CIAO: A Powerful Synchronisation and Controlling Mechanism for Human Computer Interfaces

Pericles Papageorgiou
Department of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road, London E1 4NS
United Kingdom

Abstract
This paper demonstrates how an object oriented structural and run-time toolkit architecture, called CIAO, is used to derive a framework for handling user interface dynamics. The approach exploits the object paradigm to build a powerful user interface controlling component in two phases. First, a structural description of the application domain is obtained and then incorporated into dialogue control. This provides the means for relating graphical object behaviour to application functionality. Secondly, dialogue control is further polished to provide a representation describing the correlation of behaviour between graphical objects.

Using CIAO, interactive applications are built from four basic component types. The Input and Output that define the virtual device level, the Application that defines the problem domain and the Coordinator that defines the mapping between the virtual devices and the application.

1. Introduction
The rapid advances in computer hardware technology of high speed graphics workstations and desktop personal computers have set the grounds for the proliferation of new ways of programming or interacting with computers. These come in the form of direct manipulation user interfaces. The use of direct manipulation is the prominent ingredient of graphic simulation, teaching, visual programming, and program visualization environments, program animators and debuggers, and high quality graphics environments such as those provided with the Macintosh computer or the Smalltalk programming system.

The research described here, has identified that the behaviour of graphical objects on the screen is governed by two, equally important, factors. First, graphical objects are related to and represent some form of behaviour or characteristic of the underlying application domain (e.g., the use of a slider to increase the value of a counter application). This form of correlation is called an application relationship. Secondly, there is a relation between each graphical object's behaviour and that of other graphical objects within the interface (e.g., the use of a slider to increase the charge on an electromagnet attracting a metallic ball). Such an interdependence of behaviour is called a spatial relationship. Unfortunately, the plethora of user interface toolkits [2], [4], [5], [10], for fabricating direct manipulation user interfaces, provide inadequate dialogue synchronisation and controlling mechanisms to handle these two forms of relationship.

This paper demonstrates how an object oriented structural and run-time toolkit architecture, called Coordinator Input Application Output (CIAO) is constructed and used to represent the application and spatial relationships of user interface objects. The architecture starts with a high level decomposition of interactive software into four cooperating data abstractions. These are: the Coordinator that handles the interaction syntax and the binding of semantics between the application domain and the user interface. The Input that handles a part of the lexical level operations. The Application that describes the interaction semantics. The Output that portrays the results of interaction. The design of CIAO does not stop here as user interface toolkits do. Instead, it delivers a framework for representing application and spatial relationships of interactive objects by performing a structural analysis to each abstraction, and in particular the application. In addition, CIAO is a concurrent architecture that models each of its constituent abstractions around the concurrent event processor paradigm. Concurrent event processors are event based interactive application component units (e.g., an input handler, an object representing the semantics of a calculator, an output handler) that can be used as the building blocks for a variety of interactive applications.

Section 2 illustrates the primary properties provided to interactive applications by CIAO. Section 3 describes how an interactive application is constructed with CIAO by using a graphic simulation example. Section 4 discusses issues pertaining to the implementation of CIAO. Finally, section 5 outlines the conclusions gathered from the CIAO research alongside an evaluation of its superiority as compared to existing user interface toolkits.
2. The CIAO Structural and Run-time Architecture

The design of CIAO views application state as the smallest unit around which behaviour can be built. The user interface is an agent that manipulates and portrays the state of the application. Under such a perspective, CIAO decomposes the application into smaller state structures and defines how they interact with the user interface. A structural description of the obtained decomposition is then incorporated within the dialogue controlling component. This forms the basis on which a powerful dialogue controlling component can be built to handle and represent application and spatial relationships of graphical user interface objects.

CIAO also makes use of concurrent processes communicating through event channels to enable the fast routing of input requests in an application independent way. CIAO yields an intrinsic separation of interactive applications by isolating semantic, syntactic and lexical level details into four independent data abstractions. Each data abstraction forms a concurrent event processor (CEP).

A CEP describes a portion of user interface functionality. It encapsulates within object boundaries aspects such as dialogue sequence, input action semantic binding, interaction styles and techniques including input and output handling and application layout. In this way, low level user interface components may be enriched with high level functionality (e.g., help, undo, error recovery, etc.). The CEP metaphor can be used to build libraries of specialised reusable objects (e.g., a mouse handler, a numeric displayer) that can be mixed and matched to satisfy the needs of the interactive application.

