

EC3301 – Information Systems Design

Lecture 2 – Interface Design



1. The User Interface

Interaction between the user and the computer system takes the form of a dialogue. In modern systems this dialogue is via a set of graphical user interfaces.

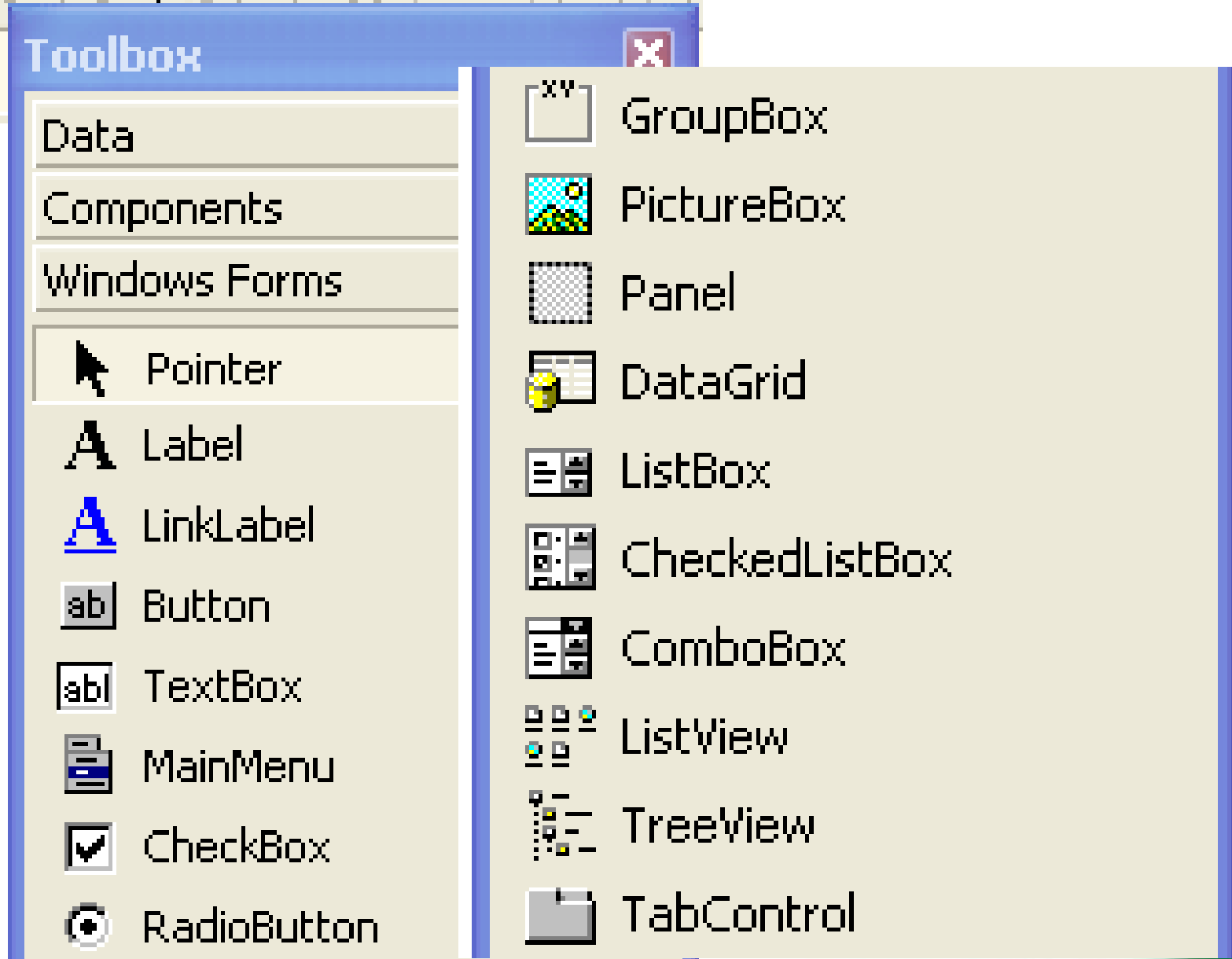
Graphical objects are displayed on the screen and the window management part of the operating system responds to events.

Such interfaces are **event-driven**, and are generated when the user:

- clicks on a button or menu-item
- types a character
- presses a function key
- holds down a mouse button



A selection of graphical components for building GUIs in a Visual Studio environment.



Secondary Tasks – concerned with using the system

- Read and Interpret information that instructs then how to use the system
- Issue commands to the system to indicate what they want to do
- Enter words and numbers into the system to provide it with data to work with
- Read and interpret the results that are produced by the system, either on screen or as a printed report
- Respond to and correct errors



Primary tasks ?

- The primary tasks for example are to take customer order.
- If the system has been designed well, the secondary, system-related tasks will be easy to carry out.



