

Application Level User Interfaces for Various Media

D. I. Clark and J. D. Newmarch
Information Science and Engineering
University of Canberra

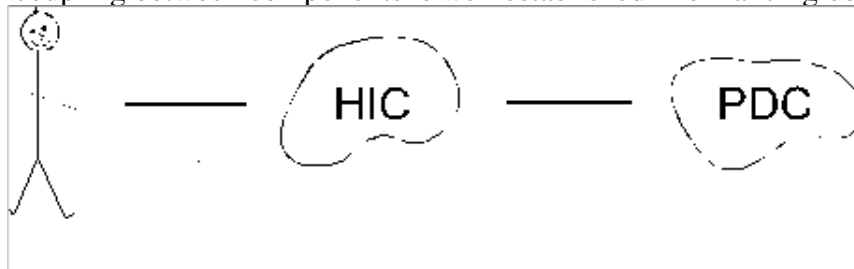
Abstract

Applications are increasingly being written with the user interface being implemented in differing media. In this paper we show how a variety of business-oriented applications can be developed so that the user interface can easily be changed from one medium to another. We abstract out the user interface requirements, show how they can be met by CUA panels and how the panels can be implemented in various media. We then use design patterns to structure the architecture of the application to facilitate easy changes between user interfaces.

1. Introduction

Since the advent of widely available graphical user interface toolkits such as Windows API and Motif, much attention has been paid to the design of human computer interfaces. Earlier interfaces were confined to text. More recently web based and phone based interfaces have also become more widely used.

Coad and Nicola [Coad+ 93] identify the problem domain (PDC), the human interaction (HIC), the data management and the task management as the four main components of object oriented design. The principle of loose coupling between components is well established in enhancing object oriented reuse.



Interest has been growing in application-oriented frameworks [Fayad+97]. These limit the scope of the problem and allow a more customised set of objects to be built. In this paper we explore how a fairly common type of application can benefit from a smaller and more specific set of user interface elements, and how these elements can be realised in a variety of user interface media.

2. User Interface Elements

The HIC can be implemented in many ways: by a graphical user interface, a command line interface, and so on. The typical set of user interface elements has been defined by the IBM Common User Access

(CUA) [IBM 88] to include elements such as

- check box
- entry field
- prompt

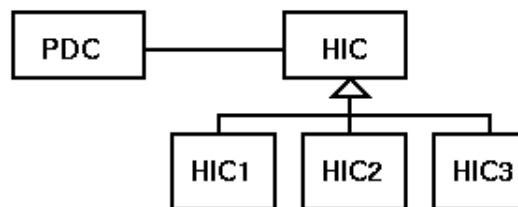
The set of interface elements is intended to be independent of any particular user interface, and may be thought of as the primitive building blocks for any interface. These are quite low level components, and may be combined into higher level "panels" such as

- menu panel
- entry panel
- information panel
- list panel

This early CUA specification showed how to target these panels to a cursor-addressable terminal, a programmable terminal and a graphical terminal. Later versions separated these into two documents, to allow for the greater capabilities of graphics terminals.

In current pattern-language terms, the user interface could be implemented by suitable Factory objects: a MenuPanel Factory could supply a CursorAddressableMenuPanel, a ProgrammableMenuPanel or a GUIMenuPanel.

In UML-like notation, the PDC/HIC interaction with this added flexibility gives a diagram such as



It should be noted that this does not cover all possible applications: drawing tools, word-processors, spreadsheets, etc, all rely so heavily on presentation aspects that it would not be possible to separate the PDC from the HIC so cleanly.

3. User Interface Requirements for Application Frameworks

Application frameworks involve a careful set of tradeoffs. Particular applications must be generalised, but not so far that their needs are lost - the generalisation must be taken as far as possible, but no further. The restraining influence is how easy it is then to implement each application within this framework. Here we are considering applications that are already partitioned into HIC and PDC components, and so the generalisations and the constraints must apply to both components.

