the design of working practices and the impact of new technologies on organisations. At the environmental level, we might wish to look at health and safety issues or office layout. At another environmental level we may wish to look at legal and ethical issues concerning human-computer systems. At a physical level we could examine the use of colour or font style on the usability of systems. At a more detailed physical level we may consider hardware and software issues concerned with rendering accurate representations of real world objects. At each of these levels there are a variety of devices and models which can help the designer.

The focus of attention here - and the models considered in the following sections - are at the logical, or conceptual level of the design of a suitable human-computer system which can deliver the required functionality and usability. We are concerned with an abstract view which can support the various presentations (or interactions) required by the component agents and devices in the system. We are concerned with who or what will perform which functions and provide which pieces of knowledge and at the structure of the human-computer dialogue.

The models which are considered in the next three sections have been chosen because they represent important paradigms within systems design and which are all candidates for the domain model which should underlie the development of human-computer systems. The three approaches differ in the type and level of abstraction which they use, the underlying concepts and notation which they employ and the constraints and operations which they support. Task-based approaches focus on what the user has to do. Object-oriented approaches exploit a representation based on the exchange of messages between objects. Data-centred approaches focus on the data which exists and which flows through the system. As is discussed in section 7, many methods in HCI utilise more than one of these generic modelling approaches.

One important aspect of modelling which is frequently misunderstood is that the concepts employed by a model constitute the material from which that model is constructed; they are not the object of the model. Thus an algebraic model of a circle, $x^2 + y^2 = z^2$, is not a model *of* algebra. It is a model *of* a circle *made from* algebra. This could be contrasted with a model *made from* programming constructs expressed in a language such as Logo (which may be represented as To Circle, Forward 1, Right 1, Circle). Thus task-based models are models of a human-computer system made from tasks, object-oriented models are models of a system made from objects and data-centred models are models of a system made of data.

4. TASK MODELS

Task models seek to represent the domain in terms of tasks. The emphasis of task-based approaches is primarily on human tasks and the need to understand what people (rather than computers) have to do. Task-models were developed because of the emphasis in HCI on people interacting with computers. The essence of the task-based paradigm is summed up by Carroll in his plea for an 'ontologically minimised' HCI (Carroll 1990). He argues that 'A task implicitly sets requirements for the development of artefacts, and the use of an artefact often redefines the task for which the artefact was originally developed' (p. 323). This task-artefact cycle would certainly appear to characterise one of the main problems which HCI faces. Carroll justifies the importance of task by saying 'conceiving of HCI activity in this way clearly implies that to design more useful artefacts we must better understand the tasks people undertake and better apply our understanding of tasks in the design process'. Although there is a sense in which this view appears self-contradictory (surely if the artefact is going to change the task, understanding the task during design is going to be of little help in the longer term), Carroll's view of the centrality of tasks is one that is pervasive in HCI circles.

Task-based approaches have as their basic concept the idea of a user 'task'. Although there are many variations in the definition of the term - and different authors define task in different ways - the concept of a task may be defined as 'a goal together with some procedure or ordered set of actions that will achieve that goal' (Amodeus, 1994). (But see Benyon, 1992a, Draper, 1993 or Diaper, 1989 for more discussion). Other basic concepts such as 'goal', (a state of the environment or agent which is desired by the agent) 'operation' 'simple task', 'unit task' or 'action' (a task which involves no problem solving or control component), 'plan' or 'procedure' (a sequence of tasks, sub-tasks and / or actions), 'job' and 'role' ('the collection of tasks that a person occupying that role performs'; Johnson, 1992 p. 162) are defined in terms of this basic concept. Task-based approaches date back to the earliest exposition of HCI theory (Card, Moran and Newell, 1980; Moran, 1981) and continue to be popular today (e.g. Browne, 1994). The wide variety of task-based models and the different uses to which they are put adds to the confusion.

The notation used in task-based methods is either based on a grammar type of representation or it is based on the structure chart notation. Structure charts represent a sequence of tasks, sub-tasks and actions as a hierarchy and include notational conventions to show whether an action can be repeated a number of times (iteration) and the execution of alternative actions (selection).

The most recent and complete description of a task-based approach is provided by Lim and Long (1994). Their method - MUSE - follows the structured approach advocated in information systems development circles and hence models the systems development process as well as the domain. The method is a well-researched and detailed scheme for integrating human factors into information systems development approaches based on a structure chart notation.

The process advocated by Lim and Long (1994) is to construct task models of existing or extant systems, develop a Generic Task Model and from this produce a Composite Task Model for the new system. This Composite Task Model describes both user and system activities and is used to derive a System and User Task Model which divides the tasks and actions required between the user and the computer. User interface specification may then be undertaken.

In MUSE, Lim and Long are careful to emphasise the role of abstraction and 'generification' in modelling. Thus the move from extant task models to Generic Task Model is vital. They recommend the following steps;

1. Take as input task information described in the statement of requirements. Temporal and conditional aspects of task execution should also be noted.

