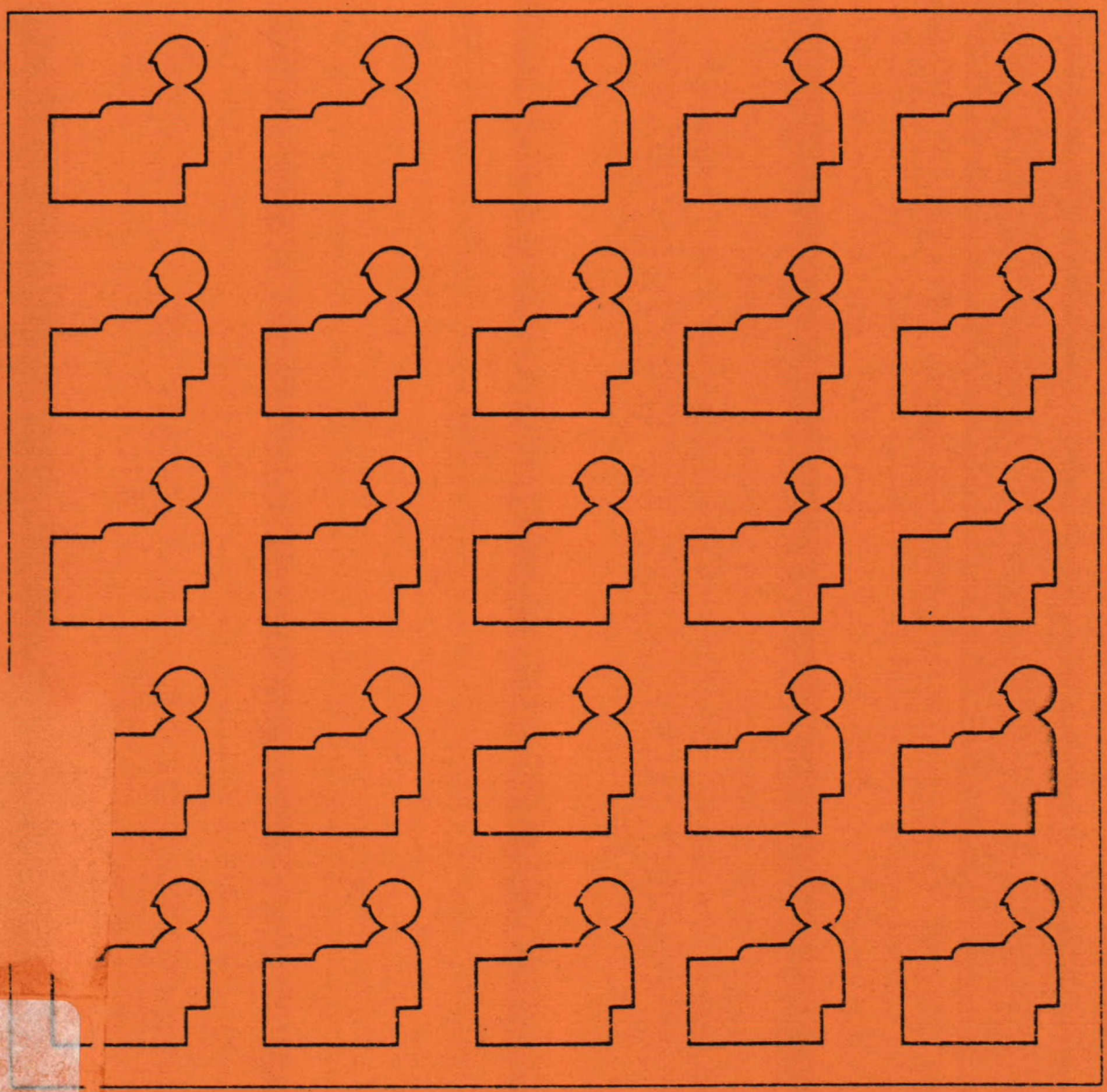
OFFICE COMMUNICATIONS SYSTEMS PROGRAM

PROGRAMME DE LA BUREAUTIQUE

USER INTERFACE DESIGN FOR OFFICE COMMUNICATIONS SYSTEMS

Dr. Tom Carey



Government of Canada
Department of Communications

Gouvernement du Canada
Ministère des Communications

Canada

Acknowledgements: Grant Boyd was contract officer, content moderator and bureaucratic troubleshooter; Roger Tessier, Art Benjamin and André Dubois were instrumental in initiating this series of reports.

Sabine Rohlfs made important contributions to the content of chapters two and three; Dick Mason influenced both the style and content of chapter three.

Preparation of this report was handled most competently by Janis Pettis, with assistance from Terry De Haan.

Dick Foster, Blair Nonnecke and John Versterfelt provided helpful comments on earlier drafts.

Table of Contents

Summary

1. The Elusive All-Purpose User Interface..... 1.1

   1.1 Wish Lists, Guidelines, Principles and Standards..... 1.1

   1.2 Conceptual Model Integration..... 1.11

   1.3 Integrating Applications..... 1.19

   1.4 Integrating Excursion Tasks..... 1.22

2. Design Principles for User and Interface Evolution..... 2.1

   2.1 Multifunction Office Systems..... 2.1

   2.2 User Evolution..... 2.6

   2.3 System Design Factors..... 2.9

   2.4 Product Design Factors..... 2.11

3. User Interface Testing..... 3.1

   3.1 Lab Exercises..... 3.1

   3.2 Testing Prototypes..... 3.6

   3.3 Interface Testing Techniques..... 3.15

   3.4 Interface Malfunction..... 3.22

4. Sample Elements in Office Conceptual Models..... 4.1

   4.1 Conceptual System Objects..... 4.1

   4.2 Office Procedures..... 4.7

   4.3 Future Office Models and Tools..... 4.16

Appendix: Summary of User Interface Tools for OCS

References

Summary: User Interface Design for
Office Communications Systems

This report is the second in a series of three reports prepared for the Office Communications Systems (OCS) Group of the Dept. of Communications. The first report, User Interface Components for OCS looked at the principles of human-computer interactions, via a framework of interface levels. User Interface Tools for OCS, the third report, considers software tools, formal techniques and knowledge representations as applied to user interfaces.

This second report, focuses on designing user interfaces for the next generation of office system products. Drawing on research in the human factors of computer systems and on experimental OCS projects, we can map out some of the design decisions for OCS products and their implications.

In the first chapter we ask to what extent a standard interface for OCS products can be developed. An initial section outlines the role of interface standards versus guidelines or principles. Then we demonstrate that different office tasks require different conceptual models of various entities; accordingly, a multifunctional office system must support more than one conceptual model of office entities to seem natural to users. Given this limit on conceptual model integration, we examine the implication for integration of office services into a single user interface.

Our conclusion is that manipulation of information or objects is more resistant to integration than information or object management. Another good candidate for standardization is the user task of navigating within the computer system: finding out "how do I get there from here", when there and here are relatively vague in the user's mind. We outline two experimental systems which focus on these 'excursion task' activities.

The second chapter explores a set of design issues for OCS. It begins with a mention of system scope--a designer must achieve a proper mix of central, local and personal facilities. Various views of user evolution are presented next, with their implications for a progression of interaction styles. The third section briefly reviews some system design questions like impacts on organizational structures and participative design. The design framework from the User Interface Components report is re-examined, with interest on the design process and its staged outputs. Design pragmatics, involving conflicting interests and numerous iterations to undo past errors, is the last topic considered.

Given the current inadequate state of our knowledge about human factors, we must rely on extensive testing of a user interface to reduce incompatabilities between user and system, a topic surveyed in chapter three. Some of this testing, done as experiments in research labs, need not be repeated every time we design an interface. But the

particular mix of ingredients in any one system will always demand testing with prototypes at various stages. This testing process can be improved by a set of techniques surveyed in the third section of this chapter. Finally we consider how to interpret and use the information about interface weakness which we obtain during the testing process.

The fourth chapter provides a conceptual perspective on current experimental OCS projects. We do not present any extensive examples--these are available in the various references cited. The focus instead is on understanding the conceptual differences between various systems and what the effects might be. These conceptual differences include the origins of the system concepts (distributed data bases, message systems or document processing), and the mechanism for specifying office procedural information.

An understanding of these design considerations helps us to see the decisions required in OCS products. That is the real challenge for designers: to see beyond their personal view of the 'ultimate interface' for the 'real office', to appreciate how their designs fit within the larger framework of diverse office communications systems.

1. THE ELUSIVE ALL-PURPOSE USER INTERFACE

Is there a user interface fit 'for all seasons', a universal OCS access tool which opens up a full spectrum of services to a full variety of users? The answer depends on the level we which to address the question: at the conceptual model level the answer appears to be no. At the dialogue level it seems the answer is close to a yes: integrated interfaces are appearing which attempt a consistent view of the user/system interaction. This includes assistance and excursion tasks.

1.1 Wish Lists, Guidelines, Principles and Standards

User interfaces, once badly neglected, now may be suffering from too much attention. Every new software product announcement contains an obligatory reference to 'ease of use', 'user-friendly' or 'ergonomics'. Most users feel that only experience with the product will serve as an evaluation. Use of a genuinely poor interface reveals its failings; use of a merely mediocre interface may be accepted, since the additional capability of a superior interface may not be evident. Paradoxically, use of a good interface often stimulates the appetite and may result in more user demands!

User interface design is still emerging as an area of study, and remains encumbered by proverbs and folklore. Designers seeking training will find some help in cognitive psychology, some in software engineering, and some farther afield in traditional ergonomics, graphic arts, organizational design, industrial sociology, etc. Managers looking for practical measures will find a mixture of wish lists, guidelines, principles, and standards.

Wish lists -- Much of the early literature on human factors in user interfaces appears now to be in the 'wish list' category: a set of desirable properties described by their effects on users. For instance, one checklist of office system usability factors [1] suggests that user interfaces be

"Approachable

Suitable (to the task)

...Require little memorization

...Supportive..."

etc. There was certainly a need at one time for interface designers to be sensitized to these kinds of user needs. But such a check-list offers at best desirable factors, not guidance in achieving or implementing them.

The checklist can be made more detailed:

"Display formats for data input should be designed so as to minimize user actions required for cursor movement from one entry field to the next" [2] . This still leaves us to subjectively evaluate whether a given design 'minimizes', and no help in techniques to implement a better scheme.

Guidelines - there have been some attempts to specify a set of design guidelines which would provide the know-how lacking in the wish lists. For example:

> "Rule 15: Provide a reset command that clearly aborts the current activity back to a convenient checkpoint..." [3]

or "In presenting data or small display screens, no more than 50-55 characters per line should be displayed..." [4]

Software engineers are not used to working with explicitly detailed guidelines to the same extent as traditional ergonomics might be. The tradition of individual autonomy may account in part for the lack of agreement on many guidelines. The compilers of the guidelines from which the last quote was taken were obliged to mark those entries on which their sponsor committee could not reach agreement. Every major section contains these

'arguable' guidelines.

There are additional contributing factors to the failure of guidelines to have wide impact:

-other human factors handbooks are based on extensive quantitative field data. "Empirical data on human-computer interfaces are simply not available for many of the questions on which designers need guidelines. A quantitative human factors reference handbook for interactive systems design appears to be well beyond our current capability". [5]

-the design decisions are not easily separable. Thus one valid guidelines may recommend providing the user with all necessary information to determine system state. Another may recommend that screen displays not be cluttered with more than a certain density of information. In practice the two guidelines may frequently be in conflict.

-the right level of detail in guidelines is very hard to judge. Too much detail of course negates the impact of the useful information. Most designers would be hard-pressed to think of any alternative to the following guideline:

"The displayed cursor should be stable, i.e. should

remain where it is placed until moved by the user (or computer) to another position." [6]

On the other hand, less detailed guidelines might miss subtle decisions like the following (from the same list):

"User action confirming entry of multiple data items should result in input of all items, regardless of where the cursor is placed on the display."[7]

Regardless of whether one agrees with this guideline, on reflection it is a decision we would want to make consistent (one way or another) across a user interface.

Principles - what is needed is not just know-how but know-when. This has two sides:

-know when decisions are being made. Design guidance can only be effective when design decisions are explicitly recognized.

-know when certain guidelines should take effect, by understanding the principles on which they are based. For example, the guideline above on line length is based on eye movement patterns and the difficulty of returning to the beginning of the next line when the line length is too great. Thus the guideline has more importance in displays of long text than in single line messages surrounded by blank space.