Despite the almost endless variety of business-oriented applications, each with its own particular interface, user interfaces can often be built from a relatively few components. A typical scenario from a user of a banking application may be:

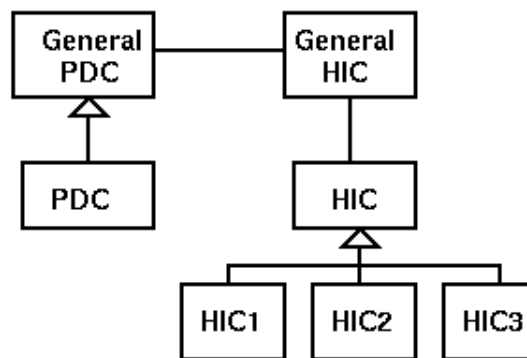
1. user identifies him or herself
2. user selects an account on which to operate
3. user is told the balance of the account
4. user selects transaction, such as withdrawal
5. user enters amount to be withdrawn from the account
6. user is told the new balance of the account
7. user terminates the application.

The requirements of this and a wide variety of other business-oriented applications can be summarized as the ability to:

1. identify a user (number 1 in list above)
2. select an object for transaction from a list of objects (number 2 in list above)
3. select a task and related sub-tasks (number 4 in list above)
4. enter fields of related data (number 5)
5. create new objects
6. amend existing objects
7. display fields of related data (numbers 3 and 6)

This interface functions at a more application-specific level than that of CUA "panels", and additionally abstracts the HIC requirements of particular applications. Thus, it is a *superclass* of application HICs, that *uses* various panels.

This produces a diagram such as



4. User Interface Components in various media

In this section we explore how the requirements identified above can be met by the use of *panels*, and how the panels are, or can be, implemented in various media. CUA defines a set of panels:

- menu panel: displays one or more lists of choices
- entry panel: displays fields in which users type information and select choices
- information panel: displays information that cannot be changed
- list panel: displays lists of items with single or multiple selection
- logo panel: displays ownership and copyright information

A panel in turn is made up of a set of panel components such as checkbox, command area, entry field

and so on.


4.1 Components as Panels

1. Identifying a user can be performed using an Entry panel, in particular a Parameter Entry Panel. Such a panel would contain a title and a set of prompts for text fields. The text fields would be for user identifier, and possibly password.
2. Selection of a task may be performed using a Menu panel.
3. Selection of an object may be performed using a Menu or a List panel.
4. Entering fields of related data may be performed using a Parameter Entry panel.
5. Creating a new object may be done using an Entry panel to enter an identifier for the object, and then using a Form Entry panel to set values.
6. Amending an object may be done by first selecting an object from a Menu panel, and then using a Form Entry panel to amend values.
7. Displaying fields of data may be done using Information panels.

4.2 GUI based

Let us look at the details of GUI based User Identification. The title, instructions and set of prompts could be set by Labels. The text fields would be single line TextField boxes. Completion (or cancellation) of entry would be done by PushButtons.

Such an interface would look like:



The image shows a screenshot of a graphical user interface window. The window has a title bar with a red 'X' icon on the left and standard window control icons (minimize, maximize, close) on the right. The title of the window is "Welcome to the Bank". Below the title bar, there is a line of text that reads "Enter name and PIN and press Enter". Underneath this text, there are two input fields. The first field is labeled "Name" and the second field is labeled "PIN". Both fields are empty and have a small vertical cursor on the left side. At the bottom of the window, there are two buttons: "OK" on the left and "Cancel" on the right.

The mapping would be summarised as

Panel component	GUI component
checkbox	checkbox/radiobutton
command area	textfield/textarea
entry field	textfield
headings	label
instructions	label
message area	label
panel title	label
pushbutton	pushbutton

4.3 Web based (WUI)

In a web based interface the equivalent of a window is a form. A form to allow user identification is given below. It uses lines of text for the title and instructions, a textfield for the name, a password box for the pin and a submit button.

