A New User Interface Architecture
The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Work-programme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Work-programme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 ORD
United Kingdom

TELEPHONE UK (0223) 323010
INTERNATIONAL +44 223 323010
FAX +44 223 359779
ARPA Internet apm@ansa.co.uk

Copyright © 1993 Architecture Projects Management Limited

Architecture Projects Management Limited takes no responsibility for the consequences of errors or omissions in this Report, nor for any damages resulting from the application of the ideas expressed herein.
A NEW USER INTERFACE ARCHITECTURE

Yigal Hoffner, John Dobson§ and David Iggulden

ANSA, Advanced Networked Systems Architecture,
24 Hills Road, Cambridge,
United Kingdom CB2 1JP

This paper proposes a new user interface (UI) architecture which is intended to help designers with the internal structuring of the UI - the mechanism which animates and facilitates the dialogue between the user and the application.

The architecture is based on the assumption that the role of the UI is to bridge the gap between the language of the application and the language of the user. There are two major structuring concepts behind the proposed user interface architecture: the language stage concept and the language transformation step concept. The language stage concept advocates the introduction of several intermediate language stages between the user language and the application language. The language transformation step concept deals with the necessary transformations between the intermediate language stages and also the actions required within each language stage.

An example of the use of the proposed architecture in the design of a UI of a chess playing application is given. Finally, directions for future work concerning the proposed architecture are discussed.

1. INTRODUCTION

There is little consensus on how to structure user interfaces (UI's) internally or how to develop a UI reference model. Furthermore, there is a certain amount of confusion in the literature concerning UI architectures. There have been attempts to develop interaction models which can help in the design of the language of interaction between the user and the system at the sensory-motor interface [5], [6]. This, however, is not the same as indicating how to structure the UI internally. Architectures of UI's which deal with their internal structure have been suggested in the past [8], [14], [16]. Some of these use Foley's language analogy [5] as the structuring principle. This provides a view of the UI which is too rigid and static to be of much use when considering different interaction styles.

There are two distinct questions which the designer of a UI must address. The first, is: "what is it I am trying to express?" and the second is: "how do I structure the design?". This paper proposes a new UI architecture which is intended to help designers with the second question - the internal structuring of the UI. The proposed UI architecture does not tell the UI designer which style of interaction to choose, which I/O devices to use, how to design the interaction metaphor, or how to address the ergonomic and cognitive issues which arise in the context of human-computer interaction. The aim of the proposed UI architecture is to provide the designer with a

§ Computing Laboratory, University of Newcastle Upon Tyne, Newcastle,
United Kingdom NE1 7RU
structuring principle that allows the design of flexible, extensible and reusable UI's.

The architecture is based on the assumption that the role of the UI is to bridge the gap between the language of the application and the language of the user. There are two major structuring concepts behind the proposed user interface architecture: the language stage concept and the language transformation step concept. The language stage concept advocates the introduction of several intermediate language stages between the user language and the application language. Each language stage deals with a different aspect of the interaction and possesses different user and application knowledge. The language transformation step concept deals with the necessary transformations between the intermediate language stages and also the actions required within each language stage. Each transformation step incorporates a translation process which provides a vocabulary change and/or syntactic structure change, and also incorporates an interpretation process which changes UI context and which may result in feedback to previous intermediate language stages.

The proposed internal structuring of the UI is not based on the principle of separation of layers according to application dependent and application independent knowledge [2], [3], [8]. Nor is it based on the distinction between interactive and non-interactive components [2], [3], or on the particular structure of a language [5], [14]. Neither is it based on existing cognitive models of how the brain functions [1], [16].

1.1. Open problems in user interface design

Cockton [2], [3] has identified three open problems in the design of a user interface management system (UIMS). These are described as:

- interaction ergonomics, which addresses the provision of operator-usable libraries of interaction techniques and dialogue styles
- dialogue control, which is concerned with describing the (partial) order in which user actions may occur, overlap and interact
- separation, which attempts to draw a clean line between an application-independent user interface and the rest of the application

Our proposal is mainly concerned with separation, though it also has something to offer in the area of dialogue control. We can also show how to structure UI's for interaction ergonomics though we have no specific advice in the area of human factors.