Types of messages in human-computer dialogue

Output	Prompt	Request for user input
	Data	Data from application following user request
	Status	Acknowledgement that something has happened
	Error	Processing cannot continue
	Help	Additional information to user
Input	Control	User directs which way dialogue will proceed
	Data	Data supplied by user



Types of messages in human-computer dialogue

Output	Prompt	Request for user input and labels for automatically generated data, shown in bold, for example Customer code
	Data	Automatic display of Order Date and Next Order No, automatic calculation of totals and tax
	Status	Screen heading; could include display to confirm that a new order has been saved
	Error	Messages to warn of incorrect data entered, for example if Customer code is entered that does not exist or if a negative quantity is entered
	Help	Additional information to user in response to the user pressing F1; may be general about the order entry screen or context sensitive – specific to a particular type of data entry
Input	Control	Use of function keys to control dialogue
	Data	Numbers, codes and quantities typed in by user.



There are a number of characteristics of good GUI design.

- **Consistency** - users expect the same interface to behave consistently in different contexts. This applies to commands, dates, layout of screens And the way the information is coded by the use of colour or highlighting.

The Windows Interface Guidelines for Software Design (Microsoft 1997)
Macintosh Human Interface Guidelines (Apple 1996) - both lay down design standards for GUIs. Hence many different applications have a consistent interface.

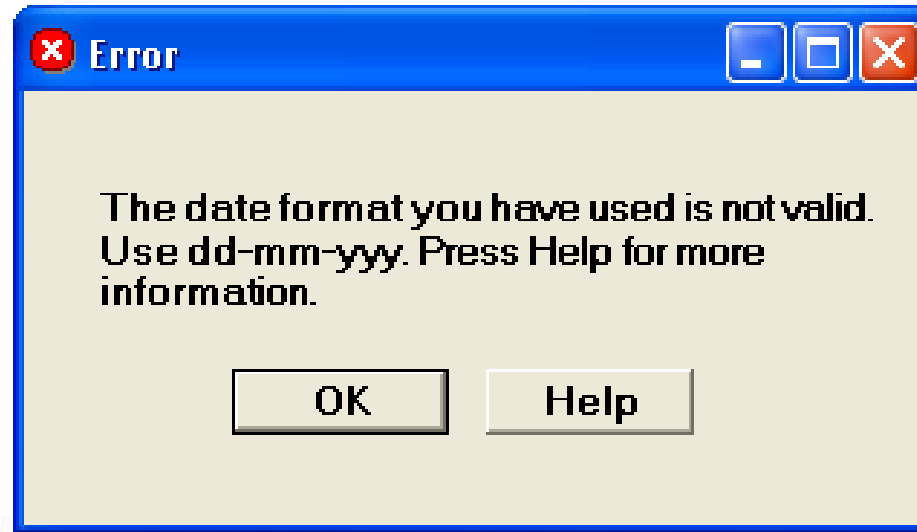
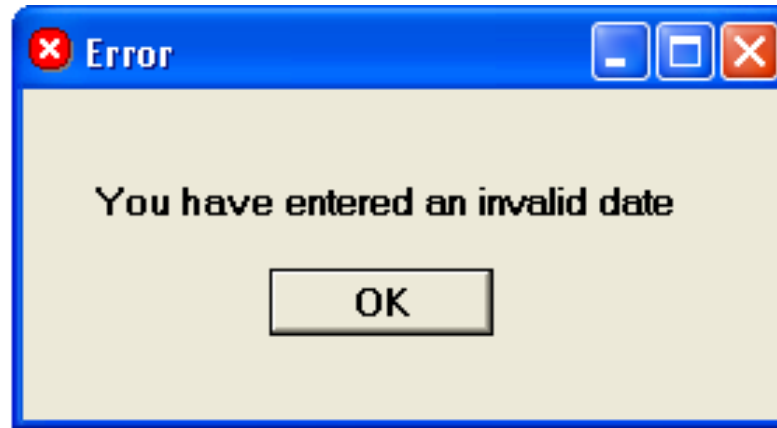
- **Appropriate User Support** (when a user does not know what action to
- Take or has made an error
 - context sensitive help messages
 - diagnostic information
 - recovery from error