Welcome to the bank

Please enter your name and personal identification number.

Name

PIN

Other HTML components can be used to implement panel components as shown below.

Panel component	HTML component
checkbox	option
command area	text field/text area
entry field	text field/password
field prompt	text in HTML
headings	text in HTML
instructions	text in HTML
message area	text in HTML
title	text in HTML
pushbutton	submit/reset/anchor

4.4 Text based (TUI)

In a text based interface there is no physical equivalent of a window or gui widgets such as pushbuttons.

However classes can be written to simulate these. A console menu class to allow user a choice of action, a console pick list to allow object selection, a text box for entry of data, and a password box for entry of non-echoing data. Information can be displayed via lines of text. Finally a console window would act as a container for these components, the event loop being simulated by a console menu. The mapping is given in the table below.

Panel component	HTML component
checkbox	input box
command area	input box
entry field	input box/password box/console pick list
field prompt	lines of text
headings	lines of tex
instructions	lines of text
message area	lines of text
title	lines of text
pushbutton	console menu

A console window to allow user identification is given below. It uses lines of text for the title and instructions, a textbox for the name, a password box for the pin and a submit button.

```
Welcome to the bank
Please enter your name and personal identification number.
Name : John
PIN : ****
Press <enter> to submit.
```

4.5 Phone based (PUI)

As in the case of text based interfaces, there are no physical equivalents of windows, buttons, etc. However, they may again be simulated by writing classes similar to those in text based interfaces. At a low level of implementation, each will use text to voice components for display, prompt, etc, and receive digits for both data and choice of action. The only difference from TUI is that there is no need of a password box .

5. Putting it together

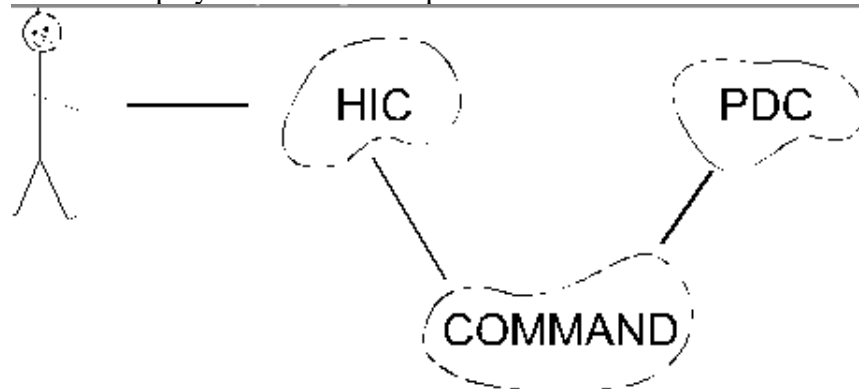
In this section we describe how to build an application so that the user interface medium can be easily changed. To do this we will use and interaction between the Command and Facade design patterns.

5.1 The Command pattern

The Command pattern [Gamma+95, pp233ff] is one of the simplest yet most powerful of design patterns. It is used when there is a need to decouple an object which initiates an operation from the object which carries out the operation. It does this by encapsulating a request as an object.

When used to decouple the HIC from the PDC, the command object will often be a menu item. A concrete command class will inherit from an abstract class Command containing a single abstract

method execute(). When a menu item has been selected by a user the execute method of the associated concrete command object is invoked. The concrete commands link the HIC to the PDC. Any user action which requests user information from or changes to the state of the PDC are passed from the HIC to the command and then to the PDC. Any information from the PDC is passed to the command which forwards it to the HIC for display. The Command pattern is thus:



5.2 The Facade Pattern

The Facade pattern [Gamma+95, pp185ff] is used when there is a need to decouple the clients of a subsystem from the individual components of the subsystem which carries out their requests. It does this by inserting an interface object (the facade) between the clients and the components of the subsystem. When the client of a subsystem interacts with the subsystem, it sends a request to the facade object which then forwards it to the appropriate object(s) in the subsystem. Responses are sent by these components to the facade object which then forwards them to the client. Among the consequences of the Facade pattern are insulating clients from any changes to the subsystem, and decoupling subsystems by reducing (to one) the number of objects in the subsystem that clients have to interact with.