As Cockton points out, separation is not the same as independence. Separation is a matter of classification: given a concern, one knows in which domain to put it. Complete independence is a stronger (and in practice unachievable) property of non-interference between the domains. Thus separation need not entail complete insensitivity in the UIMS to relevant facets of the application. The major novel feature of our UI architecture is that it proposes a structuring principle which readily permits separation while permitting interaction between those facets which indeed ought to interact (e.g. for the provision of a context dependent help facility).

Although the problems of interaction ergonomics, dialogue control and separation are still open, there is no reason to suppose that they are independent of each other nor that they can be solved piecemeal. In
particular, any structuring or mechanism for achieving separation will on examination prove to have made some dialogue control policy decisions. For example, the separation proposed in [2], which suggests defining and isolating a so-called non-interactive core (NIC) means that certain kinds of error may not be amenable to local correction (because the error detection and recovery mechanisms may be part of the core functionality). In distributed environments, this may place additional constraints on the structure of a remote dialogue controller.

An advantage we claim for our proposal is that it assists the system designer in exploring possible interplay between these problems in the particular area of design space that is being explored.

2. THE USER-APPLICATION GAP

The manner in which application functions are invoked and the manner in which they respond is distinctly different from the manner in which human users express themselves. Similarly, the manner in which applications provide data for the user and the manner in which users prefer to be presented with information is also distinctly different. In general, the gap between the language of users and applications can be ascribed to the difference between people as users and people as programmers, as well as to the difference between people and computers.

The application provides the required functionality through a set of procedural interfaces, in terms which, as far as possible, are independent of presentation and interaction style. The nature of these interfaces is such that they require an exact presentation of information. Each command will therefore have a single rather than multiple ways of being invoked. This simplifies the task of the application designer.

Users, on the other hand, generally want the information presented to them to be enhanced by rich metaphors. They often prefer to be able to express the same thing in a variety of ways and expect their intentions to be detected even though they are represented by different sets of actions. Users often expect different types of feedback to indicate that the machine has the appropriate interpretation of their intention. In addition to freedom of expression it is often necessary to allow for the motor and perceptual limitations of the user. For example, icons can be made relatively large, both to allow easy recognition and to help users point more easily at them.

The task of the UI designer is to bridge the gap between the way information is expressed to and by the user and the way in which it is expressed and expected by the application.

The UI acts as a bi-directional link between the user and the application and is responsible for presenting application state to the user in a manner amenable to appropriate interpretation. It is also responsible for allowing users to express their commands in a variety of ways by allowing, for example, soft and hard keys, or by providing different interaction styles to be supported. The UI can be regarded as a component which augments the information from the application by adding contextual information from the user's world. In addition, the UI can be seen as the component which abstracts from the variety of ways in which users express themselves when interacting with computers. The abstraction removes the information which is specific to the interaction style, presentation metaphors, user feelings and
motor acuity, thus presenting to the application only the information necessary for appropriate functions to be invoked.

3. TERMINOLOGY

It is not possible to treat the user interface in isolation from the application. In this section we therefore present the terminology necessary for proper examination of the UI topic.

3.1. User interface related terminology

The term "user interface" as commonly used does not make a clear distinction between three separate yet closely linked parts (Figure 1):

- the physical point of contact between the user and the system - referred to as the sensory-motor interface
- the mechanism which animates this interface - referred to as the user interface manager
- the point of contact between the user interface manager and the application program interface - referred to as the application program interface

![Figure 1: The User Interface: the User Interface Manager, the Sensory-Motor Interface and the Application Program Interface](image-url)

The distinction between the sensory-motor interface and the user interface manager is necessary in order to differentiate between concerns of appearances and behaviour, and the mechanisms which support these. We use the term UI to refer to all the concrete parts which support the interaction between users and computers, without making the distinction between the sensory-motor interface and the user interface manager.

3.1.1. The Sensory-Motor Interface (SMI)

The Sensory-Motor Interface (SMI) is the physical point of contact between the user and the system. The sensory part of the SMI defines the output of the system in the various forms (acoustic, visual and tactile) which a human is capable of sensing. The motor part of the SMI defines the input to the system in those forms which a human is capable of activating. In other words, the SMI defines the language by which the user interacts with the system and vice versa. The SMI, unlike the application program interface, is primarily concerned with presentation issues.
check can be made without board state knowledge whilst the pragmatics check requires board state.