2. Summarise the task (and sub-tasks) in device-independent terms to reveal the logic underlying the design of the task.

3. Re-express the task description using structured diagrams to derive a target Generalised Task Model, and record any additional notes.

Lim and Long (1994) illustrate their approach using two main case studies; an ATM and a network security system. A portion of their ATM example is shown in Figures 4 and 5. Following an analysis of current (or extant) systems, a Generalised Task Model for an ATM has been produced.

Although Lim and Long (1994) do not provide their version of the Generalised Task Model for the ATM, we may assume that it is similar to Figure 3. This shows that the 'Simple ATM' consists of two sub-tasks which are completed in sequence; Present Personal ID and Select Service. Present Personal ID consists of two sub-tasks Enter Card and Enter PIN and in its turn Enter PIN consists of a number of iterations of the Press Digit action. Select Service consists of either Withdraw Cash or Check Balance. Each of these is further redescribed (indicated by the dotted line, but not shown in this example).

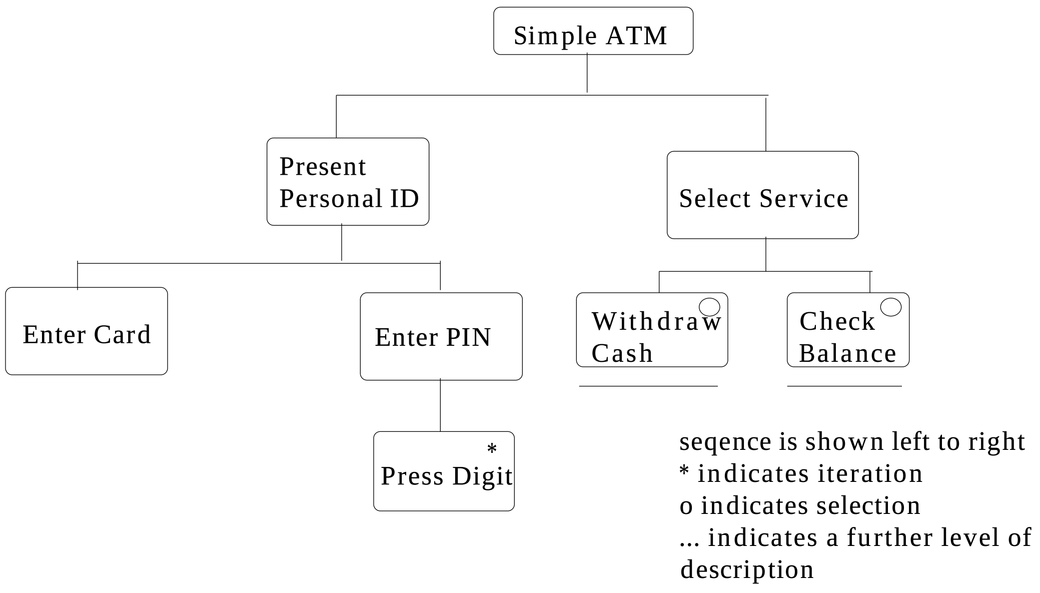


Figure 3 Generic Task Model for a portion of an ATM

Taking on board a statement of user needs, the Composite Task Model for the new system has been developed. A portion of this is shown in Fig. 4. This model must now be decomposed into the user and system tasks (Fig. 5).

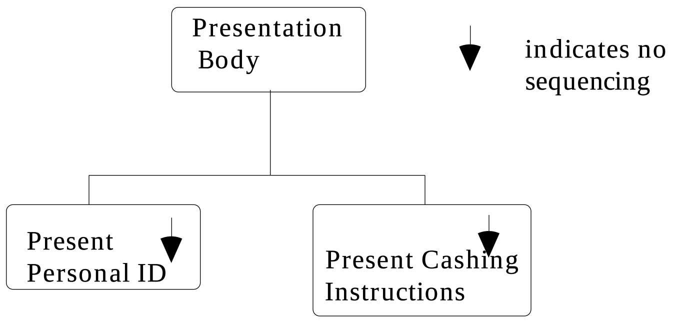


Fig. 4 Portion of a Composite Task Model for an ATM (from Lim and Long, 1994).

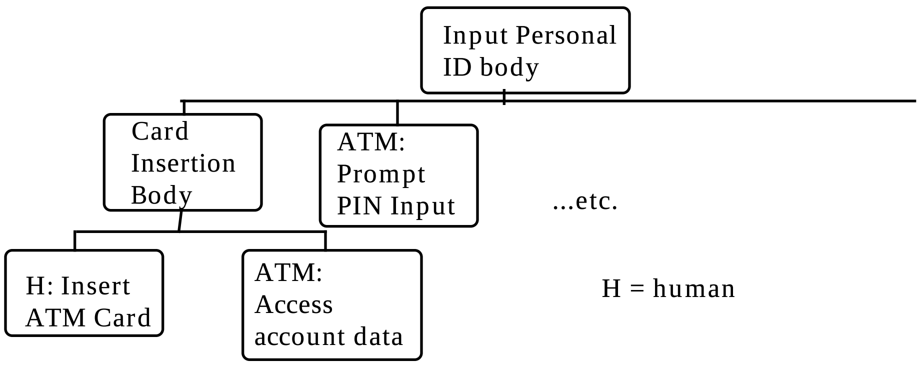


Fig. 5 Portion of a User and System Task Model (from Lim and Long, 1994)