Design principles must be linked to the design cycle, so that design questions are not missed. A set of design principles must "identify issues, suggest alternatives, and present (where they exist) hard human-factors data at the point in the design process at which this information is most relevant."[8]

In a companion report, User Interface Components for OCS,[9] an initial effort was made towards a set of design principles for the limited domain of office systems. No attempt at an 'ultimate interface' was made; rather the questions raised within the design cycle were outlined, along with selected principles (and references to more). The principles were embedded in a top-down design framework which attempted to give priority to the more fundamental decisions about the interface. That design cycle is reviewed in Chapter 2.

These initial efforts need to be tested in practice, revised, and extended. Additional inputs to the design process include

-design techniques, the body of engineering knowledge on how to solve specific problems

-better formalisms within the discipline of user interface design. Some of these are discussed in another report.[10] At an informal level, consider the notion of 'operation' a semantic unit which alters the state of a conceptual object.

Several operations may accomplish the same function of achieving a target state; carrying out the operation may entail a sequence of commands and responses.

In designing operations, the set of decisions to be made includes at least the following [11]

a) number of places in the operation predicate (roughly, the number of parameters)

b) length of commands sequence in the operation

c) invertibility

d) commutivity

e) transitivity

f) structure of appropriate command sequences

g) action type (move, create, remove, etc.)

A particular operation can change its form depending on the way it comes to be used.

We have a much fuller vocabulary for addressing lower levels of the interface like syntax and physical action than we do at the level of semantic unit or conceptual models. It is in these latter areas that the most fundamental

breakthroughs will be made.

-knowledge from other fields which establishes the cause-effect relationship behind the principles. For example ,from linguistics we could borrow ideas like [12]

a) dependency grammars to explicitly compare the syntax of user languages with that of natural language

b) semantic diagrams to explain the meaning of words in the interface language

c) linguistic parameters like valence of verbs, to help develop the characteristics of operations as described above.

Standards - While we await better design principles, software development organizations need internal standards. These could be restricted to a given project, or apply across numerous applications to be offered to the same group of users. The level of consistency the users observe is discussed below in section 1.3. From the organizational perspective standards can represent

-co-ordination amongst many developers working on the same application. This has the ultimate aim of increasing consistency to reduce complexity for the users.

-reusability of interface components across applications, both for users and developers.

-capitalization of the knowledge of the best designers. For example, a guideline cited above prescribed certain line lengths for displays. Depending on the application, a better scheme might restrict normal data to start in column 21 of a display, reserving a wide left margin for subtitles, prompts etc. [13] This convention, articulated by a graphics designer, makes this expertise available across an organization when applied as a standard for certain applications.

-standard development cycles for interfaces to facilitate monitoring, feedback and management control.

Standards, unlike guidelines, must be enforceable. A standard can be enforced by

-requiring use of an interface management system, which by providing certain lower-level features enforces a standard [14]

-use of quantifiable metrics, which developers must apply to prove that their designs fall within the standard. These measures are currently only applicable after implementation of a sizable part of the interface. Testing with appropriate sample groups can then assess usability. Techniques for testing are discussed in Chapter 4. Figure

1.1 shows one proposed System Attribute Specification for usability. [15]

-use of standard lists of allowed words, syntax, etc. For example, recent human engineering guidelines for management information systems in the U.S. Army provide a list of 39 acceptable command words, and a six page list of potential commands which they are to replace. [16]

-a 'usability committee', which receives proposed designs and applies a standard review. Review items typically include [17]

a) command syntax and semantics

b) display and function key design

c) system message texts

d) printed output.

The committee works best when its standards are communicated beforehand to designers, so that it functions in a review and not a design role. Sometimes the committee defaults to a committee of one: the manager responsible for a project. The manager normally has more than enough to do without looking at interface details; any large project requires independent reviewers.

-some standards may be policed by outside certification. This is common in programming language compilers and hardware interfaces. An effort for user interfaces has been initiated by the CODASYL committee.[18] Their development of a video interface model for an operating system has only recently begun (i.e. is still at the wish list and preliminary guideline stage). Other trade organizations, even outside the computing industry, may follow.

1.2 Conceptual Model Integration

A dominant theme in discussion of projected office system use is the integration of various application services into a consistent interface. The intent is to minimize learning overheads and promote ease of use. Much of the work on guidelines and standards with software development organizations in intended to achieve a high degree of consistency across products.

In our framework of user interface levels, true integration within the system would have to begin at the conceptual model level. We need to consider what kind of integration is appropriate at that level before considering (in Section 1.3) the levels at which integration is usually examined--dialogue style and semantics, syntax and language, and physical action or display.

Conceptual models as metaphors: the user's conceptual model of system objects and facilities must be closely tied to the

objects and functions of the task world. The interface designer must build an explicit model and convey it to the user [19].

Analogies are frequently used to help initially communicate the model[20]. For example, a user of an electronic file cabinet might be told that the electronic files are like manual office files, the electronic wastebasket is a counterpart to the physical wastebasket, and so on. The analogy must usually be left behind when it has served its purpose as an orientation device. Users would be told that the electronic file differs from its physical counterpart in that a document may be simultaneously present in more than one file; changes to the document may automatically appear in each file, unlike physical files in which only static copies can be placed in different files.

The concept of the old physical files was derived from two different views: the file as container and the file as relationship. As a container, the file physically held documents, and no document was physically present in more than one disjoint container. But the file was not referenced arbitrarily by a number: it was given a name that expressed the relationship of documents within it.

In searching for a document manually the container view was dominant. In the electronic system, the relationship view becomes dominant. We are therefore building a new

model in which old notions like copying and deleting documents must be rethought--is a copy static or does it change along with the original? How do I delete a document completely from the file system (and not just from one relationship) when it exists simultaneously in several files?

Rather than encourage users to build a conceptual model of the file system from their model of a manual system, we could have used slightly different analogies and build a different conceptual view. We could have constructed the file system so that all documents belong to one 'container' file, but there are 'relationship' files which are index lists of document names. A document would exist in only one container; a copy could be made but it becomes a separate document. The delete function could be applied to an index list to remove a relationship; applied to a container it removes the document and appearances of its name in relationships.

We don't want to argue here about which model is more efficient, whether a synthesis is possible, or which one is more marketable to new users. The point is just that the models are different, in terms of basic object types, organization of functions, etc. The way we construct operation semantics at lower interface levels depends on which model we want the user to have.

Given that the same application could have different conceptual models, how do we want to design the models to promote integration? We can apply the following vocabulary to conceptual models for different applications

-the models are compatible if they are separate but not contradictoy, for example, a simple service which retrieves time of day or ongoing weather may not share any common structure with a filing service

-the models are coherent if they share some common structure but cannot be integrated into a single whole

-the models are consistent if they can be seen as specializations of a more general concept, i.e. they can be integrated into a larger whole

For instance, we may have facilities for manipulating documents of text or line drawings. Our models of the documents could be made consistent if the same kinds of functions and organization are to be applied to each if they are organized hierarchically, say; and we are going to be editing the documents. The models could be consistent even though the mechanisms for input could be separate and the property choices to be made could be different (typeface for text, width of line for drawings).

On the other hand, if the line drawings are to be animated there is a set of facilities for them which have no counterpart for text. The most we could have then would be coherent models of the two domains. (The vocabulary of coherence and consistency comes from the study of metaphor[21], and it is suggestive to think of our conceptual models as metaphoric in nature .)

We will apply these terms to a popular model of some office activities, the electronic desktop.

Desktop as model: the electronic desktop model was pioneered in commercial products by the Xerox Star workstation and adopted by Apple's Lisa system.

The major ideas are:

-available objects, typically files and documents, are represented as graphic symbols on the desktop (display screen)

-when in use, pages of the documents appear on the screen possibly overlapping as they would on a desk

-movement of documents or their parts is the major operation for object management

-objects, properties and commands are meant to be visible to the user, especially in their altered state

after a change.

Is the electronic desktop the foundation for a consistent conceptual model across different applications? The Xerox experience suggests this will not, in fact cannot, be the case. "In a functionally rich system, it is probably not possible to represent everything in terms of a single model...Star's record-processing facility cannot use the physical-office model because physical offices have no 'records processing' worthy of the name... Therefore we invented a different model..." [22].

The models of records processing and document processing in Star are thus coherent but not consistent, using the terms in the technical sense above. Records processing, involving common operations in sequence for records in a file, has no real analogue in documents. 'Cutting and pasting' operations on documents are not implemented for sharing across record files. The models are kept coherent by using documents to define and display record files.

Another view of the difference between the models comes from noting that the manipulations desired on documents are easily visualized, and the Star ensures that they are visible. But the relationship of records within files or processing records in sequence is not to easy to visualize. Some manipulations of record files which could not be made easily visible, like specifying joins across files, were

omitted to preserve the conceptual simplicity of the interface [23] .

A different model of course might allow visualization of these operations [24] . But some concepts like event and time will not be easily treated in a consistent visual way. It is important that these kinds of concepts not be distorted to fit an interface style: recent research confirms that the way in which concepts are represented affects the ease with which users can manipulate them . As we move past records processing to user-created office procedures, it is not clear what kinds of representation can be effective, or how they can be linked as coherent models.

When the conceptual entities are restricted to objects, their management can be treated consistently by stressing visibility. As events, timings and actions need to be managed, the electronic desktop has to be extended to the true electronic office. No standard model seems likely to emerge. We survey some proposed models in Chapter 4.

Other potential common areas: as well as object management, other common aspects of conceptual models may be utilized to increase coherence. For example, the amount of change expected for various objects and attributes represents a fundamental aspect which may be similar across different views of the same information.

This is illustrated by a conceptual model for the professional tasks of accounting professionals, developed by D.N. Podger.[25] The accounting process model was derived from study of accountants at work. The author identified three zones of activity:

-an inner zone of basic values, objects and principles which the system must not allow to be overruled.

-an intermediate zone of general procedures which might be customized via user function.

-an outer zone of specific procedures which might undergo more frequent alteration.

(These zones have some overlap with the scope of processing discussion in section 2.1)

From this task analysis, a set of functions can be defined which is local to a given task group. We therefore do not expect these facilities to be shared across applications, although their definition structure might well be.

While the contents of the different zones of change may differ, we might find similar mechanisms to direct the change: the outer zone for instance might always be parameterized operations, while the intermediate zone may require user-written procedures to implement change.

1.3 Integrating Applications

The previous section illustrated that conceptual models are unlikely candidates for uniformity. Differences at the conceptual interface level imply that custom interfaces for specific tasks will always prove more effective than any standardized interface that cannot be tailored.

On the other hand, a good deal of user activity (even in different conceptual realms) is common. Objects are created, removed, related, displayed, etc. While the structures of the displays or relations may be diverse, there remains a set of common operations for managing them. In keeping with the notion of consistency in the previous section, these common operations should be consistently invoked. Typically this is done by defining generic operations which have equivalent semantics across a broad range of object types. The challenge then becomes to construct an interface dialogue level which complements both the common object management operations and the specific ooject manipulation operations.

The prooiem is compounded where different applications are to run concurrently. We then want an interface which is integrated as well as consistent. An integrated interface presents a single dialogue level view to the user. Objects are transferable between applications, although their semantic properties may change.

Integrated applications will be subject to the usual tradeoffs. For example, integration of data objects will require additional processing, and generic operations may not be as powerful for any one application as a customized set. While integration at the various interface levels appears natural, there are some inherent challenges and limitations.

Integrating data objects: The simplest route to (practically) integrating data objects is to reduce all objects transferred across applications to a lowest common denominator of structure. This is typically characters and numbers in text. Graphs and spreadsheets can be moved into documents by this conversion, but if the data is edited in some way (say two spreadsheet columns swapped) then they cannot always be moved back.

Sometimes the one-directional movement starts with the document, with the target a formatter for addition of other media like facsimile or for typesetting. Movement in the other direction is more difficult: making an editing change on the typeset version cannot usually be done by sending some small portion back to a word processor. The change must often be made to the original document and have it all formatted again. (usually a time-consuming process.)

It is now possible to maintain data on a workstation in more than one application at the same time. A spreadsheet is moved into a document, but future changes in the spreadsheet application are automatically transferred to the document. Keeping track of these structural views simultaneously will become more complex as record structures and data relations appear at the workstation.

Integrating dialogue styles: an interaction style optimized for applications stressing visibility, like the Xerox Star workstation, is not necessarily optimal for dialogue itself. A choice to emphasize marking and selecting leads to use of command menus for dialogue. Practiced users might want to type commands, but to maintain consistency we might have to allow typing of parameter values and thus open the door to more complicated specification of document pieces. Construction of office procedures may be more effective if the editor program has some structural semantics built in, but how do we relate this to the structure of other documents?