4. USER INTERFACE MANAGER (UIM) DESIGN PROBLEMS

The major problem facing the UIM designer concerns the internal structuring of the UIM with a view to:

- deciding how the appropriate context for the interpretation of user actions and application responses should be set up in the UIM. This has two aspects:
  - how much application knowledge is to be given to the UIM, where should it reside and in what form
  - how is application knowledge, together with interaction style information, used to provide appropriate feedback to the user, both in the case of a permissible user action and in the case of a user error

- making the UIM software flexible, extensible and reusable

The problem of creating appropriate contexts in the UIM (to facilitate the interpretation of user actions and presentation of application response) is basically that of assigning application knowledge to the various parts of the UIM. In practice, it is not possible to have a UIM which is totally ignorant of the application. Thus, some distribution and possibly replication of application knowledge is always necessary. Factors which are influenced by distribution and replication of application knowledge across the UIM, and which have influence on the manner in which it is done are: designer convenience, run-time efficiency and performance, dependability, help facilities, error recovery, adaptivity and general feedback.

5. STRUCTURING CONCEPTS OF THE UI ARCHITECTURE

In order to simplify the task of designing the UIM, we propose that different aspects of the interaction between the application and the user be expressed by different languages. This allows a simplification of the problem of bridging the gap between the language of the application and the user. The modular nature of the UIM structure is based on this separation into intermediate language layers.

The structuring of the proposed UI architecture into a number of intermediate languages (the language stage concept), necessitates the complementary activity of translation between the language stages (the language translation step concept).

5.1. The language stage concept

The structuring principle behind the proposed UI architecture advocates the introduction of several intermediate language stages (Figure 2). Each language stage deals with a different aspect of the language gap and possesses appropriate knowledge of presentation and application issues. Such a structuring principle allows application knowledge to be assigned to each of the stages in the UI at design (or run) time.
3.1.2. The Application Program Interface (API)

The Application Program Interface (API) defines the interface between the application and the UI [11]. The interface defines the language through which the application provides its functionality to the user. The application should, in general, not be concerned with specific presentation issues and this is usually reflected in the API by the exact and single representation of each command.

Two implementations of the same application may have different APIs; the difference may lie with the granularity of application routines, the data structures and the method of invocation and response supported by each application. (This will be elaborated upon in §6.1.3).

3.1.3. The User Interface Manager (UIM)

It is the role of the User Interface Manager (UIM) to bridge the gap between the language used by the application and the language by which the user interacts with the system. From this point of view the UIM can be regarded as a mechanism similar in its effect to that of a human interpreter or translator. As shown in Figure 1, the UIM consists of software mechanisms which act as interpreter and translator between the SMI and the API. The UIM is used to animate the sensory-motor interface and the chosen style of interaction, as well as to accept user actions and interpret them.

3.2. Application related terminology

3.2.1. Application state, semantics and pragmatics

Application state is an internal representation of application entities, their attributes and the relations between application entities.

Application semantics (or operational semantics) is a term used to describe the application's general functionality, that is, the operations which can manipulate the application state. It is more instructive to view the application semantics in terms of pre- and post-conditions on application state. Such a view regards application semantics as general rules which specify what can be done by the application [17].

Application pragmatics are a description of what functions may be exercised by the application given a specific application state. Application pragmatics are thus the general semantics made specific according to the given application state.

Application knowledge is a general term for application state, semantics and pragmatics.

An example from the world of chess can serve to demonstrate these concepts. The application state is the position of the chess pieces on the board, together with the knowledge of whose turn it is to make the next move. Generic chess rules such as "a bishop can only move diagonally" can be regarded as application semantics; this semantic rule has to be further refined by rules which are chess board state dependent - "a bishop is allowed to move diagonally only if there are no pieces between the bishop and the destination position". A test to check whether a player's move is legal will consist of two stages: the first - a semantic check to see whether the Bishop is moved diagonally or not, and a second - a pragmatics check to see whether there are no pieces between the bishop and the destination position. The semantic
The gap between the user language and the application language is divided into several intermediate language stages. The intermediate languages allow user actions and application responses to be re-expressed in different ways. Thus, certain aspects of the interaction can be augmented by re-expressing them in the context of the user and application world where necessary. Alternatively, some user and application related information can be abstracted from the interaction by removing it when redundant.