This small snippet of the task-based approaches to developing conceptual models of the domain illustrates several important features of the approach. First, two important aspects of modelling are emphasised - abstracting from current systems and aiming for device independence (so, for example the composite task model in Fig. 4 allows for the processes Present Personal ID and Present Cashing Instructions to be undertaken in any sequence). Second the approach uses a notation which can be used to model the whole human-computer system and to subsequently identify human and computer activities (as illustrated by the human and ATM activities in Figure 4). This aspect has also been emphasised by (Sutcliffe and McDermott, 1991).

The purpose of the task-based approach is to facilitate the design of systems through analysing and abstracting current practice. The concepts employed by the model are tasks and sub-tasks. The physical notation and representation of the model is the structure chart notation. Task-based models are able to show the sequencing, iteration and selection of sub-tasks and actions and to show the allocation of tasks to human or to computer.

MUSE focuses primarily on the logic of the task. Sutcliffe (this volume) also focuses on the logic of the task, but couples a task-based approach with a data-centred approach in which he focuses on 'information categories' and Paterno' (this volume) couples a task-based approach with an object-oriented model of the 'interactors'. Other task-based approaches focus more on the cognition of the task and prefer a grammar notation to the structure chart (e.g. de Haan, this volume).

5. OBJECT MODELS

Another contender for a conceptual domain modelling formalism in HCI is the object-oriented paradigm. In this approach, the domain is represented in terms of the objects which exist, the relationships between objects and the messages which are passed between objects. The proliferation in object modelling methods has resulted in a large amount of confusion over exactly what is being represented, how it should be represented and how to approach application modelling using the OO approach. McBride (1994) suggests that there are over sixty OO methods in existence. The best known methods include; Booch (1991), Rumbaugh, et al. (1991), Shlaer and Mellor (1992), Jacobson, et al. (1993), Coad and Yourdon (1992) and Wirfs-Brock et al. (1990).

Many extravagant (and typically unsubstantiated) claims have been made for object-oriented techniques; most notably that the approach leads to a more natural design. For example, Rosson and Alpert (1990) claim that ' A particularly attractive aspect of OO design is that it seems to support a better integration of problem and solution' (p. 361)....whilst admitting that 'very little empirical evidence exists concerning the naturalness of objects as ways of representing problem entities' (p. 363).

Objects are the basic concept in OO approaches. Objects are defined as 'an encapsulation of attributes and exclusive services [behaviours]; an abstraction of the something in the problem space,...' (Coad and Yourdon, 1992). Davis (1993), like many OO theorists stresses that objects correspond to real world entities.

In OO design analysts / designers are encouraged to identify objects in the domain and use these as the basis for the conceptual model. Various advice is offered on how to spot objects. For example 'identifying classes and objects involves finding key abstractions in the problem space and important mechanisms that offer the dynamic behaviour over several such objects.' (Booch, 1992).

The analysts should 'develop an object model which describes the static structure of the system with classes and relationships. The dynamic model captures the temporal aspects, the functional model describes the computation '(Rumbaugh, et al. 1991).

'The designer looks for classes of objects trying out a variety of schemes in order to discover the most natural and reasonable way to abstract the system. ...In this phase the major tasks are to discover the classes required to model the application and to determine what behaviour the system is responsible for and assign these specific responsibilities to classes'. (CACM, 1990).

OO modelling focuses first of all on structure - it is the objects which are paramount. The system's processing is defined in terms of the objects - different classes of object allow for different types of processing. The notation for object diagrams varies considerably from Entity-relationship diagrams to 'roundtangles' to represent objects to cloud type diagrams. Sully (1994) provides a comparison.

One of the problems of describing and analysing the OO approaches is that they often employ a nomenclature which is unfamiliar and which uses a variety of concepts. Thus apart from objects and classes, OO approaches use concepts of scenarios, 'use cases', methods, messages, roles and so on. There is also a problem in distinguishing object-oriented programming and object-oriented analysis and design.

The claims for the benefits of OO techniques include abstraction and encapsulation (otherwise known as 'information hiding'). Objects are viewed from the outside and users need not be concerned about how they are implemented. Objects can send and receive messages; they encapsulate the structure and processing which enables them to deal with those messages. Other features of the OO paradigm such as polymorphism (that different object classes will treat the same message in different ways), inheritance, late binding and tight cohesion relate more to the programming benefits than to any conceptual benefits of the approach.

The most significant difference between the OO paradigm and the data-centred paradigm is that OO approaches avoid the separation of data from procedures typified by the data-centred approaches. Instead of the functional decomposition into processes on a dataflow diagram, OO decomposes a system into messages flowing between objects. However, a number of authors (e.g. Sully, 1994) describe two approaches to OO design; moving from a data-centred approach to an object model and going directly from a domain or application to an object model.