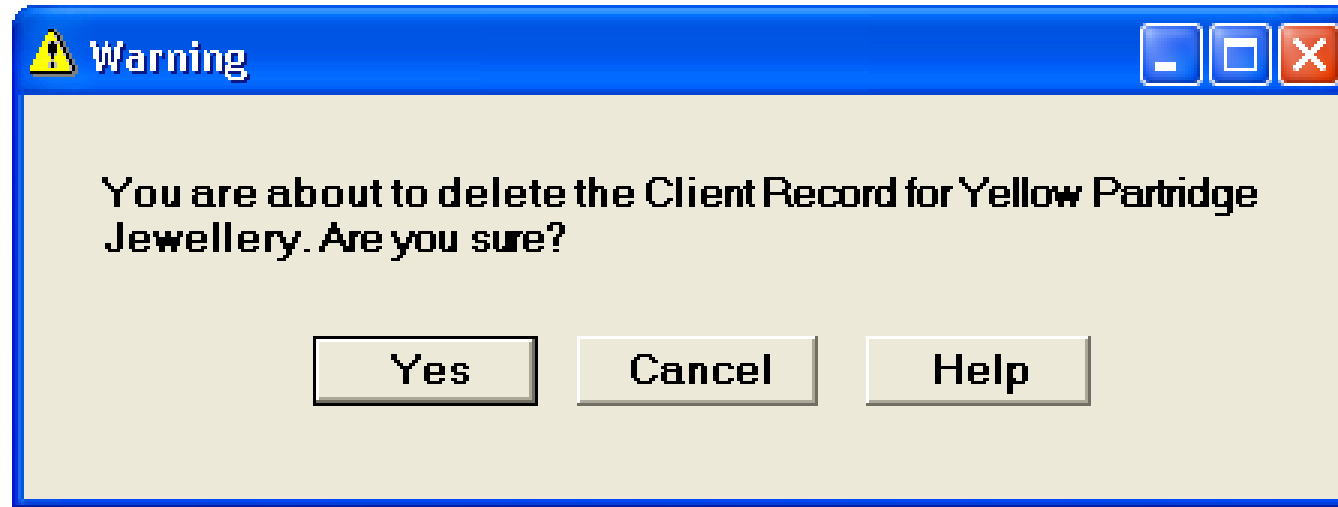
error messages

warning messages

Comment on the following error messages.



Comment upon the following warning message.



- **Adequate Feedback from the System** - users expect the system to respond when they make such action.

For example, on the command to load a file a suitable icon should show that a file is being loaded with a progress bar, and not allow the screen to become inactive.



Can you name other feedback mechanisms?

The purpose behind feedback is to reduce the user's uncertainty about whether the system has received the input and is doing something about it or just waiting for the next input.

- **Minimal User Input** - users resent making what they see as unnecessary keypresses and mouse clicks. Reducing unnecessary input cuts down the risk of errors and speeds up data entry.



The interface should be designed to minimise the amount of input from the user by:

- Using codes and abbreviations.
- Selecting from a list rather than having to enter a value.
- Editing incorrect values or commands rather than having to type them again.
- Not having to enter information that can be derived automatically.
- Using default values.



3. Approaches to User Interface Design

Regardless of the approach ([UID design factors](#)) used they will all contain three main steps:

- **requirements gathering** – determine the characteristics of the user population; types of user, frequency of use, discretion about use, experience of the task, level of training, experience of computer systems.
- **design of the interface** – (a) allocate elements of task to user or system; determine communication requirements between users and system.
(b) design elements of the interface to support the communication between users and system.
- **interface evaluation** – (a) develop prototypes of interface designs
(b) test prototypes with users to determine if objectives are met.



Approaches are:

- Structured – based on a model of the systems development life cycle which is broken down into stages -> steps -> tasks.

Claims –

better project management;
standards for diagrams and documentation;
improved quality of delivered systems.

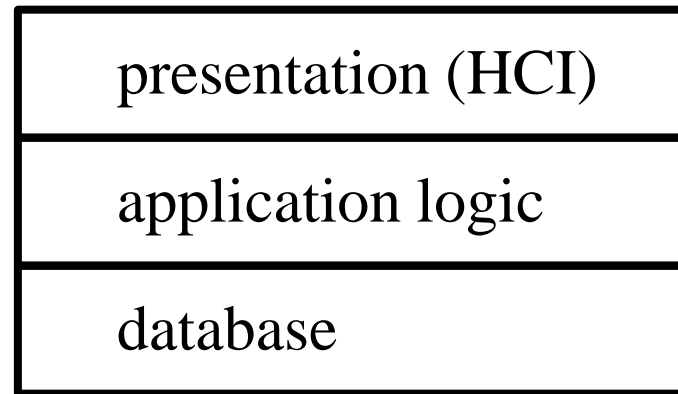


- Ethnographic – interface designer spends time with the users immersed in their everyday working life in an attempt to understand and document the real requirements of the users.
- Scenario-based – step-by-step descriptions of the user's actions captured through textual narrative, picture-based storey-boards, video mock ups or even prototypes.



4. The architecture of the presentation layer

(a) Layered sub-systems - each layer corresponds to one or more subsystems.