There are several reasons for introducing more than one intermediate language stage:

- bridging the language gap between the user and the application by a single language stage is possible but not desirable. A single language which is sufficiently powerful to express both presentation concerns (on the SMI side) and application concerns (on the API side) is likely to be difficult to define. In addition, a single language would restrict the modularity of the UIM thereby affecting its flexibility and reusability

- in most cases, to design an interpreter capable of bridging the application and user language gap in a single step places an unnecessary burden on the designer

- breaking the translation stage into a sequence of intermediate stages, if carried out judiciously, can offer the benefit of modular design: extensibility, reusability, and the decoupling of modules from changes in other modules

The important point about the proposed UI architecture is that the same structuring principle lies behind the intermediate languages. This is analogous to the ISO Open Systems Interconnection (OSI) reference model [12], which deals with the complexity of interconnection by decomposition into layers. The structuring principle, in this case, is based on the relationship between the layers which is that of "uses the services of". Each layer uses the services of the layer underneath it and provides extended and/or additional service(s) to the layers above.

5.2. The language transformation step concept

5.2.1. The Canonical Translation and Interpretation Step (CTIS)

The necessary transformations between the intermediate language stages and also the actions required within each language stage are carried out by a
transformation step called the *Canonical Translation and Interpretation Step* (CTIS) (Figure 3).

Each transformation step incorporates a translation process which provides a vocabulary change and/or syntactic structure change. Each transformation step also incorporates an interpretation process which changes UIM context. The context of a CTIS contains either or both of the following: application-dependent knowledge; local state determined by previous inputs. The interpretation process can change the context according to the input, the current context and the operational semantics of the interpreter. As the translation process can be made context dependent, application knowledge may be incorporated into the various stages of the UIM. The structure of the Canonical Translation and Interpretation Step is shown in Figure 4.

![Figure 3: The CTIS performs the necessary transformations between intermediate languages together with local interpretation](image)

The output of the CTIS is usually in a different language from that of its input(s) and the translation process is based on the relation between the two languages. This relation can be static or dynamic, i.e. it can be influenced by the current context. In addition the translation process can route its outputs to more than one output channel.

It is convenient to make each canonical step accept input from more than one input channel (e.g. for multi-media interactions). This requires a distributor to be responsible for the synchronization of input channels.

5.2.2. The use of the Canonical Translation and Interpretation Step in the design of a UIM

Canonical translation and interpretation steps can be combined as shown in Figure 5. In this example, a change which takes place in a CTIS as a result of a user action can be fed from one uni-directional channel to the other uni-directional channel, thus allowing different levels of feedback. The feedback
in such a case may be highlighting an icon as a result of selection or an error indication when the user attempts to perform an action which is not allowed.

FIGURE 5: The use of canonical translation and interpretation steps in the design of a bi-directional UI

Several CTIS's can also be combined to provide a sequence of bi-directional language transformation steps (Figure 6).

5.3. Summary of proposed UI architecture

The UI architecture proposed in this paper attempts to simplify the complex task of the UIM design by expressing different aspects of the interaction between the application and the user by a number of intermediate languages. Each language re-expresses the actions of the user and the application by addressing different aspects of the interaction. This allows a simplification of the problem of bridging the gap between the language of the application and that of the user. The modular nature of the UIM structure is based on these languages.

FIGURE 6: A sequence of canonical translation and interpretation steps

The proposed UI architecture consists of a series of language transformation steps. Each transformation step consists of translation and interpretation processes. It is not a static model of the translation process, since associated with each language transformation step is a Language Interpreter which is responsible for obeying the instructions coded in each language. In effect, each language is a program for some virtual machine and the model of the transformation process at each stage is that of an interactive interpreter. Each interactive interpreter is responsible for both interpreting the code aimed at the current stage and translating the rest of the code into a language for the next stage. The complete model can therefore be regarded as a set of interactive interpreters. As with any interactive interpreter, it is a design decision just how much code is interpreted and executed at each language stage. In a static interaction model, when the whole of a unit of
utterance (eg "Queen to d5") has to be assembled before any action is taken, the interpreter will be comparatively coarse-grained; but this need not necessarily be the case. It is possible to implement an interaction style which, for example, highlights a queen as soon as she has been selected and before the destination of the move has been announced.