In the ATM example, a OO analysis would result in diagrammatic representation as shown in Fig. 6. The objects of interest in the application; person, card, account and ATM are shown as objects with the major messages which they can send and receive shown. The internal description of the objects (the structure of the objects in terms of the attributes which they have and the processing which the objects can perform) would be shown in separate object description tables.

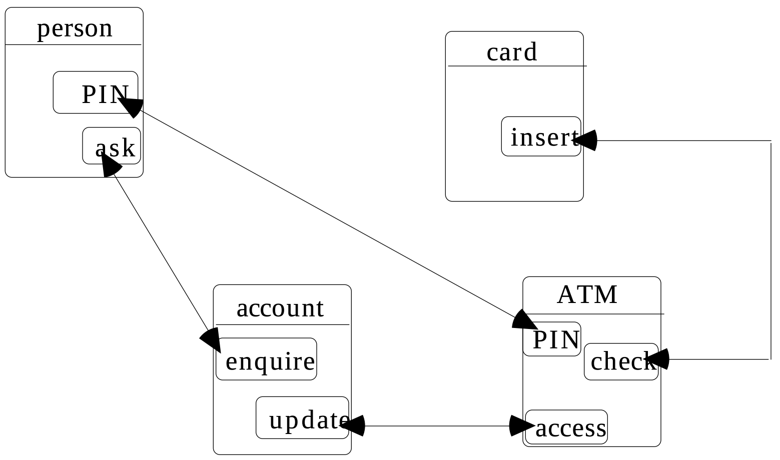


Figure 6 Object model of the ATM

The object-oriented paradigm again emphasises abstraction and generification through the notion of object classes. This is similar to the idea of generic tasks, where, for example, the task of entering a PIN is similar in all circumstances. Generification recognises, for example, that all bank cards will have common attributes and will send and receive similar messages. OO also recognise sub-types which inherit behaviours and / or attributes from the super-class. For example it there are various types of bank card which provide different types of facilities, these different types would inherit the basic characteristics of all bank cards. The focus of the approach is on structure rather than processing (unlike the task-based approaches). Objects also stress encapsulation - the 'black box' view approach of hiding information which is not relevant to a particular view of the object. The emphasis is on the existence of the object and on its presentation in terms of the messages which it can send or receive rather than on the abstract, internal view of the structure.

The attraction of the OO approaches is that the system can be described in terms which the users hould be able to understand. The objects which the analysts discovers are those that the user deals with in concrete terms. However, there are philosophical issues concerned with whether or not there is an objective reality to be modelled and even given an agreed reality to model, the analyst can find it difficult to know when an object has been successfully identified. Objects are also very popular in programming; in implementing a system. If they are clearly defined then the programmer can re-use objects which have been previously defined. The ability of objects to inherit the structure and processing of other objects means that the writing of code can be more efficient.

6. DATA -CENTRED APPROACHES

Data-centred models have existed in the world of database design since 1976 when Peter Chen published his paper on the entity-relationship model (Chen, 1976). Shortly afterwards, in 1979 the dataflow diagram was established (e.g. DeMarco, 1979) and in the early 1980s the entity-life history evolved (Rosenquist, 1982). Typically details of the entities, dataflows and processes are recorded in a data dictionary. All of these models use as their basic concept the notion of a data item or data element. The structure of the domain is abstracted as a network of entities and the functions of the domain by a network of dataflows and processes. Other representations associated with the data-centred approach include state transition diagrams (which come in a variety of guises). These focus on the data which is necessary to move the system from one state to another.