The search for generic operations also becomes more difficult as the application realm becomes less tangible. Can move and copy operations be applied to move links between records or to copy values between tuples in a relation?

Integrating language and physical action: at the language

and physical actions level it is easier to standardize on common features. Often this is implemented through a user interface manager which provides standard parsing and device handling characteristics.

In the user language, one has to try to choose command names whose syntax or use in natural language follows the prescribed standard. The designers of the Xerox Star illustrate the problem with the example of printing a document.[26] This is performed using the generic operation of moving the document's symbol (icon) to the printer symbol. In other cases, the move operation removes the symbol from its original position. But requesting that a document be printed normally sends a copy to the printer and leaves the document in its existing location. One could require the user to make a copy before printing to ensure consistency, but the designers chose instead to allow the symbol to remain.

At the physical level issues, like the choice of function keys need to be considered in integration. There have been attempts to standardize a 'universal' set of keys [27]; other key sets are customized to a given application [28]. Dynamically labelled 'soft' keys can also be used, but this raises questions of consistent positioning, frequency, etc.

1.4 Integrating Excursion Tasks

The previous section examined ways in which a user interface might support a diverse set of application services while presenting as consistant a perspective as possible to the user. One of the services which will be inevitably required is a route for 'excursion tasks'--an exploration of system facilities to determine an appropriate operation sequence for a desired function.

There is a distinction between help at this level and help with the semantics or syntax of particular commands . In the semantic case, there should be no new concepts to be manipulated outside the conceptual model for the system implementation of the task domain. In the syntax case, effective use of examples and natural language parallels can likewise keep the conversation to applications- related elements.

But the excursion task by its very nature involves manipulating system concepts in an effort to navigate from one application state to a new target state. The organization of system functions becomes the key element to be explored. User action in response to a message which reports inability to carry out a command or an unexpected result (formerly known as error messages) can also trigger excursions. Sometimes the message results in specific help requests, but other times the assistance may have to address "how do I get there from here", where there and here are

only vaguely understood.

As discussed in section 1 and 2 above, it appears that no universal conceptual model will develop to encompass all likely user tasks. We could therefore envision a separate set of interface tools which provide orientation within the system. Some suggested tools using artificial intelligence are reviewed in a separate report.[29]

We outline here two experimental schemes which attempt to provide excursion tasks as a primary service. Other applications are integrated into the orientation framework, rather than the usual practice of adding assistance to an interaction style designed for other purposes.

ZOG

ZOG is an experimental system developed at Carnegie Mellon University,[30] and based on ideas originating in an earlier application system, PROMIS, at the University of Vermont Medical School.[31] The system is intended to provide a large set of screen displays * accessed by menu selection. A new screen appears almost instantly after user selection. Selection can invoke actions as well as a new screen, and users can extend the display set.

---

* Referred to as "frames" in ZOG. This suggests "frames" in the artificial intelligence sence, which is too strong a word for the ZOG contents.

Rather than a hierarchy, the screen displays are organized as a (potentially vast) network. For any available application, termed a subjob, an explanation network can be constructed to allow users to navigate through the application.

Navigating in ZOG: On each screen there is a menu of selections which will generate new displays In addition, at any time the user may examine

-the trail of previous display contents

-a list of screens with forward links (i.e. selections) to the current screen

-a set of screens specially marked along the way.

The user can also

-clear the previously-held trail

-estaolish additional marks

-jump to an arbitrary screen (each has a reference)

-search for a character string in the screen network (or particular parts of screens)

Builders of excursion assistance can use the ZOG tools to allow users to orient themselves and explore a network of facilities.

Special hardware has been adapted to rapidly generate new displays, and software is available to design and organize storage of new screens.

However experimentation with ZOG has shown that users can readily become disoriented. The problem seems to be the very local span of information structure: the display panels need to have a map of the vicinity as well as a 'zoom-out' feature which provides an overview.

XS-1

XS-1 is an experimental system developed at the Swiss Federal Institute of Technology to run on a personal workstation.[32] The research goal was to design a single consistent framework with dialogue support at all levels, with emphasis on excursion tasks. XS-1 provides better orientation than ZOG by using additional windows on the screen to 'zoom-out' of the current activity and place it in context. There is a set of commands which can be applied at any time to these dialogue control windows, but doing so does not disrupt the current activity.

There are three basic concepts in te XS-1 framework:

-a site, the current data objects accessible

-a mode, the current valid command set

-a trail, the history of previous interactions viewed as sites, modes and commands.

These are meant to answer questions like

-where am I?

-what can I do?

-how did I get here and where can I go?

The trails are presented as sequences, while the site and mode spaces are viewed as trees. The user can explore any of these by scrolling the appropriate display. He can also move to new data, a new command mode, or a new command by jumping from the current active location to an arbitrary new location, or by moving within the tree or sequence: up, down, (and right or left in trees). This also allows a particular trail to be rerun or even undone.

The current XS-1 assumes that the data and operation spaces can be structured as trees. An interaction kernal provides the interface for application programs to the central dialogue control. The syntax of commands is defined to the interaction kernel by syntax tables, and the file structure and editor are built to support tress. (However the same concepts could be extended to more general network structures with enhanced movement commands).

Thus the dialogue author can take advantage of existing functions like automatic menus for commands and prompting for parameter completion. The dialogue author has to structure the data and mode spaces to fit the XS-1 conventions of many small objects linked together. The XS-1 developers see this constraint as an advantage in preventing poorly-designed interfaces.

Some additional sophistication could be added to allow the spaces to each be structured differently, and for the dialogue author to provide additional levels of structure allowing the user to obtain an overview of mode clusters. Diverse conceptual models will still require varying exploration mechanisms. For instance, a user of a 3 dimensional spreadsheet, might want in the site window to move between sheets, and in the data window to scroll across the given sheet. There might need to be another window to show the columns which are defined on a currently active column. Even forms definition and use in general may require different movement facilities.

The extensions of the previous paragraph are meant to show that the XS-1 excursion facilities need not be viewed as universal. But XS-1 does provide a suggestive mechanism for displaying the organization of large command sets and structured data. While we may not be able to integrate data and command structure into a single uniform framework, we can achieve integration on a different level: access and control facilities which allow us to display and explore the interface object can be unified.

2. Design Principles for User and Interface Evolution

2.1 Multifunctional Office Systems

The last chapter suggested that some aspects of office communications systems - those at the conceptual model level - are not likely to be standardized across different kinds of office work. We are still learning how to think about the properties of office work as they relate to computing systems. Some aspects of dialogue style, language syntax and excursion tasks can be consistent, but there does not appear to be a magic model which will make all the interface complexity dissolve. We are used to switching rapidly between different contexts, like record processing and text processing. In future multi-functional office systems, we will have to tell the machine about the new context and adjust to a new set of manipulation operations.

This process will never be wholly natural. Suppose we say that we can make communication with the machine at best "90% natural", without attempting to define that term further.* If two kinds of application services are each 90% natural, but I have to remember their features as I switch between them, then the overall system effect may be about 80%. That is, the probability of the communication not seeming artificially constrained may well be the product of the individual components. With more types of functions, the adequacy of the interface will drop further.

---

* except to say that we may derive some such number based on communication between people from different cultures.

We may raise our vague number of 90% to 95% by using artificial intelligence techniques or other advances, but by then we will likely be attempting to integrate even more functionality. The history of the computing industry suggests that our reach has almost always exceeded our grasp.

The design challenge is to build interfaces which treat information in diverse ways but for which the overall complexity does not rise quickly with the number of different views. If we are not able to address this problem, to design so that the overall 'naturalness' is not much worse than any individual application, the consequences may include

- rejection, partial use, or even abuse of the office communication system
- resigned acquiescence, where users adjust to the constraints and suffer the erosion of their creative ability to do things in innovative ways.

Of course office procedures often seem now to be artificially constrained, so one may argue that office communication systems will replace one kind of bureaucracy with another, more efficient one. This would ignore the potential of information technology to expand our creativity. It also ignores the fact that office technology is being designed to integrate personal task management into the office system. We are challenged both by diversity of applications and by differing scope of control.

Me-Us-Them - Developers of organization information systems sometimes categorize their products as shown in Figure 2.1.

The operational products represent the corporate infrastructure by which data is recorded and managed. This includes transaction processing, standard exception and control reporting and other data management functions.

The informational products represent information access and communication on an unstructured, ad hoc basis. Various query and retrieval languages are the common products supporting these functions.

The decisional products represent personal toolkits applied by individuals to the information they have obtained. Decision-maker support falls into this category; in a broader sense so does computer-aided design.

The three categories are supported by different development life cycles. The operational systems, because of their wide impact, are developed to meet firm requirements. The informational systems are built in gradual but distinct stages, or are built by users with generic support tools. The decisional systems are often too complex to be built effectively by users, but are built in an evolutionary fashion in response to user experience with working releases.

Operational	- transaction processing (order entry, etc.)
	- administrative operations (payroll, personnel)
	- scheduled reports
	- data dictionary
Informational	- data base query
	- ad hoc reports: trend analysis, exception reporting, etc.
Decisional	- decision maker support systems
	- computer aided design
	- other professional task support

Information System Product Categories

Figure 2.1

This view of application or service scope is adapted from Art Benjamin,[1] who sometimes refers to the levels as "me-us-them". That corresponds roughly to the degree of control to be expected by individuals in organizing the facility and the information.

The history of "management information systems" failures should remind us of the danger of locating control and development within the wrong scope. Any attempt to build centralized systems for supporting individualized activities faces major difficulties. As the growth of personal computers in large corporations attests, local tool acquisition must be considered a factor in office system design.

The design principles to be drawn from these considerations are

- design the scope of an application service and its development with a thoughtful mix of central, local and personal components. For systems designers within organizations, this means matching the organizational culture with the right set of products. For product designers, this means an explicit awareness of the scope of control and development in the intended market.
- expect the addition of personal facilities which will need access to information and which will benefit from integration of dialogue styles, with central facilities. This will mean product design which allows customization at the workstation level, so that a heavy user of a personal tool can adapt the central facilities to a personal style (and vice versa).

2.2 User Evolution

We have previously considered the effects of user differences in degree of exposure and kind of exposure to an office communication system.[2] From a design perspective, we consider here only the exposure differences over which a designer has direct control: the evolution of user behaviour caused by increased exposure to one system in one task.

There are several different perspectives from which one can examine the evolution of an individual's usage over time:

Knowledge view - this view considers the increased knowledge users acquire through more use. Typical growth stages might be[3]

- using packaged facilities with no prerequisite knowledge
- learning the basics with continual support
- independent knowledge of standard features
- probing more complex features
- evaluative and comparative knowledge (seeing the interface as it could be)

If our interface is truly successful, users will reach this last stage and request extensions and new features, as well as constructing some themselves.

Language view - it is possible to view user growth as measured by linguistic units employed in user input. This corresponds to the way people learn natural language. Possible growth stages might be[4]

In the office environment, the infrastructure consists of elements like forms, messages and procedures. The information access involves file systems, with file organization having a localized, unstructured flavour. The personal tools include calendars and project management support.

Word processing has sometimes appeared at the infrastructure level, where an organization creates a word processing pool as a corporate service. It is now more frequently treated as a generic function but with local organization. Workstations like the Xerox Star make some document preparation a strictly personal task, while leaving more general functions like text entry in a locally-shared word processor facility.

The categories by which we think of office system facilities thus depend less on the perceived type of function and more on the scope of control and scope of data management. Figure 2.2 shows a view of office systems which parallels the information systems view but revises it (corrects it?). Some facilities like electronic communication are most naturally central in scope, including access to external networks. These standardized systems will likely be built with traditional or semi-traditional (prototypes, etc.) life cycle development. Filing and some procedures will often be local to a given working group, and will be developed locally from generic functions. Many individual tools will be very personal and built or acquired in many ways.

Central	Central data base
	Forms definition and control
	Messaging systems
	External network access
	Other shared resources (e.g. mass storage)
Local	File systems
	Professional tools
	Conferencing and project control
	Other shared resources (e.g. printers)
Personal	Private tools and files
	Mailbox
	Personal procedures

Office System Product Categories

Figure 2.2

- word communication: user thinks of one word at a time
- phrase communication: user thinks in groups, e.g. all parameters
- sentence communication: user thinks of complete command lines
- paragraph communication: user thinks in sequences of command lines
- creative communication: user creates new words and phrases as appropriate

If we have highly unnatural words, like complex command abbreviations, communication may even degenerate to one character at a time.