A CEP is an event handling object. It declares interest in a series of event types and uses an event monitoring (lightweight) process to transmit or receive events to or from other CEP objects. In this way, an event based communication channel is established between the four CEPs that comprise CIAO. On receipt of an event, the appropriate event response is initiated.

By closely conforming to the principle of access oriented programming [11], CEPs are active objects. They can be programmed to react to accesses of their state by performing an operation as a side effect. This is achieved by accessing their state with the aid of suitable state messages [6]. State messages introduce a level of abstraction within the CEP object itself. From within its own methods its state is not directly accessible.

By combining four appropriate CEPs to form a CIAO structure, an interactive application is constructed as shown in figure 1. CIAO structures constructed in this way can then be used as ready made reusable agents (e.g., a dialogue facility, a menu, a slider, etc.) and combined into a larger interactive application should the designer wish.

2.1. The Input CEP

Input is the mediator between the user and the rest of the interface. It is used to convey input event tokens, in the form of new events, to the coordinator, where they are syntactically checked and given a semantic meaning. Input event tokens are validated for lexical accuracy first. The tokens that an input CEP handles, are generated by the user when interacting with a particular display region. By its nature, input is kept as simple as possible; merely transferring input event tokens to its associated coordinator.

2.2. The Output CEP

Output is the medium through which the results of user input requests take a graphical representation. It exchanges events with its associated coordinator, to maintain its own state and display. This communication takes place by using a general event based protocol (e.g., event to update its display, event to get data from the coordinator) in a "pluggable" [3], [5] way. Therefore the output CEP can be reused in a variety of similar interactive applications.

Output CEP objects can be arranged into visual hierarchies in the sense of having an output as the child of another. The hierarchical structure of output CEPs is similar to that found in contemporary window managers such as those provided with Smalltalk [3], NeWS [12], [8] and X [9]. The fundamental difference with the Smalltalk window manager is that event interest matching is targeted directly towards the interested output CEP, as opposed to routed through the output object visual hierarchy.

2.3. The Application CEP

The application describes the problem domain by encapsulating all of its state and local properties within object boundaries. It represents the semantic layer of the interac-
tive application where input requests take an executable form. The application utilizes the event based nature of CEP objects to ensure its separation from the user interface. It extensively uses state messages to transmit events notifying, whichever CEP object is interested in receiving events from it, about its state transitions. Interested CEPs can then serve their needs by querying this object to find out exactly what has changed in its state.

2.3.1 Structural Analysis of the Application

In object oriented languages, a composite object is recursively built from other, possibly simpler, objects. This recursive definition of an object in terms of its constituent objects is of increased importance to application CEPs. The constituent objects are often referred to as parts [1] of the overall object — the whole.

A whole may have either explicit or implicit parts. Explicit parts represent the concrete structure of the whole. Implicit parts are usually derived from two or more explicit parts. These can be utilised by the whole to pretend it has components that are really not members of its structure. The state of the whole can be described by its explicit and implicit parts. Explicit parts can be simple or composite. Composites can be recursively decomposed further into sub-parts. Consequently, composites are viewed as wholes of their constituent parts. In this way, a whole may be structurally described as a hierarchy of its enclosed parts — the explicit part hierarchy. Figure 2 depicts the part hierarchy associated with a whole that has a simple and a composite part.

Figure 2: Explicit Part Hierarchy within Application

Composite parts provide the necessary state messages for accessing their sub-parts. State messages can be used to simulate object encapsulators [7] by performing an operation, as a side effect, before and after a part access, as shown in figure 3. At any level in the hierarchy, a composite part can enrich the state messages associated with its parts with pre and post access operations.

In the context of the application CEP, the mechanism of state messages is used to provide access to its explicit and implicit parts. Some parts of the application CEP may be protected and treated as private by not providing state messages to access them. For each state message, the pre and post access operations take the form of event transmissions aimed to notify other interested CEPs, about state changes happening within the application.

2.4. The Coordinator CEP

The coordinator is the central controlling user interface component. It provides the means for synchronizing and maintaining consistency between the semantics of the user interface and the underlying application. It is a kind of interaction binder that ensures separation of application semantics from their visual portrayal. It is also used as an interfacing medium for combining different interactive applications.

To fulfill its role as a controller, the coordinator is explicitly aware of the existence and structure of the other three CEPs. It is loosely coupled with them, though, by taking full advantage of the event based communication channel architecture. It also performs syntactic and partial semantic checks to input requests. Finally, in interactive applications that are made of multiple CIAO structures, it acts as a means for communication to other CIAO structures through the medium of either directed or broadcast event transmission. Multiple CIAO structures are arranged into a hierarchy of communicating coordinators. The way in which these structures are placed within the hierarchy provides the means for representing the spatial relationship between the graphical objects associated with a child structure and those of the parent structure.