The flexibility of multiple linguistically-specialized CTIS's also allows dialogue control to be distributed among the various steps. It is thus possible to bind dialogue structures to feedback and error interpreters more dynamically than is possible with conventional lexically-based analysis of user inputs. This area of dialogue control requires further work.

6. APPLYING THE UIM STRUCTURING CONCEPTS IN THE DESIGN OF A UIM FOR A CHESS PLAYING APPLICATION

The architectural concepts of intermediate language stages and canonical translation and interpretation steps have been used in the design of a UIM for a chess playing application [9] which will now be presented.

6.1. Definition of problem and resources

6.1.1. The chess playing application

The chess playing application is required to:

- check the legality of user moves according to the rules of chess and report special conditions such as illegal moves, pieces removed, check and checkmate
- respond to a legal user move with an application move

6.1.2. The SMI and the Pixel identifier language

In order to demonstrate the flexibility of the proposed UI architecture we chose a direct manipulation mode of interaction [15]. The sensory-motor interface contains a graphics screen, a locator (e.g. mouse) and an on/off switch (button) (Figure 7).

The locator moves are translated through an I/O device into a Pixel Identifier Language which consists of pixel coordinates. The button presses are translated into on/off state changes which indicate when the user has made a particular selection of a set of pixel coordinates.

A move will consist of: pointing to a board location of the piece to be moved, a button press, pointing to the "to" circle on the screen, a button press, pointing to the destination board location, followed by a button press.

When the necessary parameters for specifying a move are collected by the UIM, it will call the application and await a reply. The reply will indicate whether the user move is legal and also indicate the outcome of the move in terms of which piece is to be removed, or if there is a check or checkmate situation.

6.1.3. The API and the Application invocation language

Application routines are invoked by the Application Invocation Language through the API (Figure 8). The application invocation language describes
FIGURE 7: The sensory-motor interface (SMI) and the Pixel identifier language

the necessary procedure parameters in the appropriate format and in the appropriate order.

Call Select
(Array-Board [Pos])
and
Call Move
(Array-Board [Pos])

Or
Call Move
(Array-Board [Pos, Pos])

Application invocation language

Call Move
(Array-Board [Pos, Pos])

Application program interface (API)

FIGURE 8: The application program interface (API) and the Application invocation language

Different implementations of the same application may vary in the manner in which they interact with their environment. This can be for several reasons:
different implementations may use different data structures to describe application entities. For example, one application may specify a move as:

PROC Move (3,7; 4,5), whereas another application may specify the same action as:

PROC Move (31; 37) or by denoting moves by relative changes in position:

PROC Move (+1; -2)

the granularity of the calls may vary so that of an application to accept the same move specified above would require:

PROC Select-piece(3,7), followed by PROC Move-to (4,5).

7. THE TASK OF THE UIM

In this example the UIM has to translate between:

- the pixel identifier language which describes the coordinates of the screen point to which the locator is currently pointing (Figure 7)
- the application invocation language which calls the appropriate procedures in the application (Figures 8)

This point is demonstrated by concentrating on the locator input. However, the same principles apply to different modes of input and output and to their combinations. Issues such as where and how locator and button input are combined, as well as buffering of user command segments are not described here but can be dealt with by the model. Such issues are tied closely to the difference between the granularity of application routines and the granularity of the operations of the user language.

8. DESIGN DECISIONS

Design decisions are concerned with how many intermediate language stages the UIM is going to have, how much application knowledge is going to be assigned to each stage and what is the task of each stage. These decisions depend largely on the application and the constraints and requirements imposed on the designer from technological, commercial and other points of view. A set of constraints which impose time, man-power, technology and money limitations may cause one solution to be be preferred over another. Similarly, a requirement for a flexible, re-usable and extensible UIM can be quite different from that of a "one off" product.