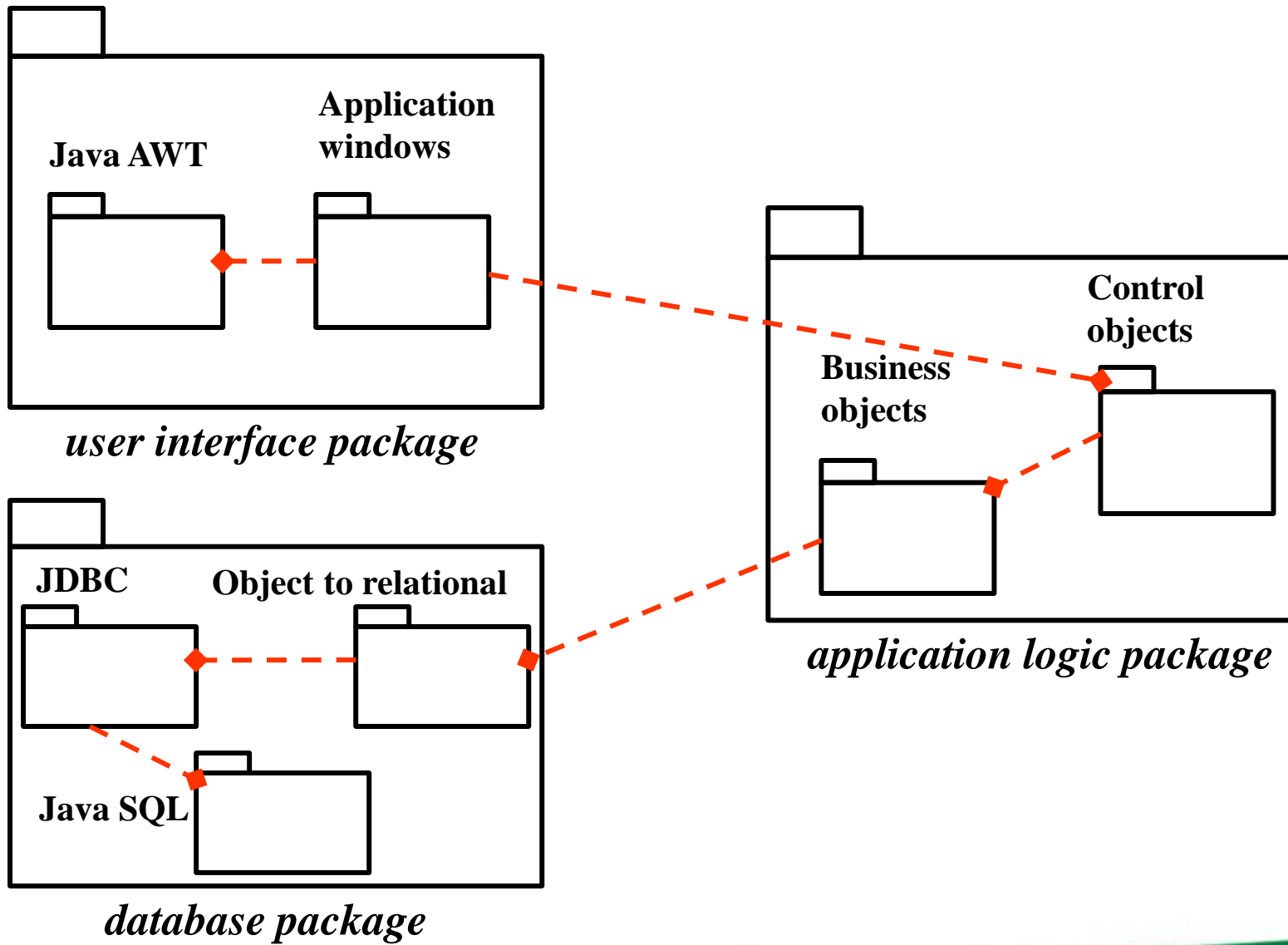


three-layered architecture



- It produces smaller units of development (good for project management and meeting deadlines).
- It helps to maximize reuse at the component level (in other systems).
- It helps the developers to cope with complexity.
- It improves maintainability (change confined to a subsystem - changes do not ripple through other subsystems).
- It aids portability - changes localized to certain subsystems.





The reasons for the separation are:

- **Logical Design**

Functionality of the business classes is developed independently of the hardware and software used to built the system, hence business classes do not include details of how they are displayed.

- **Interface Independence**

It does NOT make sense to add display methods to classes - object attributes can be used in different use cases, sometimes the attributes will be displayed on a screen, at other times on a printer.



- **Reuse**

Classes should be reused in different applications, hence classes should NOT be tied to a particular implementation or to a way of displaying the attributes of their instances.

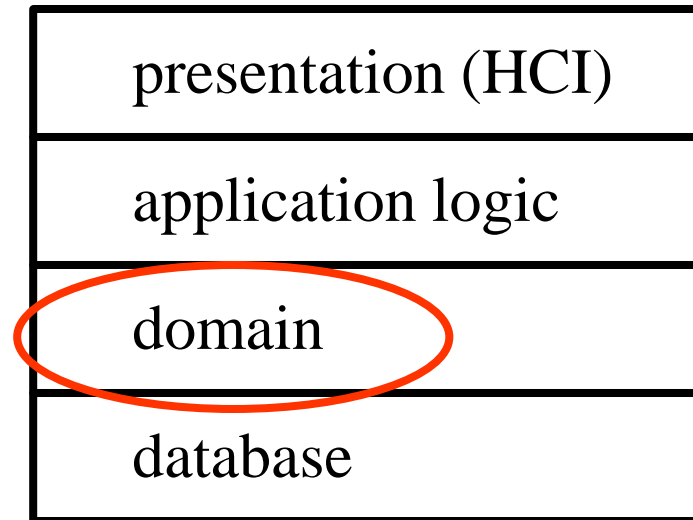
The behaviour of the interface must be kept separate from the behaviour of the classes that provide the main functionality of the system.