Concerns view - Another perspective sees users advancing through several different major concerns about the user interface. A typical progression would be

- ease of learning
- ease of use
- efficiency of use
- ease of extension and modification

Interaction style - all of the above perspectives culminate in a practical progression of interaction styles. Each stage in the progression requires increased user knowledge or language facility, but offers increased efficiency and natural use. Typical stages of interaction would be

- system displays value, user explicitly accepts or rejects. The concrete implementation could be a question/answer dialogue with yes/no options. This would suit only casual, novice users.
- system displays set of values, user chooses one or more. The implementation would typically be a menu with some form of selection.
- prompted value entry (no value display before entry). The implementation could be a form to be filled in or a question/answer dialogue where the user enters a value.
- free choice entry. The implementation could be a command language, possibly with prompts to indicate a mode, i.e. a grouping of available commands.
- user customized entries. This could include user-selected options for error messages (terse or verbose), defaults, parameterized commands, user profiles, etc.
- user created entries. This could include user created command macros, command abbreviations, and packaged kits.

There are other perspectives which consider changes in acceptance and satisfaction as well as the cognitive changes.[5]

Beside providing appropriate interaction styles for users at various stages of learning, interface design must plan for encouraging growth of experience. There must be a consistent progression of facilities. To encourage user learning, earlier interaction styles like

menus must use the command names which a later style will need; sometimes a menu or question/answer style echoes back a command in command language form to reinforce the learning process.

In the other direction, users who fall back from a command sentence to a word entry should not receive an error message. They should be treated as if word or phrase communication is perfectly natural, subject to the need to reinforce a growth path. This may involve providing a form filling mode, a menu or question/answer.

2.3 System Design Factors

Designers of office products are the major audience for this report. System analysts within a user organization, who must match their organizational needs to a given set of products and plans, face a somewhat different set of problems. The role performed by marketing groups in a product organization -- determining the target market, establishing cost/functionality tradeoffs -- will lie with the analysts in a system design environment. We sketch here three concerns these analysts will want to bear in mind; the product designers will want to consider how these issues affect their designs.

Integration and scope: as discussed in section 2.1, the system analyst will have to plan for an integrated service within which central, local and personal scopes of control can co-exist. This will include choosing carefully an analysis and development strategy suitable to the planned scope.

Organizational design: an organizational authority structure and communication pattern can be either enforced or altered by introduction of an office communication system. System designers will want to understand the implications of their designs for organizational relationships. For example, monitor mechanisms in a new message system or assignment of authority for changes in user privileges are important aspects of organizational politics.[6]

Participative design: System designers who can identify their users have more opportunity than product designers to utilize participative design methods. This represents an effort to improve task level design by major involvement of users. A deliberate effort is made to view the proposed system as both a technical and a social entity. In particular, improved job satisfaction of the users becomes a stated objective.

There are various alternatives for user participation, and case studies to outline their merits and drawbacks.[7]

2.4 Product Design Factors

The last three sections have discussed the environment within which user interfaces of office systems products must function. That environment includes a mix of system development strategies and organizational scopes, and a mix of users in different growth stages of system use.

In this section we examine the evolution of products designed for that environment, during the product design cycle. We consider first a design cycle for perfect designers with complete understanding of their target and complete control over the process! This gives us a framework for the design sequence, within which we can then consider design reality with its iteration, backing-up and compromise.

Design framework

The design framework attempts to make explicit the decisions being made and to order them in priority. Since low priority decisions are deliberately delayed as long as possible in the design process, this framework is similar in flavour to what is usually termed top-down design for functional or computational program systems. Top-down design derives its name from levels of organization for program systems, in which the top level outlines overall module structure and details are delayed to lower levels.

In user interface design, the levels correspond to organization as perceived or experienced by the user. (The actual implementation of the interface may thus be quite dissimilar in structure to this organization.) The design process attempts to model the user task at the top level of modularity, and delay action details for lower levels. Since the interaction is a series of communication acts, the framework assumes that semantic structures are more fundamental than syntactic rules and lexical variations.

The design framework sketched here reviews the user interface components previously discussed: task, concepts, dialogue, language and physical actions. They are presented here as outputs from design phases.

Task phase: the user interface designer has to have more than a functional specification to understand the role a proposed system will have in the users' tasks. The additional information about task context begins with the purpose of the task within a wider setting. For example, the wider task is not to send messages but to communicate information about project management, say. We also need to know about

- the degree of predictability and structure
- relative importance to the user
- frequency, interruptions, elapsed time.

This task analysis may be given to us as requirements, but more often it will need to be constructed from interaction with target users, marketing staff and management.

The result of the task design phase can be expressed via specifications describing the system functionality in the user task domain, plus some narrative or scenario scripts[8] filling in the task context. The output from the task phase avoids any vocabulary not already present in the original task. There may also be scenario scripts for current activities to illustrate the existing task context.

Conceptual phase: The conceptual design phase establishes a system model, and provides a mapping between task entities and system entities.

The system model categorizes all the task-related objects with which the user will come in contact: files, documents, records, messages, etc. Potential states and values of these objects are included as part of the model - e.g., perhaps files can be of type record or type message, with different characteristics. There must also be a map for the organization of the system states, so that the system functions are defined by showing possible transformations of the objects. Where the functions themselves are organized -- into modes or access levels --the map must bring that out.

The system model can be expressed via

- a glossary of objects and terms. The glossary is helpful later in restricting system response messages to a planned vocabulary.
- diagrams showing organization of objects and mapping "how to get there from here".
- enhanced scenario scripts. The scripts from the task phase are expanded to describe the system concepts being manipulated during the interaction.

The conceptual design phase normally produces a set of system models, corresponding to levels of user exposure.[9] The first model may rely heavily on analogies from the existing task environment. Subsequent models will represent planned growth in user knowledge. Certain models may be selected as primary, designating the largest usage classes.

Dialogue phase: at this point we have defined the information content of user-system interaction, but said nothing about the method of communication. In the dialogue design phase we define the interaction style or abstract means of communication. Later we can specify the concrete contents of interaction and the necessary physical actions.

In designing the interaction style we have to choose

- a sequence of dialogue types (menus, command language, etc.) which promote development of user knowledge and efficiency

- the semantics of user operations which perform the conceptual-level state changes: "what operations are valid for each element, what information is needed for the manipulation of each element, what the results of the operations are, and what errors may occur"[10]
- system responses, including meaning of various forms for system output.

At this design phase we have to consider not only the original tasks of the user, but also the new tasks of learning about and navigating in the system. Adding features for these activities may expand the conceptual model, when additional help modes or assistance components are to be distinguished. The set of conceptual operations expands to include these facilities.

The output from this design phase includes

- a specification of the dialogue user operations and system responses
- expanded scenario scripts showing usage sequences.

The structure of the dialogue is often specified in diagram form, as a high level tree of user choices or a more general interaction diagram.[11] The specification must also explicitly relate the contents of the system display to the conceptual states of the system objects -- is what you see what you get?

Language phase: designing the interaction language involves defining the concrete terms to be used in communication and their organization. This includes the actual words to be used for commands, the grammar rules of a command language, the order of terms on a menu and the meaning of spatial or typographic cues on displays.

The output from this phase includes

- a language grammar, including error correction and reporting algorithms
- a dictionary of command words (with the semantics described in the dialogue phase), menus and system messages
- expanded scenarios

Physical action phase: in the language design phase, we might have specified user selection from a menu. For the physical action design phase, we decide on the mechanism of selection: light pen, cursor movement in various ways, typing, etc. This defines at the device level what is actually happening. We would also define elements like function keys and their positions, and specific typefonts, colours or sizes for display outputs.

Output from this phase includes

- keyboard layout and detailed screen layout charts
- physical parameters for system display or user inputs
- final details of scenarios.

Format of phase outputs: we have not discussed at length the actual format for grammar rules, dictionaries, etc. Many will take advantage of automated aids and be used as tools during implementation.[12] The scenarios can, for the language level and the physical level, be implemented in appropriate prototype tools, as discussed in chapter three.

In certain application domains like data entry, the task can be partially described with standard checklists to automate part of the task design phase.[13]

Formats used for the conceptual model are much more diverse and informal in nature. Some work in knowledge representation may be helpful in making these notions more precise.[14]

Another suggestive set of terms for the design framework[15] describes these design phases as

- intention
- connotation
- denotation
- rules
- constituents

Design reality

In the interests of simplicity, the framework outlined above deliberately ignored several aspects of realistic design. In practice the sequential, top-down process consists of several iterations as knowledge is added from other sources. That is, the framework above accounts only for input from a single designer. While overall interface design usually maintains its integrity by allocating design responsibility to one individual, the interface is improved by contributions from other professional developers, from management, marketing or other organizational perspectives, and from users.

Design teams: the relation of user interface design to other design elements is still being explored. Originally, user interface design (if it could be called that) was a last sequential stage once the functional implementation was complete. More recently, some development methods recommend the exact opposite sequence - user interface design precedes all functional development.[16]

In new products, where the tradeoffs of various elements are initially unclear, it appears inevitable that functional and interaction components will be designed and developed in parallel (along with any special hardware). This can be done either by accident -- returning to earlier stages in one design cycle or the other as errors and omissions appear in the sequenced development -- or by deliberate intent to have both kinds of design proceed concurrently.

The communication and control paths between the two design streams are still unclear, although some effort is now going into definition of concurrent design methods.[17]

Design politics: "The designer must face the fact that design is as much a political as a conceptual process. Unfortunately politics have been equated with evil... but politics are the process of getting commitment, or building support, or creating momentum for change."[18]

A designer's view of the user interface will thus have to be defended to, corrected by and modified for the following groups within the designer's organization:

- a development group, whose perspective will stress ease of implementation and maintenance
- a 'product integrity' group, whose perspective will stress consistency across product lines.
- a marketing group, which may be seen as representing the users' perspective
- other management levels, concerned with cost-effective development and an opportunity window (the time within which the product must reach the market in order to be most successful).

Design iterations: the various input sources and the testing strategies of the next chapter will inevitably produce loops back to modify earlier design decisions. The lure of a quick fix to a perceived problem, must

be resisted; concern for maintaining design consistency requires that change at any level causes us to rethink the interface level above it, as well as letting the changes affect lower levels. This feedback requires that an explicit representation of the interface exists at the higher levels.

Experienced designers evolve a set of informal design techniques and rules for understanding the effects of tradeoffs and change across levels. The choice of object/verb versus verb/object syntax in a command language, for example, is determined in large part by the conceptual view and dialogue style of the interface. The object/verb organization does make default entries slightly more efficient, but we wouldn't want to sacrifice much conceptual integrity for efficiency.

User-oriented design

The participation of users in the system design process was mentioned briefly in section 2.3. For product design, it may not be as easy to identify representative users and secure their co-operation; for reasons of practicality or organizational protocol, a marketing representative may represent the users during product design.

If the feedback from users can be obtained early in the design process, the following points need to be remembered to take advantage of the feedback[19].

- users are not inventors, and will not usually be able to suggest new methods to replace those deemed unsatisfactory
- users cannot usually relate to tradeoffs except in very concrete terms
- users should take a reactive perspective, expressing likes and dislikes, and reasons.

For later design stages, the user interface testing techniques of the next chapter can be applied.

Note: we have not discussed the design of user documentation.

3. User Interface Testing

User interfaces need to be tested before, during and after implementation. These test stages involve lab exercises, simulated use, demonstration prototypes and field trial versions. Testing guidelines and techniques increase the benefit from each stage. When users experience difficulty, we need to determine where the trouble occurs, how the trouble occurs and why the trouble occurs.

3.1 Lab Exercises

There is a growing body of behavioural science research addressing the psychology of user interfaces. These laboratory exercises can also be used early in systems development.

A typical lab exercise contains the following steps:

- isolation of a single issue to be studied
- design of a quantifiable test exercise related to the issue
- administration of the test exercise to sample groups
- analysis, formal or informal, of the test results
- interpretation of test results relative to the original issues

A classic lab study which illustrates the steps was performed by S.K. Card and others to evaluate a mouse cursor control against other mechanisms.[1] The original issue was which device (mouse, joystick or key controls) was more effective as a text selection device for text displayed on a terminal. The test exercise involved timing users as

they positioned the terminal cursor onto displayed targets. The results showed that mouse users took less time than users of other devices. Formally, the result was statistically significant; informally, users saved fractions of a second, on average, for each target selection. One plausible interpretation of the results is that a mouse interface will be more effective for text selection.

The central strength and central weakness of lab exercises are one and the same: the original issue is reduced to a one-dimensional problem. This reduction lets us isolate and study a single behaviour; it also ignores other interesting behaviours. For example, the mouse study did not consider mouse learning times, user anxieties, the problem of hand movement from keyboard to mouse, etc. Even in restricted settings, lab behaviour and field behaviour may not match.