2.4.1 Coordinator Hierarchies

The coordinator has a hierarchical structure that closely resembles the part hierarchy of the application. Each node in the part hierarchy of the application is associated, via event linkage, to a node in the coordinator hierarchy. Coordinator nodes are responsible for interfacing their associated part of the application with the rest of the user interface. To do so, each coordinator node is interested in the state of an explicit or implicit part of the application. The coordinator hierarchy is constructed in the following way. The node at the top is associated with the application as a whole. Other coordinator nodes are inserted as children of the top coordinator, each associated to a simple or compo-
site (explicit) part of the application. Nodes within the coordinator hierarchy communicate by using an event based channel. Those dealing with simple parts, are associated through their parent with the appropriate part. Alternatively, nodes dealing with composite parts, are associated, via event linkage, directly with the appropriate part. Nodes within the coordinator hierarchy can be associated with multiple inputs and outputs, so that multiple interaction techniques and displays can be provided to a single part of the application.

As an additional advantage, coordinator hierarchies allow the structural representation of application implicit parts, which is not catered for by the application part hierarchy. Implicit parts are usually derived by one or more other parts. Coordinator nodes can be inserted in the coordinator hierarchy to echo the dependence of implicit parts on other parts. These have as many parents as needed to reflect the required relationship between parts.

3. Graphic Simulation Example

Consider a direct manipulation graphic simulation for teaching the laws of physics as that shown in figure 4. This example depicts the effect of a magnetic field, produced by varying the charge of an electromagnet, on a metallic ball attached to the end of an elastic spring. As the voltage of the electromagnet is increased with the slider, its associated charge is raised. The intensity of the magnetic field becomes stronger and the ball is attracted further down towards the electromagnet. In this example, the laws of physics are presented in a simplified form to enable concentration on user interface synchronisation and controlling issues. There are four graphical objects involved in this example. The spring and ball, the distance separating the ball and electromagnet, the electromagnet itself, and the slider for varying the voltage of the electromagnet. All four have to be synchronised and controlled for the example to be successful. Note that the input component associated with the slider, although not shown, can be thought as if it is overlapping the shape.

This example comprehends three application objects. They describe the behaviour of the ball and spring, the magnetic field, and the electromagnet's voltage and charge. Variation of the electromagnet's voltage and charge affects the intensity of the magnetic field and consequently the displacement of the spring. For this reason the CIAO structures, associated to these applications, are hierarchically arranged as shown in figure 5 to exemplify the spatial relationship between the graphical objects. The structure of the hierarchy tolerates complex permutations of the above example. For instance, the specification of this example may be slightly changed and allow the electromagnet to be moved, with the mouse, closer to or further away from the ball. The impact of such an event on the distance and consequently on the spring is automatically
Figure 5: Modelling the Graphic Simulation with CIAO

The application associated with the spring and ball consists of three simple parts. The elasticity and length of the spring and the mass of the ball. This sort of structure is portrayed by the coordinator construct depicted in figure 6. The graphical display of the spring and ball is dependent on the three parts of the application. Any changes to their state have to be synchronised and portrayed on the display. For this reason, a forth coordinator node, labelled spring & ball, is added to the coordinator hierarchy. It controls the relationship between the spring and ball graphical object and the spring application.

Figure 6: Coordinator Hierarchy for Spring & Ball

The application related to the magnetic field consists of two composite parts. The start and end points of the line separating the ball from the electromagnet. In addition, the magnetic field application is able to calculate the intensity of the field close to the ball when delivered with the electromagnet's charge. The coordinator associated with the structure of this application is shown in figure 7. Because the start and end points are composite objects, their internal structure is further decomposed and represented by the two pairs of x and y nodes in the coordinator hierarchy. For the purposes of the graphic simulation, it is necessary to calculate the distance between the two end points. The distance is an implicit part of the magnetic field application. It is derived from the x and y coordinates of the two end points. This relationship is represented by the coordinator node labelled distance in figure 7.

Finally, two simple parts dwell within the application modelling the electromagnet. These are the voltage and charge of the electromagnet. The relationship between application objects and their associated graphical objects, is maintained by a coordinator structure as that rendered in figure 8. Consider now how these coordinators communicate and synchronise interactive application behaviour at run-time.