9. THE SINGLE CANONICAL TRANSLATION AND INTERPRETATION STEP UIM

Let us suppose that the objective is to design the UIM quickly without taking into account issues such as extensibility, flexibility and re-usability. In such a case, the designer may choose to translate the pixel identifier language directly into the application invocation language which invokes the appropriate application routines (Figure 9). This solution requires a single CTIS.

Having defined the API to expect information which identifies the data structure in which the chess board state is stored, it will be necessary for the
CTIS to know how to map pixel selection onto the appropriate data structure references. The CTIS needs to know which pixel coordinates correspond to which board positions and to the "to" circle on the screen. Provided each selection of a screen location corresponds to the appropriate entities and is in the correct order, the CTIS can substitute the coordinates with the appropriate application invocation language description of a move.

To make the analysis of the user input more tractable, when expressed in the pixel identifier language, mappings between the pixel identifiers denoting an application object and the name of that object are required. Otherwise it will be necessary to define a pixel location dependent grammar for analysing the user input. This is an incredibly tedious and error prone activity for a designer.

The CTIS needs to know how the board is represented and which pieces are on the board. It can therefore perform certain checks which correspond to application pragmatics and semantics. For example, the CTIS can check that the first selected board position actually contains a piece to be moved. If it does not, there may be no point in continuing the translation process and error feedback may be sent to the user. How much application knowledge to assign to the CTIS is a design decision. The problem with this implementation of the UIM is that it lacks any indication of how to structure the single CTIS internally. The transformation performed by the single CTIS is complex and unless further structuring is employed it is difficult to see how feedback issues can be resolved in order to provide the user with the appropriate feedback. Further structuring of the single CTIS, resulting in the introduction of additional CTIS's are suggested in the next section.
10. THE MULTI CANONICAL TRANSLATION AND INTERPRETATION STEP UIM

In this example we suppose the designer chooses to incorporate several intermediate CTIS's in the UIM (Figure 10). These correspond to the following intermediate languages:

- **The Geometric Shape Language**: this is a language which describes the shapes of the board, the chess pieces and the “to” circle (Figure 10). This language addresses an aspect of the nature of the chosen interaction style and metaphor. This language allows the user action of selecting a specific pixel to be re-expressed as the selection of a geometric shape represented by the set of pixels which include the selected pixel.

- **The Chess Game Language**: this is a language which describes the game of chess (Figure 10). The shapes selected in the geometric shape language stand for chess pieces.

11. EXTENSIBILITY AND RE-USABILITY

The test of the flexibility and extensibility of a UIM takes place when changes to the existing system are necessary. The test of reusability takes place when trying to reuse existing UIM parts in the design of a new system. The following sub-sections describe the changes which are likely to affect the UIM.

11.1. Changes in the application

- Changes to the existing application: if another chess playing application is given to the designer, only the translation stage between the generic board game language and the new application invocation language need be modified.

- Different application in similar application domain: the advantages of the multiple intermediate language approach becomes clear when we consider using the same UIM as a front end for a checkers playing application. The designer will find it relatively easy to modify the chess game and application invocation languages to accommodate a game of checkers. However, the re-design can be made even easier by introducing another intermediate language (Figure 11):
  - **The Generic Board Game Language**: this is a language which describes generic board games in an abstract form.
FIGURE 10: The application of several canonical translation and interpretation steps (C\textsc{tis}) (only user input is shown)
11.2. Changes in user requirements

- changes in the **Ergonomic and Cognitive Model of the User (ECMU)** [9]: The ECMU is the designer's model of the user from an ergonomic and cognitive point of view. In this example, the ECMU will indicate to the designer whether the user prefers to play blind chess rather than have a continuous display of the chess board. In such a case a command language mode of interaction UIM is more appropriate [15]. Additional intermediate languages can be attached to the UIM at the geometric shape or the board game, or at the application invocation intermediate language level (Figure 10). The intermediate languages will have to perform the task of lexical and syntactic analysis to check whether the commands which the user types are correct. No other changes will be required.

- modification of the shape of the board or the pieces can easily be done by changing the definition of geometric shapes in terms of pixel coordinates without altering the rest of the UIM or application.

11.3. Changes in technology

- **I/O technology**: minor changes in I/O devices will probably only require change of the pixel to geometric shape mapping. Any radical changes to I/O technology would probably require a redefinition of geometric shapes. For example, a three dimensional direct manipulation mode of interaction (e.g. with a dataglove) can easily be accommodated by changing the geometric shape language from describing two dimensional shapes into a language describing three dimensional shapes.