Surveys of reported lab exercises and a view of their implications are just now beginning to appear.[2] While these are not specific to any one system, there are instances in developing particular systems when suitable issues arise for informal lab exercises. Here are two representative examples:

-- To study user compatibility of a set of proposed operations, sample users are given a list of operation definitions. Then a series of task situations is presented and the users are asked to match these cases to operations (or sequences of operations).

Where the user responses differ from what the designers were expecting, we will suspect that either our descriptions are inadequate, the breakdown of operations is inappropriate, or the whole task model needs rethinking. If new descriptions do not produce any change, we will want to try new decompositions of the system functions into operations.

It is important that command names not be introduced before we have investigated the clarity of the operation set. Of course well-chosen names will aid understanding of descriptions; however, the various ways names interact will make it difficult to distinguish poor name choices from poor operation definition.[3] (We may want to ask the user subjects to name the operations as an indication of description clarity, but we shouldn't expect users to be skilled at choosing good names for others to use.)[4] On the other hand, a lab exercise could help determine the boundaries of user vocabulary at the conceptual level.[5]

Similarly the operation descriptions should be at the same level of generality. Otherwise, on first exposure more people may tend to use generic operations,[6] regardless of the clarity of function decomposition.

-- The previous example examined a conceptual-level question where testing during use was not suitable. Usage, real or simulated, would have required the dialogue, language and physical levels to be speci-

fied, and we would lose the conceptual-level question. Lab exercises can also be useful at the opposite end of the design cycle, to study execution issues like the mouse vs. key control study outlined above.

A major difficulty facing execution level studies is the effect of learning. We know that efficiency of use is not a concern in the first stage of user exposure. Lab exercises typically employ novices as subjects. This minimizes the interference effect of previous learning but makes it unclear how the lab result should be generalized to actual use. With physical actions like mouse control this may not be important, but studies of the efficiency of different syntax rules will be more difficult to interpret.

The organization of output displays has been frequently examined using subjects without prior system exposure. One particularly interesting example compared a fixed format for display of messages with a format which users could adjust.[7] This was done early in the design process to help indicate whether personalized display formats were worth adding as a feature.

Two groups were exposed to a timed sequence of messages. One group was encouraged to design their own order for the different message components (eighteen in all); the other group was given a fixed format chosen by the designer. Both groups were allowed to take notes. After the messages were completed, a comprehension test was administered (without notes).

Subjects allowed to personalize their message formats made a similar number of errors on comprehension tests as did subjects shown a fixed format. However, the personalized format group took much fewer notes on messages during the exercise, possibly suggesting that their message formats may indeed have been more effective in conveying information. There was also a good deal of agreement on the personalized formats, indicating that the designers' original choice of display format probably needed improvement.

Note that users were always tested by objective measures rather than by their own preferences. Given equal performance, we may want to let user opinion have considerable weight. However, it is difficult for users to always estimate or evaluate the impacts of system variables on their effectiveness.[8]

Outside of a lab exercise format with strict controls, statistical formalism is frequently not productive. Individual differences in performance amongst experienced system users is often as high as 30 to 1. That variability and the inability to limit external factors make valid experimental design difficult in the prototype testing situations to be described next. Only obvious performance differences between groups will be significant, and then only as we analyze their causes. We will rely in prototypes more on differences between observed and expected user behaviour.

3.2 Testing Prototypes

Software developers recognize the value of constructing functional prototypes.[9] Early in the system life cycle, users can provide feedback and view progress. Prototypes can correct incomplete or faulty functional requirements specifications; less tangible, but possibly as important, they develop a sense of system ownership for users. Management within software development also responds positively to viewing a prototype - including funding decisions in a product environment.

Using prototypes to test user interface characteristics is sometimes more difficult. Users exposed to early system mockups comment quickly on missing functions or faulty logic. But awkward features and inconsistencies, when noticed at all, may be attributed to lack of training or personal inadequacy. By the time an initial release is made available, the complexity of interaction may encourage designers to look for quick fixes to problems rather than their underlying causes. Patching up on-line assistance messages often replaces careful diagnosis of usability flaws.

In this section, we consider the different stages of interface testing with various prototypes, including the kinds of user interface problems which may be exposed at each stage. Section 3.3 contains guidelines for running test sessions which address interface quality. Table 3.1 summarizes the prototype stages.

Table 3.1

Testing Summary

Lab exercise	- single issue experiment
	- implications typically for novice use
Scenario	- increased knowledge of users and task
	- very useful in requirements specifications
Demonstration	- observations of novice use
	- shows impact of all interface levels
	- shows ease of learning
Field Trial	- actual usage patterns revealed
	- observation of intermediate and practiced use
	- shows ease of use and efficiency of use

Scenarios

A scenario is a simulation of a prototype system capability. Unlike a lab exercise, users should see themselves in an actual task situation. Unlike a demonstration prototype, a scenario prototype responds to only a restricted subset of the user options available. The scenario is frequently implemented in a quite different way than the final system.

Scenario scripts were mentioned in chapter two as a design tool. These include task descriptions constructed by analysts or users, and system interaction descriptions constructed by designers or independent reviewers. Scenario prototypes differ from scenario scripts in intention. Prototypes are meant to allow users to experience, early in the design cycle, what it will be like to use the system being developed. They should be an animation of the current design - to make it come alive.

Scenarios impose two artificial aspects on system usage

- functionality is simulated, so that response times are not necessarily realistic
- the problem or example is chosen by the designer, not the users.

Within these constraints, scenarios can help in the following testing areas:

- evaluating a narrow focus question (much like a lab exercise) in relation to a specific current design

- examining the correctness of control flow and information display
- observing problem solving and information handling by users

The degree of realism required by the scenario depends on what we want to test. A paper simulation can capture enough of the live experience to address single issues, like a lab exercise. A software prototype provides better animation, so that we can get information on a wider range of questions. We can work from a fixed script and obtain user opinion and reaction for control flow and information display. To observe problem solving, we prepare a scenario which follows enough decision paths to allow users to make choices. We can record the choices and times to yield objective data, as well as noting subjective reactions.

Within data processing literature, there are numerous examples of scenario usage to examine control flow and correctness of information by user opinion.[10] The following two examples illustrate ways to address single design issues of problem-solving behaviour.

-- A quick and simple test of a videotex interface was performed to consider the effectiveness of the initial instruction set.[11] A drawing of a sample keypad and the set of proposed instructions was given to a large group (students in a classroom). After reading the instructions, the subjects were to write, in a space provided, the label on the key which they would press first in response to the instructions. Over half

the respondents had the wrong key. Several design iterations followed, each using the same convenient test method, until the instructions were deemed effective.

-- A computerized Flight Design System to aid in planning space shuttle flights was simulated on paper in increasing levels of detail[12] (requiring use of more specific commands as the design proceeded). In each simulation, the goal was to determine the extent to which the task structure required by the user was properly supported. All computer responses were calculated or estimated by the designers on the spot.

Demonstrations

A demonstration prototype provides realistic processing of user queries or data within some restricted limits. The system responses are not built in; the users can therefore bring their own problem data provided it fits the prototype's limited functionality. The users must receive training so that they employ the actual commands of the current design.

In demonstrations we typically need to probe the reasons for user actions. In this way we try to determine which of many contributing factors is helping to shape the observed behaviour - training materials, interference from other systems, and the various levels of the system interface. For scenario prototypes, we have to define in advance the

kind of test data we will use, given what kinds of observations are of most interest. In demonstrations we may start with sample cases, but asking users to bring along current work helps to detect unusual cases.

Example demonstration payoffs include:

- detection of menu problems in an IBM System/34 user interface.[13] A prototype version was programmed and run with sample tasks and participants for various user groups. Usage problems fell into two clear classes: cases where more than one menu item seemed a likely choice, and cases where no menu item seemed likely and a shotgun approach resulted. With this information, the designers were able to generate a new menu set. A second set of demonstration runs confirmed that the major problems had been addressed.

- improvement to help messages for a business graphics package.[14] Faced with an already existing system in which only the assistance messages fell within their mandate, the experimenters produced four redesigns of the help facility in response to user error patterns.

These examples illustrate a more focussed demonstration than is usually possible. Unlike lab exercises or scenarios, demonstrations sometimes must serve as 'fishing expeditions' where we are not sure what user behaviour will be of interest until it appears. The most we can do then is note the user responses we expect on the sample data. Tech-

niques for recording the actual behaviour are treated in section 3.3; some suggestions for categorizing the causes of problems are included in section 3.4.

Field trial or version '0' usage

Demonstration prototypes are usually exercised with close interaction and recording by developers. That fact alone makes usage artificial to some degree. There are also limitations on duration of use and the scope of actual test cases.

Testing, therefore, has to include use in a real task environment by real users. The users should be typical, but sometimes for political reasons the test sites are selected in unusually supportive surroundings. User management must be prepared for the additional burden of developer observation and some product flaws yet to surface. Field trials within the system development department are not as instructive as an external trial or 'version 0' of a product.

A version 0 or field trial prototype is a working release of a system which is intended to receive use under conditions approaching the production environment. While it is specifically designed as a test release, it is usually expected that the final product will build on version 0 by enhancing the implementation of functions, adding requested alterations, and generating additional documentation.

The field trial is the first objective source of information on intermediate and practiced users. Since the volume of data generated by the trial will be large, we want to look for patterns of use rather than individual incidents as in a demonstration.

Table 3.2 lists the various techniques to be employed and their goals. Techniques are described further in section 3.3.

Many field trial techniques have been used to observe production release use. Such observations record surprise about commands which were used improperly or never used at all. There are also numerous accounts of partial use, in which users stick to a small subset of commands and ignore other system features which could be easier for them to use.

Descriptions of planned field trials are available for products like a decision support system[15] and an electronic mail system.[16] For each, a variety of observations, measurements and interviews was used to illuminate patterns of use. In the first case, in which the phrase 'version 0' is used, the major payoff was a better fit with the task structure. In the mail system trial, an attempt was also made to measure productivity gains.

Table 3.2

Field trial data collection

Question	Data Collection Technique
What are users doing? (incl. documentation usage, asking other users)	Software logs Observation and recording
Why are they doing it?	Debriefing, talk-through sessions
What do users think they're doing? (conceptual model) What do users think about what they're doing (job satisfaction)	Interviews, questionnaires, talk-through sessions. Possibly scenarios (alternate scenarios to evaluate conceptual model)
What would users like to do (including task model)	Interviews, possibly scenarios

3.3 Interface Testing Techniques

During the stages of interface testing summarized in Table 3.1, the need arises to observe and record user behaviour. This includes both visible activities and the cognitive activities behind them. The techniques used are summarized in Table 3.2.

Interviews and questionnaires are well-known instruments for data collection. We will describe in more detail observation records, software logs, and talk-through sessions.

Observation records

User activities communicated to the system can be recorded by the system in a software log. Other simultaneous activities must be recorded separately to interpret the meaning and timing of the software log. Such activities include:

- reflection
- consulting other users
- consulting printed documentation or written instructions
- composing personal notes about the system
- task activities not directly related to the system, such as interaction with clients.

An observer during a demonstration or field trial can note any of the above which appear to be of interest. But it is more productive to have an observation routine in addition. The observer can still be free

to record particular items which fall outside the routine. Some of the routine may repeat measures taken before system use to estimate system impact.

A variety of observation methods are outlined in Table 3.3. In each case it has to be clear to the users that it is the system being evaluated, and not them. To reduce bias, observers should not be the designers. The evaluators or observers should make this apparent (that they have no personal interest in the current design), so that the users will not suppress comments out of politeness.

Task records, communication diaries, etc.:[17] These tools were mentioned in chapter three as an early part of the design cycle. Their repetition here may serve as a measure of system impact. If used in that way, there should be a control group who do not use the system but who complete the measurements in the same way. This may reveal the effects of a second record, independent of the system.

Activity sampling:[18] This technique may also have been used early in design. At periodic intervals, the user's activity at that instant is recorded. A typical interval time is 15 seconds, which provides a detailed record of actions but permits time for encoding. Observers are given training in use of special coding forms.

Table 3.3

Observation recording methods

Task record, communication diary	Compare with same record before system use to assess impacts
Activity sampling	Compare with same record before system use to assess impacts
Video recording	Synchronize system interactions with other activities
Audio recording	Synchronize system interactions with talk-through sessions

Video recording: A videotape or film of a user session can be electronically synchronized with a software log, and can be timed separately by frame or with a visible clock. This permits detailed analysis of the causes for system problems, need for documentation, etc. It is especially useful as an animation of user difficulties for the designers.