Assume that the value of the slider is increased as shown in figure 4b. While the slider is still being dragged, its associated input component generates an event to notify the voltage coordinator in figure 8. The event contains the value by which the voltage of the electromagnet is to be increased. The voltage coordinator uses its parent link to send an event, containing the appropriate state message and its parameters, to its application to adjust its value.
change in the voltage probably will cause the charge of the electromagnet to be updated. The implementation of the state message within the electromagnet application carries two side effects. First, the charge of the electromagnet is also computed and secondly, two new events are generated stating that both the charge and voltage of the electromagnet have changed. Both events are captured by the electromagnet coordinator and simultaneously propagated to its two children. These in their turn obtain the new values of their associated parts and decide whether an update of their respective output objects is required.

Next, the electromagnet coordinator generates a new event containing the updated value of the charge. This event is captured by the magnetic field coordinator, in figure 7, which informs its associated application to compute the new intensity of the magnetic field at the edge of the distance line. Once this is done, the field coordinator generates a new event containing the newly computed magnetic field intensity. This event is captured by the spring coordinator, shown in figure 6. As a result, the application is requested to compute the spring displacement given the new field intensity. The length of the spring is the part within the application that is adjusted to a new value. When this is done, the spring coordinator is sent an event, by the spring application, notifying the change in the length of the spring. This event is propagated to the length coordinator, which obtains through its parent link the new value for the spring length. Consequently, the spring and ball coordinator is informed to adjust the presentation of its associated output. Also, the spring coordinator generates another event notifying the field coordinator that the ball has come closer to the electromagnet. It in its turn informs the end point coordinator to contact the end point within the application and adjust its value. Finally, the new value of the end point is propagated to the distance coordinator which computes the new distance value and updates its associated output object.

4. Implementation

The CIAO structural and run-time architecture has already been implemented under Smalltalk-80. This takes the form of supporting classes, each modelling one of CIAO's components. In addition, an event based mechanism (similar to that of NeWS) has been incorporated within the Smalltalk-80 programming environment. Basic dialogue objects and interaction tools structured around CIAO have also been constructed (e.g., push buttons, menus, dialogue boxes, notifiers, etc.). Currently, the implementation effort is directed towards the construction of object editing tools that would assist the automatic creation and editing of coordinator hierarchies. Given an application object, these tools would enable the graphical construction of a large proportion of user interfaces structured around CIAO. The exact implementation details of CIAO will be presented in a forthcoming paper.

The design of CIAO has been tested with real interactive applications. The implementation of the graphic simulation example presented in the previous section is largely completed. Experience gained so far has proved the design of CIAO to be successful.

5. Conclusions & Evaluation

The research presented in this paper, starts with a high level decomposition of interactive applications and continues to describe a framework for analysing each user interface component. The main emphasis is given in fabri-
cating a powerful dialogue controlling component that links the application with the user interface. The approach presented here, offers the following advantages to user interface design.

- Spatial and application relationships between graphical objects can be easily and efficiently handled by suitably arranging the coordinator nodes within the coordinator object hierarchy to reflect the needed relationships (e.g., the arrangement of coordinator nodes in figure 5, the coordinator hierarchy dealing with the distance between the ball and electromagnet in figure 7).
- “Fine-grain” fast feedback can be provided in response to user requests. The coordinators that are associated with the affected parts are only involved to service the request (e.g., the end point coordinator in figure 7 is only involved to adjust the length of the distance line). This comes in contrast to PAC [2] and COA [4] where the traversal of unnecessary and possibly complex controlling sub-hierarchies cannot be avoided.
- “Multi-view” interfaces can be provided to an application with minimum effort. The structure of the coordinator hierarchy automatically synchronises the output objects involved, for any type of application. In contrast, PAC, MVC [5], MoDE [10], and COA require special purpose synchronisation mechanisms which sometimes violate software modularity.
- There is a large scale distribution and handling of application semantics among the coordinator nodes. Such fine-grained distribution of semantics reinforces separation of the application from its interface and is not supported by object oriented toolkits. For example, application semantics dealt by the field coordinator in figure 7 and its multiple children, would be usually handled by a single controller.
- Coordinator hierarchies can form the basis for implementing complex command undo and redo, macro recording and playback, and context sensitive help mechanisms. In contrast, object oriented toolkits provide no guidance with respect to such operations, which are usually regarded as indispensable features of direct manipulation user interfaces.

Acknowledgements

I am indebted to my thesis supervisor Mr. Mel Slater for his invaluable comments on early drafts of this paper and effort in helping me polish my ideas. This work was funded in part by the Onassis Public Benefit Foundation under research grant GROUP K-72/1988 - 1990.

References