- **communication technology**: if the UIM is implemented on a distributed system and the communication between the application and the UIM is done through a network, the only change which may have to take place is in the application invocation language.

- **computing environment technology**: the application invocation language may have to change depending on the type of software environment. For example, a C language support environment would require a different type of interaction from a Modula-2 [18] or SmallTalk environments [7].

12. FEEDBACK ISSUES

Use of the four intermediate languages advocated by this particular design suggests a method of choosing the appropriate level of feedback to the user. As each intermediate language can have different knowledge of the interaction, it is possible to arrange the required type of feedback to be given by the intermediate language with the relevant knowledge.

If, for example, a player points with the locator to a board location which has a chess piece displayed on it and clicks the button, a single pixel is selected by the I/O device (Figure 12). The pixel coordinates are transferred to CTIS\textsubscript{1}. This CTIS contains the pixel maps which relate sets of pixel coordinates to a geometric shape. Thus, the selection of a specific pixel by the user is interpreted by CTIS\textsubscript{1} as if the player intended to select the relevant geometric shape of which the selected pixel is a member. The CTIS thus translates the input to a selection of a geometric shape which denotes the
FIGURE 11: The introduction of a generic board-game intermediate language
chess board location pointed to by the player. The next CTIS which receives
this information translates it into a selection of the respective chess board
position. The information is now expressed in terms of the chess game
language. If the designer decided to distribute application knowledge
relating to the position of the chess pieces to this intermediate language
stage, then chess board position selections can be associated with a selection
of a specific chess piece. Thus, for example, the information received can be
interpreted not just as:
   Select Chess-board(6:3), but also as:
   Select White Rook at Chess-board(6:3). This information can be
   recorded in the context of CTIS₂ and/or passed on to the next CTIS if
   necessary. It can also be sent to the CTIS₃ which spans the chess game
   language and the geometric shape language thus being in effect feedback to
   the user. CTIS₃ translates the command:
   Highlight White Rook at Chess-board(6:3) to its equivalent in the
   geometric shape language as:
   Highlight Square51. In addition, if the geometric shape language
   contains the geometric definition of the chess pieces, then CTIS₃ can also
   request to flash the White Rook icon (or shape6 as it is known to CTIS₃)
   which resides on Square51 by issuing:
   Flash Shape6. This in turn is translated by CTIS₄ into the relevant
   commands to the I/O device in terms of pixel coordinates.

In the case of a user error, an intermediate stage which has knowledge of the
screen pixels can give the user feedback in the form of: "can't select pixel
(34:77)". A further intermediate language which deals with geometric
shapes can give a more meaningful response such as: "can't select geometric
shape 31". A further stage which has knowledge of application entities will
be able to respond with: "can't select a board position without a chess piece",
or "can't select an opponent's piece" (Figure 13). Of course, the ability of a
stage to respond in this way depends not only on the intermediate language
but also on the rules and pragmatics of chess available at that stage. Thus,
in order to give the above two reactions, the stage will either have to know
about chess rules and the state of the chess board, or it will re-express an
error response coming back from the application in terms which are more
amenable to the user. A combination of the responses possible from different
intermediate stages can provide a message together with the board location
or piece flashing to provide an enhanced form of error indication.
FIGURE 12: User actions, such as pointing and clicking on a board position, are interpreted as the selection of the piece which resides on the selected board position.

FIGURE 13: Different intermediate language stages provide different feedback in case of a user error. The designer has to choose the level with the appropriate language and provide it with the relevant application knowledge.
13. APPLICATION KNOWLEDGE DISTRIBUTION AND REPLICATION

An important point to notice in both the single and the multi CTIS examples, is that because the language translation processes are general the designer can choose how much application pragmatics and semantics information can be put into each. The UIM designer may choose to denote some chess rules in the board game language step. In this case the rules will have to be defined in terms of the intermediate language and can be used in the translation stage to check whether user actions conform to the rules. Thus, there is no need to defer all semantic and pragmatic checks to the application itself.