5.3 Command Facade Interaction

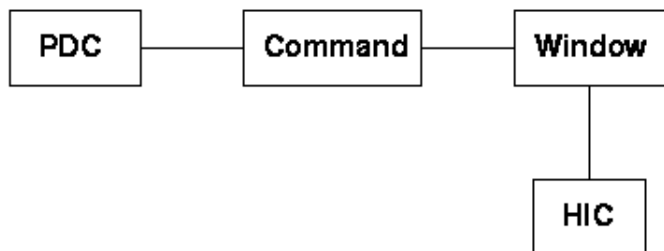
In order to simplify the substitution of one user interface medium with another we use a combination of the Command and the Facade patterns. We treat each window as a facade, and user requests are passed to a command object via the facade. The command object does not interact directly with widgets such as check boxes. Instead the facade has a number of fields representing data which the user has entered (account number, deposit amount, etc.) The command object queries the facade object for the data fields which are then used to parameterise requests to the PDC. Responses from the PDC are forwarded by the command object to the facade object. The execute method of the command object is then

```

get fields from window
ask PDC if fields are OK // editing
if fields are OK
    request to PDC with fields as parameters // query, update
    inform window of result
else
    ask PDC what is wrong
    inform window of what is wrong
end
  
```

The facade object then forwards the responses to the medium specific object for display. One difference

in this use of a window as a facade is that normally a facade is a singleton, whereas here there will typically be several in an application. However the same facades will be kept when the user interface medium changes. Each facade window is an abstract class containing abstract getField() and displayMessage() methods. It is these that the command object sees. For each medium a concrete window class inherits from the abstract facade window and implements the methods using the widgets applicable to that medium. All commands are then independent of the HIC medium and both commands and PDC can be used without change irrespective of the medium used.



5.4 Examples

Our first example is "The Weighing Game". Whilst it is not a conventional business application it has the same user input requirements. In the weighing game, a user is given 12 coins and a pair of scales. One of the coins is fake. Its weight differs from the weight of the genuine coins. The player is given 3 weighings to gain enough information to decide which coin is fake and whether it is light or heavy. Each weighing consists of selecting which coins to put on the left scale and which to put on the right scale. The response to each weighing is "right side is heavier", "left side is heavier" or "sides are equal weight". After 3 weighings, the player chooses the fake coin and whether it is light or heavy. The response will be "correct" or "no, it was coin ... and it was ...".

In this application there is a window to get a player's weighings, and a window to get the player's choice of fake coin and weight. Examples of these for a web based interface are shown below.