Coad and Yourdon 1991 - suggest keeping the Human Interaction Component separate from the Problem Domain Component.

Larmon (1998) advocates the use of a three-tier architecture.



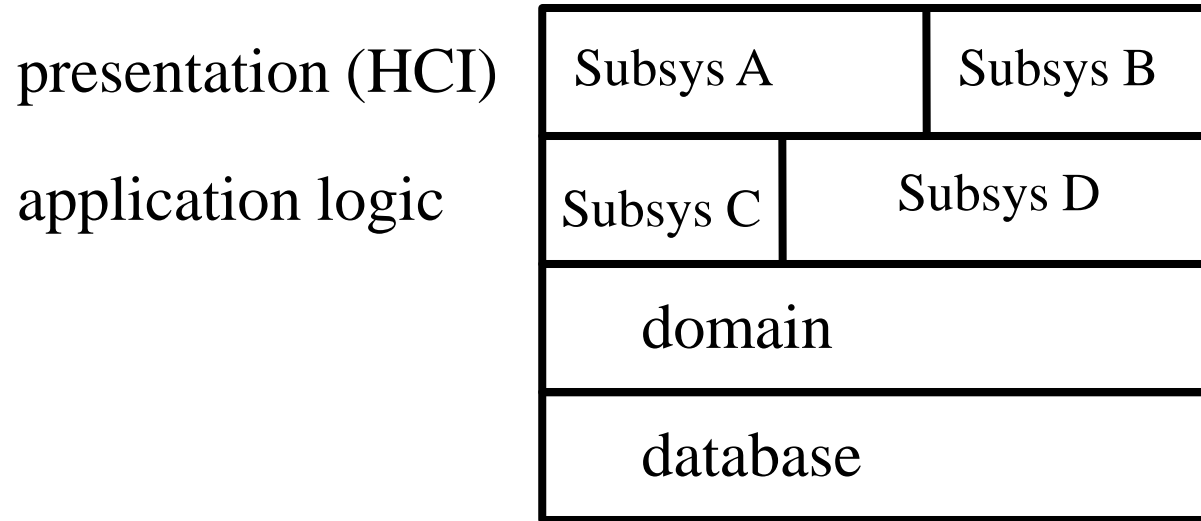
An extra (domain) layer can be inserted to factor out common routines that may be used by the application logic.



four-layered architecture



(b) **Partitioning** - some layers within a layered architecture may need to be decomposed because of their intrinsic complexity.



Subsystems that result from partitioning should have:

- clearly defined boundaries (functionality)
- well specified interfaces
- high-levels of encapsulation



5. AGATE - Case Study Scenario

An advertising agency deals with companies that it calls its clients.