Audio recording: In sessions where users are encouraged to talk through what they are doing, and to work co-operatively with another person, an audio tape recording captures the conversation. It is again a vivid presentation for the designers. Stereo tapes have been used in which one track carries user keystrokes and one conversation.

Talk-throughs

A 'talk-through' or 'thinking aloud' session involves users who make verbal comments as they work in an interactive session. The goal is to illuminate the reasons behind various actions, for later analysis. Recording of such remarks was discussed above. We will note here some procedural considerations; a fuller discussion appears in a report by Clayton Lewis.[19]

A natural way to encourage comments about the interactive process is to have users work in pairs, one at the terminal and the other adjacent, acting as a helper.[20] This is most useful for observation of novice users who can be expected to seek out advice.

A less natural but equally effective technique is the presence of a 'silent partner', to whom questions and comments are addressed. The observer must be perceived as neutral, i.e. having no stake in the system. It is usually important for the observer to initially prompt the users to keep talking. The observer's comments must not influence the user's course of action, so they are limited to questions like "What thoughts are coming to you now?" (not "Why did you do that?"). If the observer wants to ask detailed questions, the session can be replayed later as a debriefing with the user.

Thus, the user may ask questions of the observer but will not get answers. In extreme cases - the user is about to quit - perhaps that rule could be waived in the interest of keeping the session going. At times users are informed that they will be given the answers to only a fixed number of questions (say, 3) during their task. Users should be encouraged to express frustrations and emotions as well as conceptual problems!

Software logs

The observation records listed above provide anecdotal and context material which supplements the record of user-system interactions. There is no substitute for data showing actual usage of the system, especially if statistics on large-scale use are desired. With minimal software overhead, such data can be recorded by the system itself.

A software log records user entries and system responses. The records may be coded if the monitor is an integral component of the system. Otherwise the actual dialogue is recorded and later parsed to recover language constructs from the characters transmitted.

The log will furnish frequency statistics after a little lexical analysis. The occurrences of various commands, error messages and help messages are thus readily available.

Commands: Designers have sometimes been surprised to learn about the relative frequency of command use (or lack of use).[21] To analyze command usage further, pattern recognition techniques can be employed, with some effort, to distinguish different activities based on length of an interactive session and command pattern.[22] Formal models can then be built of the different types of system use.

Similarly, keyword parameter frequency is easy to obtain. It may also be of interest to analyze user-detected errors by recording lines deleted before full entry, erasures within a line, etc.

Errors: The frequency of error messages, expressed as a raw statistic, will not usually yield all the information we need to improve the interface. A sophisticated software log will also compute the frequencies of various pre-error patterns (eg. the preceding 3 commands) and post-error patterns. Post-error patterns are particularly interesting when another

error message follows, indicating the original error diagnosis or suggested options were unsuccessful. We can also detect probable typing mistakes by comparing the command strings before and after error messages, so that examining error patterns distinguishes keying problems from more serious difficulties.

Tracking use of an undo operation, which reverses the effect of a previous command, can likewise aid understanding user behaviour. Help messages can be analyzed in similar fashion to error messages.

Privacy concerns: A software log naturally raises the issue of user privacy. The simpler frequency statistics can be kept without reference to a specific user. Analyzing patterns requires recording of at least a few commands in sequence, but there again need not be associated with a particular person. All non-keyword entries like filenames should either not be logged or be coded to disguise the user.

To analyze whole sessions, user learning, or variability between people, the usage record must be tied to a subject. If the number of users is small or some users have distinctive patterns then their identity may be recovered from their pattern of use. Users should be told the purpose of the software log and how to turn it on if they wish to contribute. Note that this is not the same as being given the option to turn off the monitor. Moreover the decision to participate

must be revocable - once a user chooses not to participate, due to the nature of the data desired, no software log will be kept from then on (i.e., the log cannot be disabled for one session only).

3.4 Interface Malfunction

The information obtained by interface testing is, of course, pointless unless we are able to apply it for improved products. We need to consider the data on user-system interaction to determine what situations need to be classified as problems and analyze them appropriately.

The term malfunction is deliberately chosen to label undesirable performance without assessing blame (unlike the labels (user) error, (software) bug or flaw). It is a mismatch or poor fit between the task requirements and the human resources applied.

We will consider

- where the problem appears (the external appearance of the malfunction)
- how the problem emerged (the interface level of the malfunction)
- why the problem occurred (the mechanism of malfunction)

Some of our terminology here is adopted from another interface area in which malfunctions can be critical -- control displays for nuclear

reactors.[21] Extensive testing is performed on operators using simulators, since critical incidents are rare but have enormous consequences.* Each instance of malfunction is analyzed for the benefit of the operator and the interface designers. The symptoms and interface levels of malfunctions are different in an office system environment, but the underlying mechanism of malfunctions appear to be task-independent.

External appearance of malfunctions: The most obvious symptom of malfunction is detection by the computer system of an incorrect user request. This could be triggered by

- omission of a required item within an entry
- a syntactically invalid entry (grammar rules broken)
- a semantically inappropriate entry, i.e. syntactically legal but not correct

For example, if user wants to send an electronic mail message, MAIL1, and the proper command is

SEND MAIL1 TO SRON ,

then MAIL1 TO SRON is an omission,

SEND SRON MAIL1 is a syntax problem,

and SEND MAIL2 TO SRON if MAIL2 does not exist, is semantically incorrect.

---

* Reading the accounts of errors occurring on nuclear plant simulators is not recommended for the faint of heart.

Of course, a more worrisome case would occur if MAIL2 does exist and gets sent instead of MAIL1. When the system doesn't detect the problem, the symptoms will be

- cancellation or reversal (undo) of the operation
- repeating the operation with changes.

The former case can be located in a software log, but the latter cases must usually be noted on a one by one basis.

A final type of external malfunction appearance concerns timings:

- unexpected elapsed time between entries (user takes longer than expected to decide next entry
- unexpected length of operation sequence (user takes a seemingly roundabout way to reach a target state).

Instances of long elapsed time between entries will be recorded by software logs, but may have extraneous causes, like interruptions from external sources. Apparently inefficient use is typically not detected by the user.

Malfunction level: The external appearance is the symptom of malfunction. It shows where a problem has occurred. How the problem arose requires further analysis. Typically, we have to examine malfunction patterns, study think-aloud records, and interview users. Figure 3.1 uses the levels of an interface to describe malfunctions.

intentions not fully formed	
task goals not supported	TASK
status or target identification inappropriate	CONCEPTUAL
operation inappropriate	DIALOGUE
language expression inappropriate	LANGUAGE
physical execution inappropriate	PHYSICAL

Malfunction Levels

Figure 3.1

In the previous discussion of interface levels, we noted that the stages may not be distinct, particularly past novice or intermediate exposure. However, before we can improve the interface, we have to attempt this kind of analysis. It is pointless to emphasize language syntax for example, when we really need to alter the semantics of system functions.

The first element of the figure could really occur at any stage in which the user's attention has not been sufficiently focussed on the problem at hand. Reasons for this are discussed later in this section.

At the task level, the malfunction may have arisen when the user attempted a task not supported by the interface. This could typically be an attempt to reach a task state which is not attainable, including information access in an excursion task. The user may have chosen the closest alternative available in hopes of arriving at the desired state.

At the conceptual level, the user may have misinterpreted the current status of the system or attempted to reach an incorrect (but reachable) target system state. Where these states have been correctly identified, an inappropriate operation may have been chosen (at the dialogue level) which will not move from current to target state. This may include interruption or other facilitative operations.

Syntax errors, invalid command names, turn-passing omission (eg. failing to press RETURN key) are examples of language-related malfunctions. Typographic errors and perceptual mistakes like reading a word incorrectly are typical physical level problems.

The connection between external appearance of a malfunction and the action which brought it about is often indirect. Tracing back from the symptoms of a problem to the action can be challenging, frustrating, or both. However, knowledge of the user's intent, a record of the action sequence and an understanding of the interface will always hold the clue to how errors or wrong answeres have arisen (inefficient usage as a malfunction is less straightforward).

Malfunction mechanism: To understand why a malfunction has occurred, we need to determine the user's cognitive actions which determined the observable actions. Not all of these will be accessible, so we will often have to infer the processes by which actions developed.

Figure 3.2 illustrates some of the analysis questions which might be asked to clarify possible interface improvements.

Where normal attention has not been applied, most of the causes will not be under the control of the interface designer. Improving the aesthetic appeal of the display or providing more media of output can alleviate some boredom, but motivation or fatigue problems are job related. They do indicate a mismatch between the resources demanded by the task and the resources applied.

Physical problems are often mixed in with stereotypes. Where a display is too small to read or keys are hard to reach, the problems (and solutions) are physical in nature. Where a person reads or types a word incorrectly, the actions may involve predicting/confirming skills and suffer from interference of similar words. In these cases, we may need to alter the word choices at the language level to avoid mistake.

Stereotyped behaviour can involve more than perception. For example, a user may correctly "read" a status display, but the information may not register. That is, the need for higher level processing

may not be noticed and a common response may be involved incorrectly. To correct this kind of malfunction, we will need to foster user awareness of non-typical situations and more noticeable displays.

Where the need for appropriate knowledge has been recognized, the information may not have been applied. The user may have not recalled the required knowledge, may have recalled it incorrectly, or may never have known the needed facts. In each such case we might need to improve the interface (including supporting training and documentation) so that information is easier to learn, recall and access.

Sometimes false conclusions will be drawn from correct information. An analysis of such situations may lead us to reduce the complexity of mental processing required, to provide more information cues in potentially difficult cases, or to alter training materials to include them.

Partial use of an interface, where users are satisfied with a subset of knowledge (sometimes applied awkwardly), is typical in early use. If it persists into intermediate or practiced use, then the interface is failing to encourage user development. The fault may be in documentation and support materials, or in an overly complex pattern which discourages learners.

These mechanisms of malfunction appear to arise in most cognitive processing tasks. User interface designers have to develop the humility to blame their designs -- and not the users -- when the interface requires more work than they can apply.

4. Sample Elements in Office Conceptual Models

Some aspects of conceptual models for office systems were discussed in section 1.2 above to illustrate the difficulty of designing a single model of diverse, multifunction systems. We compare here some elements of proposed conceptual models (implicit or explicit) to see how an underlying model affects the orientation of a system and what its designers considered as important.

These models differ from analytic models whose purpose is to log and categorize actual office activities, e.g. information control nets[1] or the Kayak EA/AQ[2] family. The analysis models may be used to help design portions of the office system, but they are not intended to form the users' framework during interaction.

Our focus is on two aspects of conceptual models.

- the objects in the system and their organization
- the mechanism for describing information management procedures or automated methods.

In particular, we will not discuss information manipulations like various editing strategies or query language structures.

4.1 Conceptual System Objects

Three basic orientations appear in conceptual models for office systems, yielding three perspectives on the central activity of an office:

- office functions can be perceived as centering around the information stored and manipulated by individuals and groups. From this perspective, office services are viewed as a distributed data base and central concerns include data consistency and access.
- office functions can be perceived as centering on the output produced by office workers. From this perspective, office work focuses on document production and distribution.
- office functions can be perceived as centering on the communication process amongst people and groups. From this perspective, the major system functions are message processes and co-ordination.

We present the conceptual models of information management in three distinct systems: the Xerox Star records processing, which has its origins in document production; IBM's Office-by-Example (OBE) research system, which has its origins in data base systems; and the Office Forms System (OFS) from University of Toronto, an experimental system derived from data base work but deliberately aimed at transcending the limitations of a data base orientation. OFS includes plans for extensive message co-ordination, which we will not dwell on; other systems exploring a messaging perspective are Officetalk-D[3] and the Kayak[4] family. The conceptual objects are summarized in figure 4.1

Figure 4.1

Conceptual Entities

	Definition	Storage	Display and Query
Star	Defining document	Record file	Display document Filter Sort order
OBE	(Table definition)	Tables	Form Report (menu, program)
OFS	Form type	Form instances Forms	Form templates (text, voice, data)

Star: the Star's information management facilities form the records processing extension to the original document production functions.[5] Since it was a released product before records processing was added, particular effort was directed at maintaining consistency with the original document conceptual model (discussed in section 1.2).

Record files are initially defined through a defining document, whose field structure and names are carried into the file definition. Display or query of a file requires a view, consisting of

- a display document, which selects a subset of record fields and possibly combines them with other information
- a filter, which selects a subset of records
- a sort order, which determines the sequence in which records are presented.

Insertion, update and deletion of records follows basically a document editing model. The notion of a blank form used for updating is foreign to the spirit of this model; record insertion is like inserting a line in a document. Similarly, a filter requires a specification of patterns to be matched in a record, following the model of an editor searching for all occurrences of a pattern.