Your third weighing

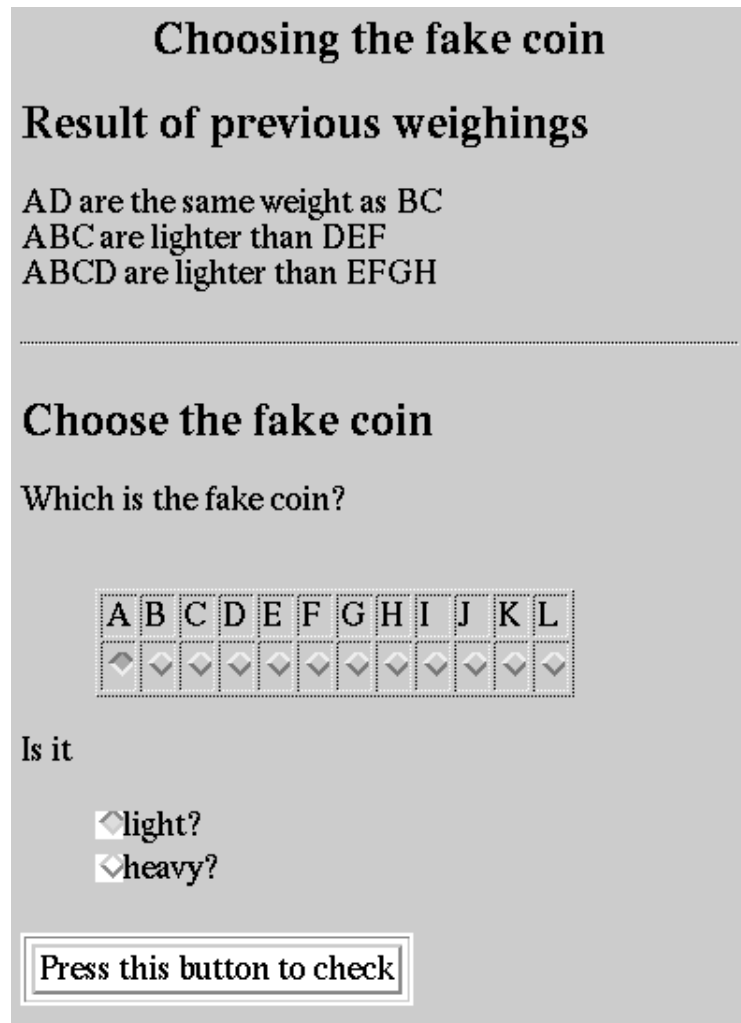
Result of previous weighings

ABC are lighter than DEF
ABCD are lighter than EFGH

Choose coins to weigh

	A	B	C	D	E	F	G	H	I	J	K	L
Coins on the left scale	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼
Coins on the right scale	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼
Coins omitted	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆

Press this button to weigh



Corresponding to each window will be a command, which edits the input and applies it to the weighing game. The weighing window facade is

WeighingWindow
coins on left scale coins on right scale previous weighings
display result of weighings display edit error generate next window (weighing or choice)

The execute method of the WeighCommand is

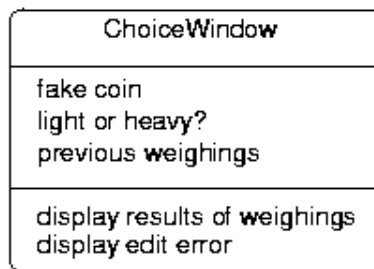
```
get weighings from the weighing window
```

```

ask weigh game edit weighings
if weighings are OK
    apply weighings to weigh game
    get result from weigh game
    inform weighing window of result
else
    get edit error from weigh game
    inform weighing window of edit error
end
ask weigh window to generate next window

```

The choice window facade is



The execute method of the ChooseCommand is

```

get choice (coin + weight) from the choice window
ask weigh game edit choice
if choice is OK
    apply choice to weigh game
    get result from weigh game
    inform choice window of result
else
    get edit error from weigh game
    inform choice window of edit error
end
ask weigh window to generate next window

```

The weighing game can be played at <http://willow.canberra.edu.au/~davidc/weigh.html>

Our second example is taken from a more traditional business application in a banking domain. We illustrate the technique with a deposit command which links a transaction window to the bank. The transaction window is shown below.