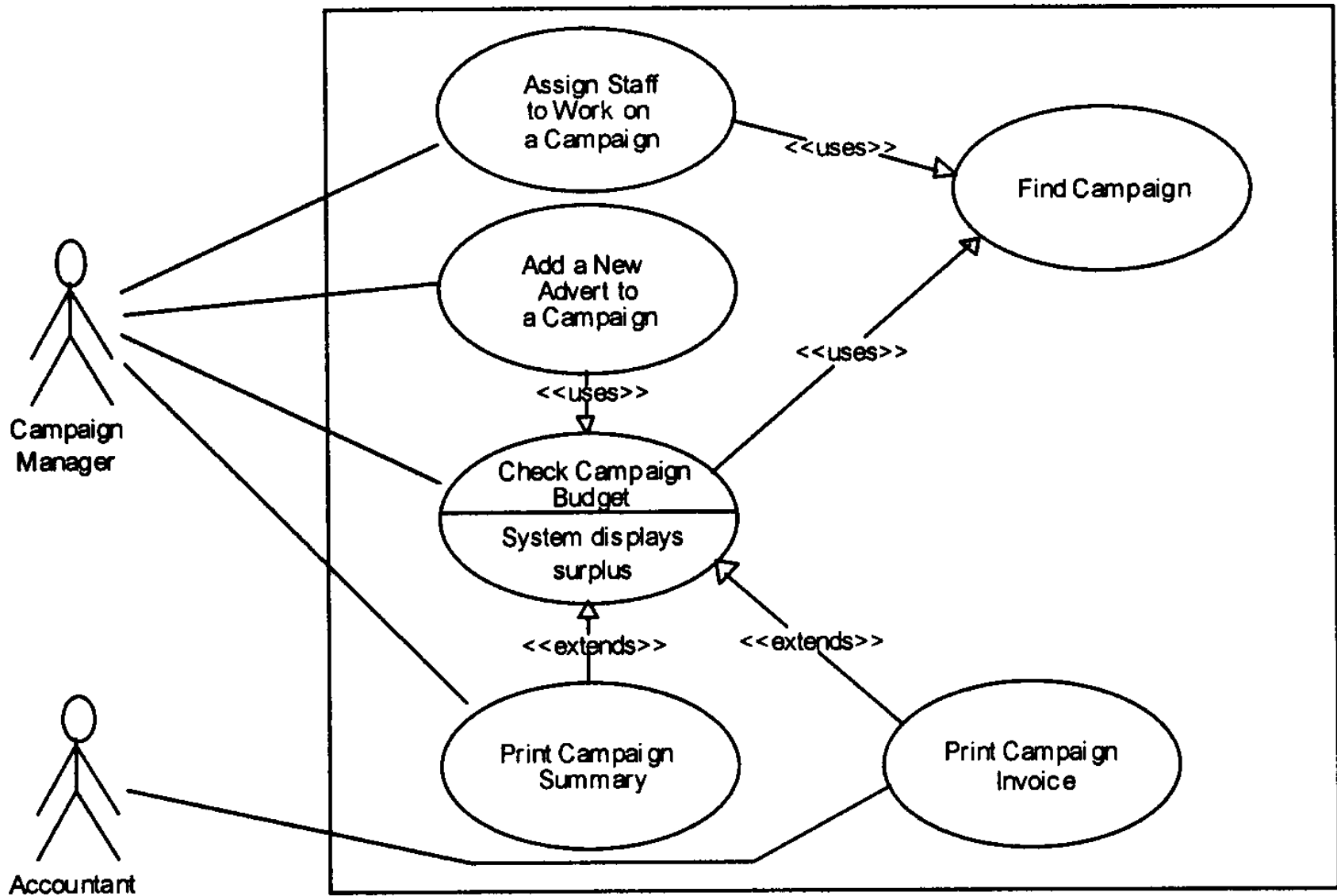
Clients have advertising campaigns, and a record is kept of every campaign.

Every campaign has a campaign manager and a number of staff who work on the advertising.

The campaign manager estimates the likely cost of a campaign.

From this estimate, costs are deducted as they are incurred, the balance is known as the budget surplus (but equally could be a deficit).





As an example of designing the presentation layer, consider the interface for the use case **check campaign budget**.

This requires that we know:

- the name of the client
- the current campaign for that client
- the calculated budget surplus

There will be an object that provides an interface onto the functionality of this use case. This can be the foreground window shown on the next slide.