The decision about how much application-knowledge should be distributed and replicated is one of tradeoff between design time effort and the amount of communication traffic required at run-time. At design time, the designer has to replicate the appropriate application knowledge as well as distribute it to the appropriate location in the UIM, having performed the appropriate transformations.

Related to the issue of application knowledge distribution and replication is the problem of the interpretation of user actions. The question of user intentions and how best to deal with them have not been fully investigated. For example, in the context of chess playing, how are the application and/or the UIM modified if a user wants to be able to distinguish between commands such as: "try this move without committing" and "commit to this move".

Of course such distinctions between the options must be presented to the user at the SMI and must be recognized by the UIM. Several questions arise:

- given that the user wants to try moves without commitment, should copies of the chess board state be made or should moves simply be performed on the same board with a roll-back option (e.g. Undo option)? In the first case, should the replica board be kept in the UIM or at the application?

- how does this influence the intermediate languages and the tasks associated with each intermediate language? Where should replicas of the chess board for "non committed" moves reside - at the application or the UIM and if so at what intermediate language stage?

These issues relate to the replication and distribution of application state, semantics and pragmatics. Basically, given the intermediate language structuring principle, the question is where application knowledge will reside and in what form.

An interaction style which requires a continuous display of application state requires the presentation component of the UIM to have the appropriate knowledge to display the application state. The relevant application state can, of course, be requested from the application by the UIM. For example, if the screen display requires refreshing every few milliseconds, the UIM will have to request the application to send it the relevant state information. Different interaction styles require different application knowledge to be replicated and distributed to the UIM.
14. EXISTING UI DEVELOPMENT ENVIRONMENTS

It is interesting to consider available UI development environments in the light of the proposed architecture [10]. These environments (in addition to providing specific SMI features which dictate a distinct style of interaction) attempt to address the problem of relating activities at the I/O device level (e.g., locator moves, button clicks, key strokes) to activation of application routines. Mechanisms which facilitate this relation, are partly provided by systems such as X™ Windows and its associated toolkit and widget set [19], the Apple Macintosh™ and its toolkit and other facilities such as graphics packages [13]. Windows, for example, allow the mapping of user actions such as mouse clicks and pointing devices to the invocation of specific application routines. They can therefore provide the initial translation from pixel identifiers to UIM or application objects. This, however, may not be sufficient for the design of flexible and extensible UIM’s. Such facilities alleviate some of the problems designers face but do not offer a comprehensive solution to UIM designers.

15. CONCLUSIONS AND FUTURE WORK

The chess application UIM example serves to demonstrate the validity of the proposed architecture as well as the type of intermediate languages which will form the internal structure of a particular UIM. More examples will be necessary before any generalizations can be made and decisions as to whether there exist any universal intermediate languages for all interaction styles and for all application domains. Thus, any decisions about the usefulness of this method with regard to standardization of the internal UIM structure will require further work. It is possible that agreement on intermediate languages will only be possible with relation to specific interaction styles and within specific application domains.

The decomposition of the user interface into intermediate language stages with their associated sequence of CTIS’s requires several design decisions. Issues such as the choice of language(s), the granularity of interaction (i.e. what does each stage do and what does it pass on to the next) and so on now become crucial. The important set of decisions however, is:

- how many and what intermediate language stages are necessary
- how to spread knowledge (pragmatics and semantics) of the application across the various interpreters

These two issues are closely tied to each other since the interpretation and translation at each stage will be performed in the context of the knowledge of the stage.

We have not indicated the relationship between the interaction style supported by the SMI and the internal structuring of the UIM. We believe that there is a strong link between these, since the interaction style dictates

---

™ X Windows is a trademark of MIT.
™ Macintosh is a trademark of Apple
™ Star is a trademark of Xerox.
which aspects of the user's world have to be represented in the UIM, thus influencing the type of intermediate language or languages which are necessary. Further work is necessary in order to clarify this relationship. A classification of interaction styles which indicates the type of intermediate languages necessary would be extremely useful.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the generous help of Professor John Monk from the Open University, Rob van der Linden and Hugh Tonks from the ANSA project. The authors would also like to thank Gilbert Cockton from the University of Glasgow for his useful comments.

REFERENCES


[12] Information processing systems - Open systems interconnection - Basic reference model.