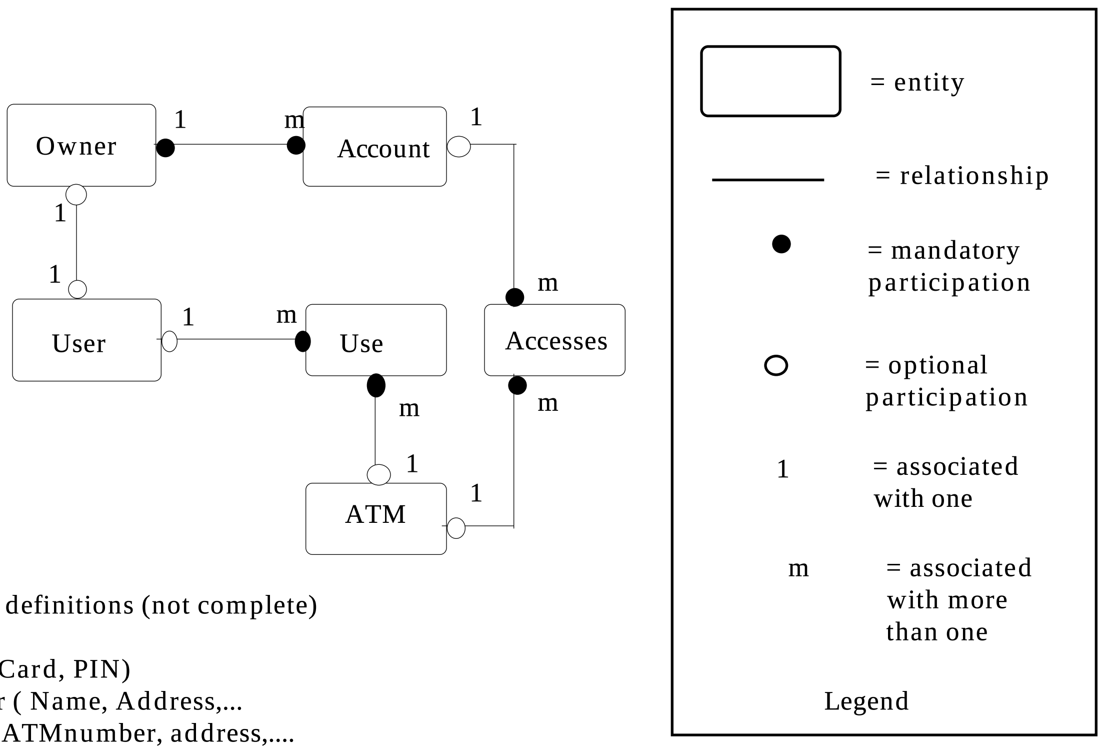
The basic concept of a data model is a data element (or data item). A data item is consists of one or more symbols, a name (and usually a more comprehensive description of the meaning of the data item) and a context. The name, description and context ascribe the semantics to the data item. Data items are generalisations of the actual and potential values which that data item can take. These values are known as the domain of the data item (Benyon, 1990).

For example, in the context of the ATM we may have some data, say '2341', associated with a name, PIN. This would allow us to represent that a particular PIN is 2341. This can be generalised in terms of the domain of PIN which may be any 4-digit number. The concept of a domain is very similar to that of an object in the OO approaches as the domain defines the operations which are allowable. For example in the ATM application we might define the data items PIN and Card where PIN has the domain of 4-digit numbers and Card has a domain of BankCard (and hence supports certain operations such as 'insert', 'swipe', etc.).

Another level of abstraction in data-centred approaches is provided by something which is usually known as an *entity*. Although some authors argue that entities are things which exist in the 'real world', others prefer to focus on an entity as a purer data-centred concept. An entity is an aggregation of data items, expressing the semantics that certain data items belong together. Once these groupings have been established, the analyst can refer to the collection of data items - the entity - by name, thus suppressing detail which would otherwise clutter the model. Relationships between entities - expressed in terms of the number of instances of one entity which are associated with the instances of another entity - can then be considered.

Figure 7 shows an entity-relationship diagram of the ATM example. It may be interpreted as follows. An Account (identified by Accountnumber) must have just one owner (identified by Name and Address), though an owner may have many accounts (but must have one). An (account) owner may be a (ATM) user. A user (defined as a Card + PIN) makes use of ATMs (identified by an ATMnumber). A use is defined as a user using an ATM. An ATM accesses accounts.

One of the interesting things which arises once one starts looking at the data is that further insight is gathered; for example, that the entity User may be different from the entity Owner. Indeed that we define a User as a person who is possession of (a valid combination of) PIN and Card and this may be different from the owner of the account.



Entity definitions (not complete)

User (Card, PIN)
Owner (Name, Address,...
ATM (ATMnumber, address,....
Use (Card, PIN, ATMnumber, Amountwithdrawn,....
Account (Accountnumber,.....
Accesses (Card, PIN, ATMnumber, Accountnumber,....

Figure 7 Entity-relationship model for portion of an ATM.

Representing the domain in terms of data items, how they are grouped together into entities and the relationships which pertain between entities describes the structure of the domain. Another way to model the domain is to look at how data flows between processes. A process, or functional, model made of data concentrates on the data which is strictly necessary for processing to occur.

Functional data models are abstractions of systems which do not show flow of control i.e. they strive to be as independent of the device as possible This may be contrasted with a model of the physical system (such as a flowchart) which models the movement of physical objects such as documents and the control flow (i.e. the sequencing of actions) of particular implementations. Functional data models concentrate on the data processing which is strictly necessary for some process to function in a given domain. For example, in order to get cash from an ATM, a user *has* to be validated and to have enough money in the account (Figure 8) and these things have to take place in sequence. The cash retrieval process, logically, needs both of these data items in order to function. In contrast, Present Personal ID and Present cashing instructions are not dependent on each other. Importantly dataflow models *do* show the data which is necessary for certain processes to happen (compare this representation with Figure 4 which does not show the data which needs to flow between processes.)