Transactions on sav001

Current balance = \$111

Choose transaction type

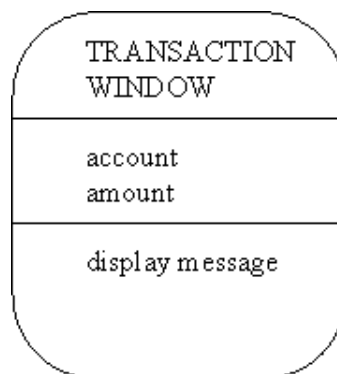
Deposit

Withdraw

Amount of transaction

Submit your transaction

The transaction window facade is simply



The execute method of the withdraw command is

```

get account and amount from the transaction window
ask bank if amount can be withdrawn from account
if amount can be withdrawn
    request withdrawal from bank
    inform transaction window of withdrawal
else
    get reason for not being able to withdraw from bank
    inform transaction window of reason
end
  
```

A simple banking application using this approach has been written (in Eiffel) with GUI and TUI interfaces. The source code can be downloaded from <http://willow.canberra.edu.au/~davidc/banking.zip>

6. Automating the change between media?

When Gamma, et al [Gamma+95] describe the Abstract Factory pattern they specifically address changing between different look-and-feel standards such as Motif and Presentation Manager. They further claim (p 89) that "It makes exchanging product families easy." This gives rise to the question of automating the change between media. This is an attractive proposition, but we do not believe that it is practical due to different characteristics of the media. It is certainly possible to have a set of abstract classes which can be implemented in various media. For example, there could be a common parent of menu, submit buttons and console menu. However this would be a minimal set of user interface classes and not include components such a scroll bars that are specific to one or two media, it would not address the problem of GUI and WUI being non-modal whilst TUI and PUI are modal, and the event loops in Windows and HTML Forms are quite different to the simulated event loops in TUI and PUI windows.

7. Using the Technique in Teaching Object Oriented design

Both the Command pattern and the Facade pattern are readily understood by students and they do not find any difficulty in implementing them. A teacher, however, aims for understanding on the part of the students and this is where use of these patterns opens opportunities for learning. One of the authors takes a 3rd year undergraduate elective in OO design. Students develop and implement a system using text based HIC in the first half of the semester. (Application domains have included a library and a hire firm.) In the second half of the semester students replace the text based HIC with a Web based interface. This gives students first hand experience that loose coupling really does enhance the reusability and maintainability of their software. It also helps in teaching reuse. Firstly, because both systems are in the same problem domain, students reuse classes they have written themselves and they learn from their mistakes and from the feedback they are given on their first system. Secondly the text based HIC classes (console menu, console pick list, etc) are developed in lectures, in the context of developing a banking system. This means that students reuse classes they have seen developed and know the design decisions made during development. And thirdly, the students are supplied with web based classes to help in writing cgi scripts and generating HTML documents. This means that they have to understand and reuse a toolkit of co-operating classes - the one used being EiffelWeb, from Interactive Software Engineering [EiffelWeb]. This approach to teaching was presented by Clark at the 21st Australasian Computer Science Conference [Clark98]. He reports a significant improvement in the quality of students' assignments when the approach was adopted.

8. Conclusion

In this paper we have shown how applications can be written so that the medium of the user interface can be easily changed. This has been achieved through an architecture which loosely couples the user interface with the rest of the application, and employs both the Command pattern and the Facade pattern. We have also shown how the adoption of this architecture and different interface media can enhance learning by students of O/O.

9. References

- [Clark98] Clark, D. I., (1998) Using the World Wide Web to teach Object Oriented Design, Proceedings of the 21st Australasian Computer Science Conference (ACSC'98), Perth, February, 1998, 357-365.
- [Coad+93] Coad, P. and Nicola, J, (1993) Object-Oriented Design, Prentice-Hall.
- [EiffelWeb] Interactive Software Engineering, <http://www.eiffel.com/>

[Gamma+95] Gamma, E., Helm, R., Johnson, R., Vlissides, J.,(1995) Design Patterns: Elements of Reusable Object Software, Addison-Wesley
[Fayad+97] Fayad, M. E. and Schmidt, D. E, editors, Object-Oriented Application Frameworks, Comm ACM 32-38, October, 1997
[IBM 87] IBM (1987) System Application Architecture: Common User Access Panel Design and User Interaction
Last modified: Tue Jul 13 14:18:23 EST 1999