AGATE

Clients Campaigns Adverts Notes Staff Help

Check Campaign Budget

Client Yellow Partridge

Campaign Fashion Jewellery Magazine

Budget Surplus £2,500.00

Check Close



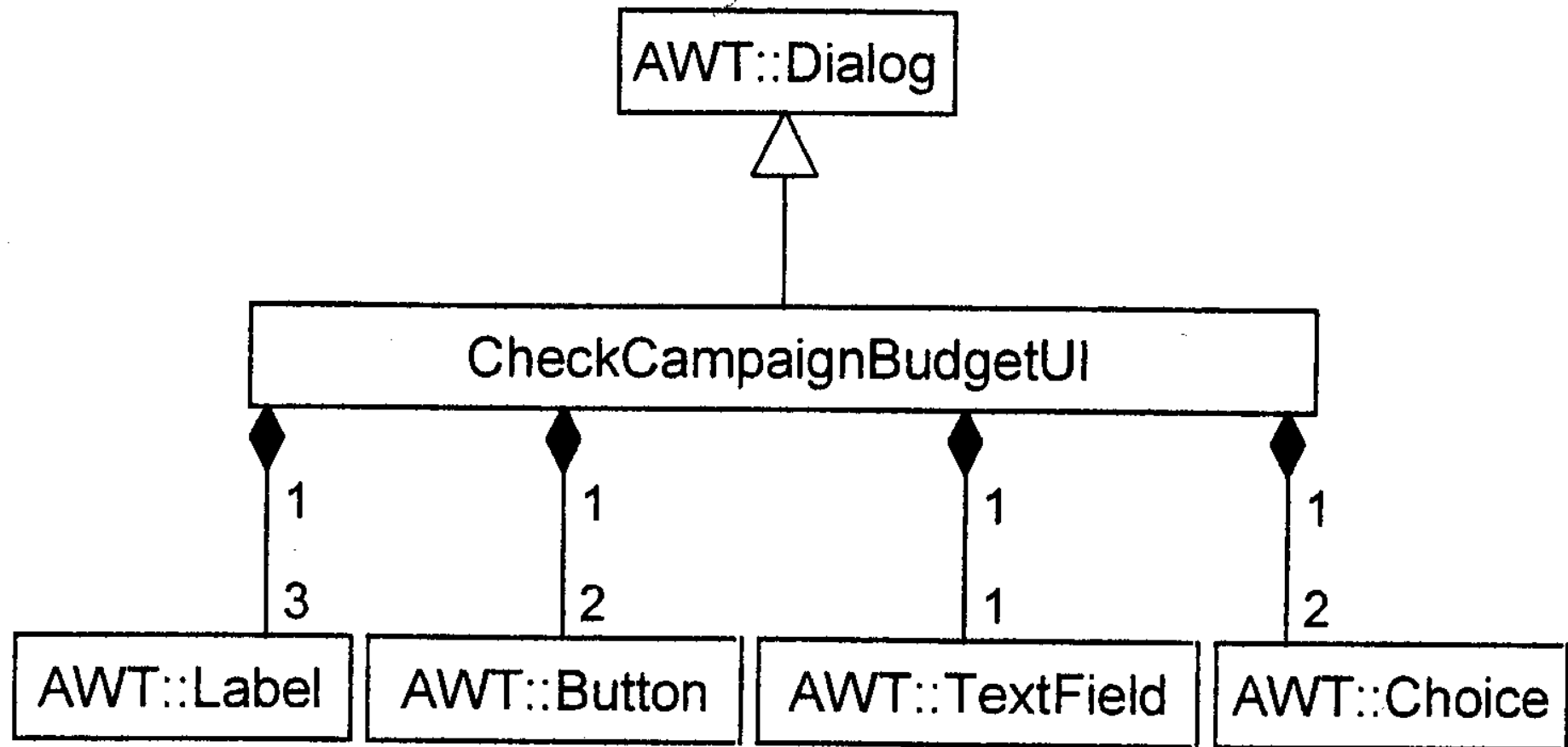
Despite the window containing different component objects it may be regarded as a single object.

In reality, the window may be an instance of a subclass such as Dialog, and may contain a number of components such as:

- buttons
- labels
- listboxes (Choice box in Java)
- and a textbox



Composition of GUI for Client/Campaign/Budget Surplus



Class diagram showing AWT components with their package name.



6. UML Statechart Diagrams

Systems and entities within a system, such as objects, can be viewed as moving from state to state.

An Account object within a banking system can exist in the states of:

- zero balance – when just opened (instantiated)
- in credit – when money paid into the account
- overdrawn – when debited for more money than is in the account



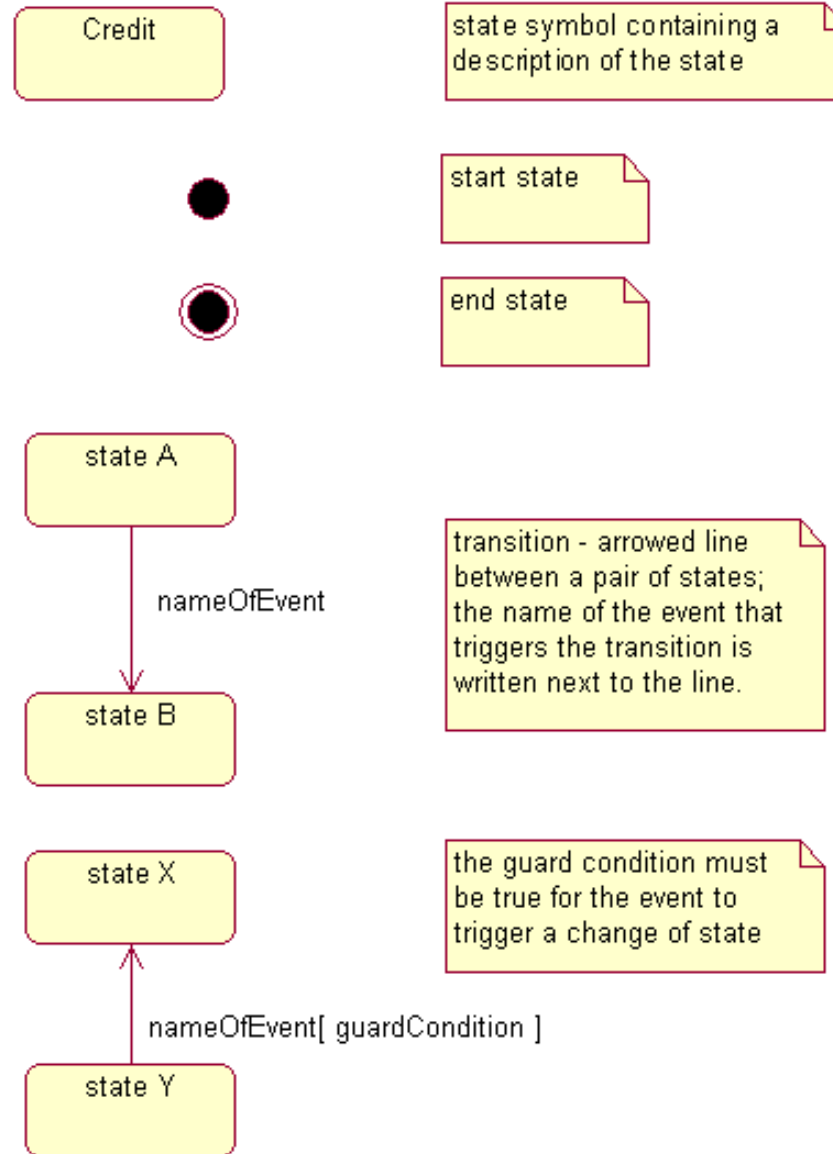
Systems are driven by events, for example in the banking system the event of paying money into an account will cause the account to go into the credit state, the event of withdrawing more money than is in the account will cause the account to go into the overdrawn state.