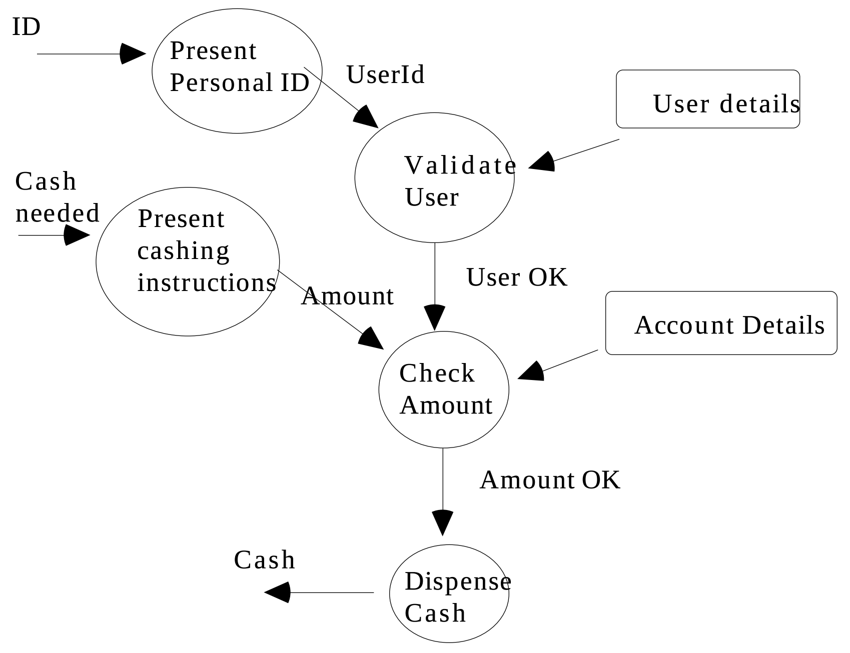


Figure 8 Portion of Dataflow diagram for ATM

Several well-developed techniques are available to assist the designer develop a conceptual data model. Normalisation enables the analyst to explore the semantics of the data by examining the relationships between data items (Date, 1986). The entity-relationship (ER) model represents aggregations of data items and the relationships which exist between them (Benyon, 1990). Dataflow diagrams (DFDs) concentrate on consistent and complete descriptions of data flow, showing the minimum amount of sequencing required for processing to occur and the data dictionary describes the semantics of data items, stores and processes (DeMarco, 1979). Entity Life Histories (ELH) use a structure chart notation to show the behaviour of entities over time.

7. DISCUSSION

Developing an appropriate domain model is critical for the success of interface design. For model-based approaches to HCI a domain model is strictly necessary. Even where a model-based approach is not employed, the analyst needs to be able to abstract the domain and represent it in a way which facilitates analysis of the problem situation and design of an effective solution.

The purpose of developing a conceptual model of the application domain is to support both the analysis of the extant system (understanding the requirements of the whole human-computer system) and the design of a replacement system. The two important requirements of a modelling method are

(i) that it must be at an appropriate level of abstraction for these purposes

and

(ii) that it should support a useful and relevant conceptualisation of HCI.

The task-based approaches focus on the sequences of actions which are necessary to accomplish a goal. They do not represent the data or information which flows between these actions. Nor do they represent the knowledge required to undertake an action successfully. The notation of task-based models includes control flow (i.e. the sequencing, selection and iteration of actions). Task-based models - because they show control flow - are device dependent; the sequencing of actions is dependent on the device being used to achieve the goal. Of course the analyst can abstract from a number of existing devices and produce a task model of a generic task - the approach recommended by Lim and Long 1994. This idea is also present in cognitive task models; for example, Kieras (1988) produces generalised methods and de Haan (this volume) also discusses such abstractions. One of the problems with task-based models is that different users view tasks in different ways. Draper (1993) found a variety of interpretations of a simple word processing task and notes 'If the notion of task is so unstable and indeterminate in small scale word processing...then what hope is there of using it as a theoretical concept...? (p. 207).

Object-oriented approaches focus on objects in the existing world and the messages which objects can send and receive. Analysts are encouraged to abstract from the existing system, but are not given any mechanism by which to do this. The lack of precision in object definitions means that ambiguity may arise. There are no rigorous techniques through which an analyst can be sure that the model does not contain any redundant duplication of concepts. Another problem with object approaches is that it is not easy to see how an object definition at the conceptual level can be split between human and computer at the physical level.

Data-centred approaches provide robust mechanism for detecting poor representations both at the abstraction level of data items and at the entity level where the ER diagram facilitates a number of useful operations which can be used to determine useful entities which are *not* obviously present in the 'real world'. The focus of the data-centred approaches is on information structure and information flow. The main problems with data-centred approaches is that the entities in a structural model of the domain or the dataflows in a procedural model may not correspond to user concepts or user tasks.

Besides the material from which the model is constructed - tasks, objects or data - the approach to HCI development needs to be considered. The approaches may be classified as structured, iterative or 'toolkit' (Benyon and Skidmore, 1987). A structured method offers a prescribed step by step approach. An iterative approach emphasises the cyclical nature of development. A tool-kit approach encourages the selection of models which are appropriate for the purpose at hand.

One of the difficulties with examining the different paradigms and methods is that they are frequently confused. First, there is a confusion between structured methods and data-centred approach. A structured method can be based on any modelling formalism. Thus MUSE is a structured method based on tasks. The data-centred approach does not have to be structured; one can undertake data-centred analysis and design without following a method (e.g. a tool-kit approach).