The document paradigm does not easily handle rules or relations amongst fields of a record. The defining document may have rules to fill in fields from other fields, but these are consistency checks and are not maintained as part of the record file.

Office-by-Example (OBE): OBE derives from its relational database origins (child of QBE) a perspective with relational tables at its centre.[6] The tables are defined first. Data objects which can be linked to tables include forms and reports - forms are used for both input and output, reports are output only. Forms are generalizations of relation tables; they are not tied to single tables the way Star's views are tied to a single record file. Reports and forms are linked to tables dynamically by specifying example elements shared by the table and the other data objects.

Where the Star regards record processing as an extension of text processing, OBE views text processing as a degenerate case of a report - one in which no fields are related to tables.[7] In a wider sense, tables

are seen as prototypes for general two-dimensional boxes, which may also contain sequences of commands (programs) or preset forms and relations (menus).

Office Forms System (OFS) : OFS has its origins in data base management systems and has more concern for data types and consistency control.[8] The ubiquitous word form is given a very general meaning here and several variations.

- a form type specifies the data types of fields in a form and optionally some procedures for consistency checks, generating field values from other fields, etc.
- a form instance is an occurrence of an object with a given form type, plus results from associated procedures. For instance, a form type may include a procedure for logging changes to a particular field, so that the form instance contains a history of previous values. Form instances have unique identifiers to track time of creation and user.
- a form is the set of field values (= a form instance without attendant procedures)
- a form template specifies a way of interpreting or displaying a form instance. Text templates can be used as display documents in the Star sense; data templates can be used to treat the data in form instances as a relational table. Templates are more general than OBE forms, since they can also transform data in the

form instance itself. Many different templates can be defined for a form type, providing alternate views of a file of form instances.

OFS thus attempts to generalize beyond its data base roots to allow a variety of interactions between fields in a form, via form type procedures and template mappings. This requires the careful separation of types and instances (technically we could have spoken of template types and template instances), and a departure from OBE's world of pure relations. Consistency issues and maintenance of data base integrity are addressed in a more general way when the procedural elements are tied to the form type.

We have examined three different conceptual models for functions which seem on the surface to be similar. The conceptual model we choose will structure the user's thinking and interactions with the system.

There are other models which depart even further from what we might be used to in traditional office objects, for instance a system modelled on 'migrating forms' which are independent processes moving about a network.[9] If the forms carry the intelligence and the workstations are seen as passive servers, this is nearly a reversal of roles. In previous work in programming languages this kind of metaphoric change in system initiative and control has been disruptive.[10] Choosing the right model for a given group, especially the degree of alteration in traditional models of work, has to be approached modestly and cautiously.

4.2 Office Procedures

The objects and operations described in the last section mechanize certain task actions, like searching through files, and offer substantial integration of various tasks into a common framework. In order to accomplish some measure of task automation, we need a mechanism for recording sequences of decisions and operations - an automated office procedure. To go further and offer task augmentation, i.e., opening new tasks within the office, we would require new capabilities like monitoring of current workstation loads or of delays in processing.

In this section we compare elements of several schemes for office procedure processing. The proposals are in various stages of development ranging from research design to substantial (but incomplete) implementation.

One way of viewing procedures examines whether they express a set of activities associated with office services or a set of activities associated with office control. Office services procedures describe the actions of a single role or position with respect to a given work unit - document, message, etc. They are thus local in scope and may need to be modified by clerical or professional staff performing a service on the work unit. Service procedures could be linked to positions, roles or even particular workstations.

Office control procedures would automate or support co-ordination of numerous tasks or roles operating on a unit of work. The actions of a purchasing agent with respect to a purchase order are part of a service procedure; the co-ordination of actions of the purchasing agent, accounts payable, receiving department, and so on from an office control procedure. These are likely to be under centralized control, with scope covering a large organizational unit. Authority for these procedures will be limited, and change more restricted. There will be a greater need to track progress of such a process, so that the procedure may be associated with a given workstation but it will need to have access throughout the office.

This distinction in perspective is, of course, sometimes blurred: the service performed by some office groups is primarily one of control or monitoring, perhaps as an after-the-fact audit. Another view of the distinction perceives the difference as one of hierarchical structure, in which the control procedure occupies a high-level or more abstract role and the services procedures are the concrete low-level activities.

We choose to view office procedures by looking at their scope of application because it clarifies some major differences between contrasting systems proposals -- in terms of operations, control conditions, specification method, and system processing.

Office services procedures

A primitive form of procedure is provided in the form editing conditions or fill-in rules provided in the system objects of the last section. They were part of the mechanics of filling in a blank form and maintaining integrity during later manipulations.

A service procedure represents typical form processing by an office worker. The operations for checking input forms, manipulating information, and generating output, use facilities such as those outlined in section 4.1. In addition, we must be able to specify decision conditions and their effects.

Conditions: The kinds of conditions affecting processing reflect the conceptual model of system entities.

The OBE system, with its heavy data base orientation, uses modifications to the data base as a key 'trigger' in initiating actions. Certain timings can also be given to start events, including periods like daily or weekly. There are conditions on field contents to pick out specific modified records.

OBE does not have explicit conditions based on arrival of a message of given type from a specified source (although of course it can be handled in a more cumbersome way by using a MAIL relation). OFS does provide for checks on origins of documents, including ability to

list origin points which are not to be selected for processing. Other desirable conditions include elapsed time: if no confirmation message arrives within three days of receipt of a form, then certain action is to begin. We might also want general pattern matching in documents; for example, a mention within a document of a particular product may mean a copy is sent to a certain department.

Specification: How do users express the relationship of operations and conditions?

OBE uses trigger programs: a list of commands dependent on various triggers, including commands to execute other programs. Trigger programs can be easily packaged into menus for invocation by other users.

OFS tries to stay within a forms paradigm rather than create a new command box like OBE. Procedures are a collection of form 'sketches':[11]

- a precondition sketch which describes the form field values of interest, using a form template
- an action sketch which describes the changed values on various forms, using form templates
- 'pseudo-sketches' which allow for additional conditions like point of origin and operations like sending a form instance to a destination.

The new vocabulary of form sketches and pseudo-sketches is roughly analagous to condition boxes and command boxes in OBE. The OFS developers feel it is desirable to retain a form for everything and everything in its form.

Another way of specifying procedures is to use a simulation, in which a sequences of user operations is recorded for future use.[12] This can be extended to a procedure-by-example format, in which the user provides operation sequences on sample data and the system asks for conditions when two sequences differ.[13] In this last case, the condition could produce an iteration construct; in the other specifications, the only looping is the implied loop on all records in a file.

A proposed system from Siemens with more explicit iterations has been described as "a nonprocedural specification language for process reactions to trigger stimuli".[14] It includes a 'whenever' construct, which gives a logical condition whose change at any time from false to true will initiate specified actions.

Procedure handling: service procedures can be associated with the users who create them, with copies sent to others. Procedures can also be linked to the form type for general access.

When triggers are placed on a data base, a central monitor must initiate the procedure. When triggers are placed on messages for certain users, the initiation can be done by a central monitor or a

local, workstation process. This latter organization implies that the workstation is either always active or can be activated by message arrivals; timing triggers can be implemented with similar conditions.

Office control procedures

The OFS developers distinguish between a desk activity, a mail activity and a co-ordination activity.[15] Desk activities correspond closely to service procedures; co-ordination activities correspond roughly to office control procedures. Mail activities fall somewhere in between, depending whether they are specified by the user or centrally. However, the portion of OFS implementing procedures does not provide extensive co-ordination: decisions made after processing a form are difficult to handle, and there is no mechanism for passing procedural control from one workstation (user) to another.

OBE would handle co-ordination indirectly through a data base relation. If we wanted to say that department one must pass a form before department two, then we would have to create a field which was modified by department one.

An experimental system from Xerox Palo Alto Research Centre, Officetalk-D[16] uses a database with precedence relations to handle co-ordination. This specifies the sequence of activities through which a given task must progress. Workstation users are notified when

activities for which they are authorized are available for processing. An alternate scheduling process would require a user performing an activity to initiate the next task step if possible.

Two systems which directly address the notion of procedural flow between workers are the Business Definition Language (BDL)[17] and the Office Procedure Automation System (OPAS).[18]

BDL

BDL was an experimental system developed within IBM to explore high level generation of data processing applications. Its data transformations do not encompass all that we would want in an office system, particularly text processing. But there is an explicit document orientation and a Document Flow Component which shows the sequence of processes for a given document, like a purchase order or travel request.

To specify the document control procedure, users interact with a graph editor which creates a top-down hierarchy of graphs describing document flow. The graphs illustrate control flow by document flow - a correct document goes to one next step, an incorrect document goes somewhere else. Another description mechanism allows specification of actual processing at each step.

BDL is of interest partly for what it does not contain - general purpose communication and messaging. In keeping with the document paradigm, events are not easily triggered by a confirmation message from a given user; either there is a separate form for that purpose, or the condition is hidden outside the Document Flow Component. Similarly, the data processing applications are not perceived as requiring a separate timing mechanism to trigger clocked events or measure elapsed times.

A later development with some roots in BDL is the Office Specification Language (OSL).[19] OSL is intended to be more flexible, more interactive and less structured than data processing specification languages. The conceptual model of OSL involves documents but is 'function-oriented': its developers want to "focus on the end being achieved rather than the means."[20] This organizational perspective restricts its utility for general users. OSL is intended for specialist users, partly as a modelling tool and partly as a policy device.

OPAS

OPAS contains a forms processing component similar in most aspects to OFS, except that each process only has one resulting action. Individual processes manipulate form contents under specified conditions to yield output forms. A separate mechanism exists to link the processes.

A procedure specification form shows a sequence of activities to be performed, with triggers and conditions. As usual, the triggers are event-oriented (RECEIPT of a form, COMPLETION of an activity, ERROR in an activity) and the conditions are data-oriented. There are potential timing triggers for initiation of each activity, along with a parameter list, input and output forms, and error handling statements.

Each activity causes a forms process or another procedure to be initiated. These can be specified to run concurrently or in sequence. One can also give a specific workstation where the activity is to take place.

There are REPEAT actions, but these are intended to keep a procedure active rather than cause an iteration. That is, a procedure instance is an active entity: it must be invoked by a user or another activity. This is a slightly different concept than an OBE or OFS process in which it would be inactive but invoked automatically when its conditions are met.

This causes the user to need REPEAT statements to keep the procedure instance running, but it also permits dynamic update to the procedure definition. Once a procedure is active, the definition can be changed without affecting the already active copy. Alternately, if a procedure instance suspends execution, its procedure specification form

can be edited without affecting the main procedure definition. This yields a convenient way to patch around errors in unanticipated exceptions.

The listing of input forms required also serves as an execution condition, since the procedure can be active but awaiting their input. The timing triggers do not permit relative or variable times, 'one hour after receipt' or similar.

The procedure specifications form provides a fairly general purpose mechanism with a document and message orientation. It remains to be seen how easy the technique will be to use. The final form would be more effective if its two dimensions were used more creatively.[21] Also, some combination of a data flow graph like BDC and a procedure form like OPAS might prove particularly effective. The graph would give a flow overview, while the form would specify the details.

4.3 Future Office Models and Tools

The future of integrated interfaces appears to be in user interface management systems. The future of system design and user evolution may lie in better tools for representing both the interface under development and the prospective users. The future of procedural specification and the incorporation of better conceptual models appears to lie in knowledge representation techniques which can connect with user thinking rather than just user action.

These three areas are assessed in a companion report, for which a summary follows as an appendix.

Summary:User Interface Tools for Office Communicatins Sytems

This report is the third in a series of three reports prepared for the Office Communications Systems (OCS) Group of the Dept. of Communications. The first report, User Interface Componenets for OCS, looked at the principles of human-computer interactions, via a framework of interface levels. User Interface Design for OCS, the second report, focused on designing user interfaces for the next generation of office system products.

This third report considers software tools, formal techniques and knowledge representation applied to user interfaces. Each of these kinds of tools has been primarily a research effort, although in limited ways their influence has already begun to appear (eg. software tools in the Augment User Interface Serive).

The software tools form a user interface management system. Development of these systems has some parallels with development of data base management systems.* The dbms centralizes certain specialized functions in a data-oriented 'back-end' module. The user interface management system centralizes other specialized functions in a user-oriented 'front-end' module. This could be a software component or a separate intelligent workstation processor. The uims can

---

* A comparison suggested by J. D. Foley

provide dialogue control, command translation, language parsing, device mapping and assistance facilities.