Second the same notations are used to support multiple methods. For example ER models are used in a number of object-oriented approaches and structure charts are used in both data-centred and task-based approaches. Third, there are many hybrid techniques. For example, in the task-based technique KAT, Johnson (1993) explicitly states that the knowledge required by users to undertake a task must be represented and uses object models for this. Many of the methods described in this volume combine aspects of the different paradigms. For example, the object approach of Palanque and Bastide shows the data which flows between objects. Sutcliffe's task-based approach also shows data whereas Paterno's task-based approach uses objects. Fourth, different authors usurp the advantages of other methods, claiming them for their own. For example, the claim that the object paradigm takes a modelling point of view is equally true for both the data-centred and task-based approaches. Certainly task-based approaches are more procedural, but they clearly take a modelling perspective.

In order to develop effective human-computer systems we need to develop techniques which capture a conceptual model of the whole domain. These conceptual models must be expressed using a medium which is applicable to both agents and devices. The conceptualisation of HCI presented in section 2 emphasised that communication in agent-device systems takes place through the exchange of signals and that both the signals and the systems need to be described at various levels of abstraction.

Data modelling makes the semantics of, and relationships between data items explicit. At the level of abstraction appropriate for human-computer systems, data are the signals which are exchanged between systems. Thus data-centred approaches are suitable for describing the abstract view of both agents (whether human or not) and devices. For the presentation view concerned with the interaction with users, data must be organised into user-centred objects and processing must be organised into user-centred tasks. For the internal presentation views both data-centred and object-oriented models have their place.

The tasks which we design when we develop computer-based systems must be openly declared and evaluated with users. The objects which users think about and interact with when using the system must correspond to objects which they understand and use. However, shifting attention from existing tasks and existing objects to the tasks and objects of the new system requires an abstract view to be taken. Developing appropriate conceptual models of domains demands that the material from which the model is constructed is suitable for that purpose. A well-constructed, abstract model means that the allocation of tasks, knowledge, functions and control to human, to artificial agent or to a device becomes a more considered, rational and open activity.

REFERENCES

Amodeus (1994) Glossary of terms. At http://www-lgi.imag.fr/Les.Groups/IHM/AMODEUSGlossary/GlossaryEntries/task.html

Barlow, J., Rada, R. and Diaper, D. (1989) Interacting WITH computers *Interacting with Computers* 1 (1) 39 - 42

Bench-Capon, T. J. M. and McEnery, A. M. (1989a) People interact through computers not with them *Interacting with Computers* 1(1) 31 - 38

Bench-Capon, T. J. M. and McEnery, A. M. (1989b) Modelling devices and modelling speakers *Interacting with Computers* 1(2) 221 - 224

Benyon, D. R. (1992b) Task Analysis and Systems Design: The Discipline of Data. In *Interacting with Computers,* 4(2) 246 - 259

Benyon, D. R. (1993) A Functional Model of Interacting Systems; A Semiotic approach in Connolly, J. H. and Edmonds, E.A (eds.) *CSCW and AI* Lawrence Erlbaum, London,

Benyon, D.R. (1990) *Information and Data Modelling,* Blackwell Scientific Publications, Oxford

Benyon, D.R. (1992a) The Role of Task Analysis in Systems design in *Interacting with Computers* 4(1),

Benyon, D.R. and Skidmore, S. R. (1987) A Tool-kit approach to information systems development. *The Computer Journal* 31(1)

Booch, G. (1992) *Object-oriented design with applications* Benjamin/Cummings

Browne, D. (1993) *STUDIO* Prentice Hall

CACM (1990) *Communications of the ACM* 33(9)

Card, S. Moran, T. P. and Newell, A. (1980) *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ

Carroll, J.M. (1990) Infinite detail and emulation in an ontologically minimised HCI in J.C. Chew and J. Whiteside (eds.) *Empowering People; CHI '90 Proceedings* ACM press

Checkland, P. B. (1981) *Systems Theory, Systems Practice*. Wiley

Chen, P. P-s (1976) Towards a unified theory of data *ACM Transactions on database systems* vol 1 (1) 9 - 36

Cherry, C. (1966) *On Human Communication* (2nd edition) MIT press

Coad, P. and Yourdon, E. (1992) Object-Oriented Analysis. and edition Prentice-Hall

Coutaz, J. Nigay, L. and Salber, D. (this volume) Agent-based agent modelling for interactive systems

Date, C.J. (1986)*An introduction to Database Systems* 4th edition Addison-Wesley

Davis, A. M. (1993) Software Requirements; Objects, Functions and States. Prentice Hall

de Haan, G. (this volume) ETAG-based design: User Interface design as User Mental Model design

DeMarco, T. Structured Analysis, Systems Specification. 1979 Prentice Hall

Diaper, D. (1989) (ed.) Task Analysis for Human-Computer Interaction. Ellis-Horwood, Diaper, D. and Addison, M. (1992) Task Analysis and Systems Analysis for Software Development. In: *Interacting with Computers* vol 3(3) pp 124 - 139,

Draper, S. (1993) The notion of task in HCI. In Ashlund, S., Mullet, K., Hendersen, A., Hollnagel, E. and White, T. (eds)*Proc of Interchi '93 Adjunct proceedings* ACM press

Eco, U. (1976) *A Theory of Semiotics* Indiana University Press, Bloomington.