Formal representations of user interfaces would be useful during design, implementation and testing. For programming languages, the existance of formal tools has led to compiler-compilers and test case generators. The second chapter of this report considers proposed tools for specification of a user interface, for prediction of user performance with the interface, and for analysis of user knowledge required for the interface.

Providing guidance on-line for the user demands some representation of the semantics of user actions and some representation of possible user goals and plans. Various techniques for utilizing conceptual and semantic level knowledge have been developed by researchers in artificial intelligence (AI). The third chapter surveys advances in this field and projects potential impacts on office systems. We conclude that a number of state of the art AI techniques have short term applicability in user interfaces for OCS.

A special listing of readings on AI in office systems is included at the close of chapter three.

References

1. Seybold, P.B., Comparing the Usability of Office Systems, AFIPTS Office Automation Conference, 1982, p. 225-236.

2. Smith, S.C., Requirements definition and design guidelines for the man-machine interface in C system acquisition, Report M80-10. The Mitre Corporation, 1980.

3. Gaines, B.R., The technology of interaction-dialogue programming rules, Intl. J. of Man-Machine Studies, 1981, p. 145.

4. Williges, B.H. and R.C. Williges, User Considerations in Computer-Based Information Systems, Technical Report for Engineering Psychology Program, U.S. Office of Naval Research 1981.

5. Ramsey, H.R. and M.E. Atwood, Man-Computer Interface Design Guidance: State of the Art, Proc. Human Factors Society 24th Annual Meeting, 1980, p. 87.

6. in ref [2].

7. in ref [2].

8. Ramsey, H.R. and M.E. Atwood, in ref [5].

9. Carey, T. User Interface Components for Office Communications Systems, Dept. of Communications, August, 1982.

10. Carey, T. User Interface Tools for Office Communications Systems, Dept. of Communications, March 1983.

11. Morton, J. et al., Components of Incompatability in Man-Computer Interactions, 8th Intl. Symposium on Human Factors in Telecommunications, 1977.

12. Rohlfs, S., Linguistic Considerations For User Interface Design, in N. Naffah, ed., Integrated Office Systems, North-Holland, 1980.

13. Marcus, A. Typographic Design for Interfaces of Information Systems, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 26-30.

14. see ch. 1 in ref [11]

15. Bennett, J.L., Management to Meet Usability Goals, AFIPS Office Automation Conference Digest, 1982, p. 163, with

credit to T. Gibb, "Design by Objectives", unpublished draft.

16. Hendricks, D. et al. Human Engineering Guidelines for Management Information Systems, U.S. Army Material Development and Readiness Command, November 1982.

17. Demers, R.A., System Design for Useability, CACM (24) 1981, p. 494-501.

18. Clamons, E.H., A Model for a Video Driven Common Operating System Language, 2nd Phoenix Conference on Computers and Communications, 1983, p. 591-596.

19. cf ch. 2 of ref. [11]

20. Halasz, F. and T. Moran, Analogy Considered Harmful, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 363-386.

21. Lakoff, G. and M. Johnson, Metaphors We Live By, U. of Chicago Press, 1980.

22. Smith, D.C. et al. Designing the Star User Interface, Byte, April 1982, p. 258.

23. Purvy, R., J. Farrell and P. Klose, The Design of Star's Records Processing, ACM Trans. Office Inf. systems, Vol. 1, p. 5.

24. cf Elmasri, R. and J.A. Larson, A User-Friendly Interface for Specifying Hierarchical Queries on an ER Graph Database, in J.A. Larson, Ed., Tutorial: End User Facilities in the 1980's, IEEE Press, 1982.

25. Podger, D.N., High Level Lanugages--A Basis for Participative Design, in Szyperski, N. and E. Grochla, ed., Design and Implementation of Computer-Based Information Systems, Sijthoff and Noordhoff Pub, 1979.

26. Smith, D.C., et al., Designing the Star User Interfae, Byte, April 1982, p. 270.

27. Williams, G., The Epson QX-10/Valdors System, Byte, September 1982, p. 65.

28. p. 274 in ref [25].

29. ref. [11]

30. Robertson, G., D. McCracken and A. Newell, The ZOG Approach to Man-Machine Communication, Carnegie Mellon University Technical Report CMU-CS-79-148, 1979.

31. Schultz, J. and L. Davis, The Technology of Promis,

Proceedings of the IEEE, September, 1979.

32. Beretta, G. et al., XS-1 - An integrated interactive system and its kernal, 6th ICSE, 1982, p. 340-349.

References Chapter 2

1. Benjamin, Art, Automating the Office of Yesterday, CIPS Conference 83, Ottawa 1983.

2. Ch. 3 of Carey, T., User Interface Components for OCS, Dept. of Communications, August 1982.

3. Mozeiro, H., A Human/Computer Interface to Accommodate User Learning Stages, CACM Feb. 1982, p. 100-104.

4. Schneider, M.L. et al., Designing Control Languages from the User's Perspective, in Beech, D., ed., Control Language Directions, North-Holland Pub., 1980.

5. Gilfoil, D.M., Warming Up to Computers: A Study of Cognitive and Affective Interaction Over Time, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 245-250.

6. Sirbu, M., Programming Organizational Design, Proc. ICCC, 1980.

7. Mumford, E. and D. Henshall, A Participating Approach to Computer System Design, Wiley Pub., 1979. Also Mumford, E., Designing Secretaries, Manchester University Press, 1982.

8. Pearsall, R.J., Technique for assessing external design of software, IBM System Journal, 1982, p. 211-219.

9. Schorer, P., Structure the Use, Computer, 1981, p. 77-86.

10. Meyrowitz, N. and A. Van Dam, Interactive Editing Systems, ACM Computing Surveys, Vol. 14 (1982), p. 325.

11. Denert, E., Specification and Design of Dialogue Systems with State Diagrams, in Morlet, E. and D. Ribbens, eds., Intl. Computing Symposium 1977, North-Holland Pub., 1977.

12. cf. ch. 1 and 2 of Carey, T., User Interface Tools for OCS, Dept. of Communications, March 1983.

13. Smith, S., Patterned Prose for Automatic Specification Generation, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 342-346.

14. cf. ref (3), ch. 3.

15. Dunn, R.M., A Control Structure Model for Interaction, in Guedj, R. et al., ed., Methodology of Interaction, North-Holland Pub., 1980, p. 53.

16. Mason, R.E.A. and T. Carey, An Approach to Prototyping Information Systems, CACM, May 1983.

17. Nicholls, R.E., Programming by the End-User, Infotech State of the Art report on Man/Computer Communications, Pergamon Press, 1981, p. 270-271. Also Yunten, T. and H.R. Hartson, Human-Computer System Development Methodology for the Dialogue Management System, CS Tech. Report CSIE-82-7, Virginia Polytechnic, 1982.

18. Williamson, H. and S. Rohlfs, The User Interface Design Process, Computer Message Systems, N. Noffah ed., North-Holland Pub. 1981. The second sentence is a quote from Keen, P., Information system and organizational change, CACM, Vol. 24 (1981).

19. Wynn, E., Linking User Responses to the Design Chain, AFIPS Office Automation Conference Digest, 1982, p. 169-175.

References Chapter 3

1. Card, S.K., W.K. English and B.J. Burr, Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys and Text Keys for Text Selection on a CRT, Ergonomics 21:8, 1978, p. 601-613.

2. eg. Human/Computer Interaction Series, ed. B. Shneiderman, Ablex Publishers.

3. Scapin, D.L., Evaluation of an Electronic Mail Language, 6th ACM European Regional Conference, 1981, p. 425-432.

4. Black J.B. and T.P. Moran, Learning and Remembering Command Names, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 8-11.

5. See the first exercise in Clark, I.A., Software simulation as a tool for usable product design, IBM Systems Journal, Vol. 20, No. 3, 1981, p. 272-292.

6. Scapin, D.L., ref. (3).

7. Geiselman, R.E. and Sanet, M.G., Notetaking and Comprehension for Computer-Displayed Messages, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 45-50.

8. Bair, J.H., Avoiding Working Non-Solutions to Office Communication System Design, Proc. IEEE Compcon, Spring 1980, p.

9. eg. Carey, T.T. and R.E.A. Mason, Information System Prototyping, to appear in INFOR, 1983.

10. Tombaugh, J., personal communication.

11. Ramsey, H.R., et al., Paper Simulation Techniques in User Requirements Analysis, Proc. Human Factors Society, 1979, p. 64-68.

12. Savage, R.E. et al., Design Simulation and Evaluation of a Menu Driven User Interface, Proc. Conf. on Human Factors in Computer Systems, 1982, p. 30-40.

13. Clark, I.A., Ref. (5).

14. Keen, P. and T.J. Gambino, The Mythical Man-Month Revisited, Proceedings APL Conference, 1980.

15. Tapscott, D., Investigating the Electronic Office, Datamation, March 1982, p. 130-138.

16. Conrath, D.W., R.H. Irving, C.S. Thachenkary and C. Zanetti, Measuring Office Activity for Bureautique: Data Collection Instruments and Procedures, Proceedings 2nd International Workshop on Office Inf. Systems, Saint Maximin, France, 1981.

17. Hoecher, D.G., Activity Sampling Applied to Interactive System Designs, Proc. Human Factors Society, 1981, p. 462-466.

18. Lewis, C., Using the "Thinking-aloud" Method in Cognitive Interface Design, IBM Research Report RC9265, 2/17/82.

19. Booth, T.L., R. Amwar and R. Lenk, An Instrumentation System to Measure User Performance in Interactive Systems, Journal of Systems and Software, Vol. 2, p. 139-146, 1981.

20. Wimmer, K.E., Research on Human Interface Considerations for Interactive Text Generation, Proc. ICCC 78, p. 720-732.

21. Rasmussen, J., Some Trends in Man-Machine Interface Design for Industrial Process Plants, in Computer Applications in Shipping and Shipbuilding, North-Holland Pub., 1980.

References Chapter 4

1. Ellis, C.A. and G. Nutt, Computer Science and Office Information Systems, Computing Surveys, 1980.

2. Dumas, P. and G. Du Roure, Office Modelling: The CETMA/KAYAK Families of Models, Proc. Workshop on Integrated Office Systems, St. Maximin, 1981.

3. Ellis, C.A. and M. Bernal, Officetalk-D: An Experimental Office Information System, Proc. ACM Conf. on Office Information Systems, 1981, p. 131-140.

4. Naffah, N., Communication Protocols for Integrated Office Systems, Proc. Workshop on Integrated Office Systems, St. Maximin, 1981.

5. Purvy, R., J. Farrell, and Paul Klose, The Design of Star's Records Processing, ACM Trans. on Office Information Systems, 1983, p. 3-24.

6. Zloof, M., Office-by-Example: A business language that unifies data and word processing and electronic mail, IBM Systems Journal, 1982, p.272-304.

7. ref. (6), p. 287.

8. Tsichritzis, D., Form Management, CACM 1982, p. 453-478.

9. Ellis, C.A., An Office Information System Based on Migrating Processes, Proc. Workshop on Integrated Office Systems, St. Maximin, 1981.

10. Carey, T. and R.E.A. Mason, Productivity Experiences with a Scenario Tool, Proc. IEEE Fall Compcon, 1981.

11. Hogg, J., O.M. Nierstrasz and D. Tsichritzis, Form Procedures, in Tsichritzis, D. ed., Omega Alpha, Tech. Report CSRG-127, U. of Toronto, 1981, p. 101-133.

12. Ellis, C.A. and M. Bernal, Officetalk-D: An Experimental Office Information System, Proc. ACM Conf. on Office Information Systems, Philadelphia, 1982, p. 134.

13. Attardi, G. and M. Simi, The Power of Programming by Examples, Proc. Intl. Workshop on Office Information Systems, St. Maximin, 1981.

14. Kofer, R., Saving Money while doing Empirical User Research, Proc. Intl. Workshop on Office Information Systems, St. Maximin, 1981.

15. Tsichritzis, D., OFS: An Integrated Form Management System, Proc. 6th Conf. on Very Large Data Bases, p. 165.

16. ref. (3).

17. Hammer, M. et al., A very high level programming language for data processing applications, CACM, 1977, p. 832-840.

18. Lum, V.Y., D.M. Choy and N.C. Shu, OPAS: An office procedure automation system, IBM System Journal, 1982, p. 327-350.

19. Hammer, M. and J.S. Kurrin, Design Principles of an Office Specification Language, Proc. NCC 1980, p. 541-547.

20. ref. (19), p. 544.

21. eg. Larson, J., A Data Manipulation Language for Electronic Forms, Proc. Compsac 81, 1981, p. 348-354.