Fischer, G. (1989) Human-Computer Interaction Software: Lessons Learned, Challenges Ahead *IEEE Software*, (January) pp 44-52

Goedicke, M. and Sucrow, B. (this volume) Towards a flexible software architecture of Interactive Systems

Hix, D. and Hartson, H. R. (1993) *Developing User Interfaces* Wiley

Jacobson, J., Chistensen, M., Johnson, P. and Overgaard, G. (1993) *Object-Oriented Software Engineering.* Addison-Wesley

Johnson, P. (1992) *Human-Computer interaction; Psychology, Task Analysis and Software Engineering.* McGraw-Hill, Maidenhead, UK

Kangassalo, H. (1983) *Structuring Principles of Conceptual Schemas and Conceptual Models.* In Bubenko, J. (ed.) *I*nformation Modelling Chartwell-Bratt

Keeler, M. A. and Denning, S. M. (1991) The challenge of interface design for communication theory: from interaction metaphor to contexts of discovery *Interacting with Computers* 3(3) 283 - 301

Kieras, D. (1988) Towards a practical GOMS model methodology for user interface design. In *The Handbook of Human-Computer Interaction* Helander, M. (ed.) pp 135 - 158 North-Holland

Laszlo P. (1969) *System, Structure and Experience* Gordon and Breach Science Publishers, London, 1969

Laurel, B. (1990) Interface Agents: Metaphors with Character in B. Laurel (ed.) *The Art of Human-Computer Interface Design* Addison-Wesley

Lim, K. Y. and Long, J. (1994) *The MUSE method for usability engineering* Cambridge University Press

Long, J, Cognitive Ergonomics and Human-Computer Interaction. In P. Warr (ed.) *Psychology at Work* 1989 Penguin, Harmondsworth.

Lyons, J. (1977) *Semantics* Cambridge University Press,

Lyytinen, K. (1983) *Reality modelling considered harmful - the need for a linguistic framework*. In Bubenko (ed.) Information Modelling. Chartwell-Brat

Maes, P. (1994) Interface Agents. In *Communications of the ACM*)

Maes, P. and Kozierok, R. (1993) *Learning Interface Agents* AAAI '93 Conference on Artificial Intelligence, Washington,

McBride, N. (1994) *Bridging the gap between structured analysis methods and object-oriented analysis.* In Object Technology Transfer Alfred Waller,

Moran, T. Command language grammar: a representation for the user interface of interactive computer systems 1981 *International Journal of Man Machine Studies 15* 3

Preece, J. J., Rogers, Y. R., Sharp, H., Benyon, D. R., Holland, S. and Carey, T. (1994) *Human-Computer Interaction* Addison-Wesley

Rosenquist, C. J. (1982) Entity-Life Cycle Models and their applicability to information systems development life cycles *The Computer Journal* 25 (3) 307 - 315

Rosson, M. B. and Alpert, S. R. (1990) The Cognitive Consequences of Object-Oriented design in *Human Computer Interaction* vol 5 345 - 379

Rumbaugh, J., Blaha, M., Premerelani, W., Eddy, F. and Lorensen, W. (1991) *Object-Oriented Modelling and Design* Prentice-Hall

Shannon, C. E. and Weaver, W. (1949) *The Mathematical theory of Communication*, University of Illinois press

Shepherd, A. Analysis and Training in information technology tasks in Diaper, D. (ed.) *Task Analysis for Human-Computer Interaction*. 1989 Ellis-Horwood

Shlaer, S. and Mellor, S. J. (1992) *Object Life cycles*. Prentice-Hall.

Stamper, R. (1977) *Information* B. T. Batsford Publishers, London

Storrs, G. (1989)Towards a Theory of HCI In *Behaviour and Information Technology*, 8(5) pp 323-334

Suchman, L. (1987) *Plans and situated actions: The problem of Human-Machine Communication* Cambridge University Press

Sully, P. (1994) *Modelling the World with Objects* Prentice-Hall

Sutcliffe and McDermott (1991) Integrating methods of human-computer interface design with structured systems development International Journal of man Machine Studies 34 631 - 655

Sutcliffe, A. (this volume) A method for task related information analysis

Wilson, M.D., Barnard, P. J., Green, T. R. G. and Maclean, A. Knowledge-Based Task Analysis for Human-Computer Systems in van der Veer, G. C., Green, T. R. G., Hoc, J-M and Murray, D. M. (eds.) *Working With Computers: Theory versus Outcome*. Academic Press (1988)

Wirfs-Brock, R., Wilkerson, B. and Weiner, L. (1990) Designing Object Oriented Software. Prentice-Hall