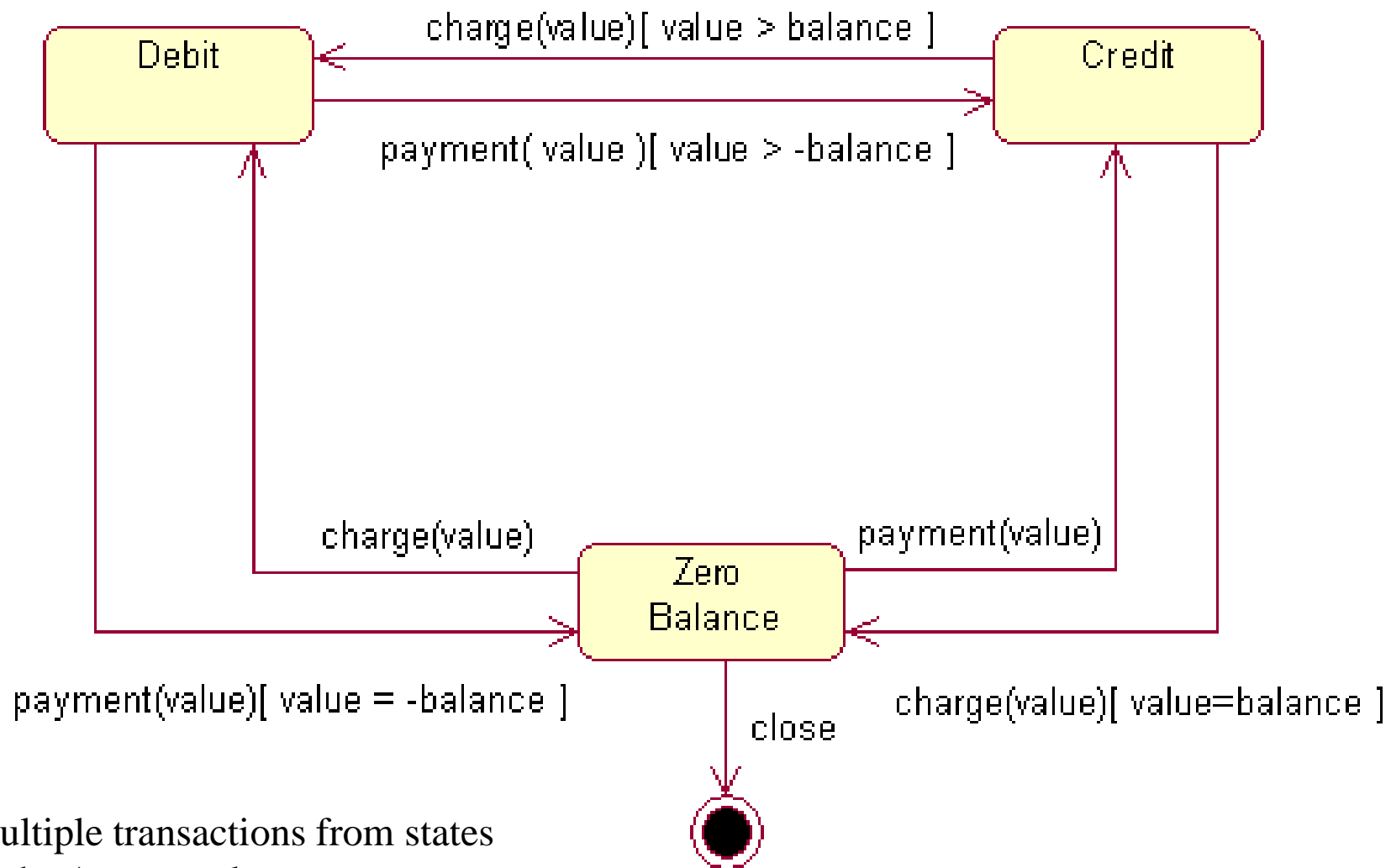
Statechart diagrams are useful for describing:

- complex business entities, such as customers and accounts;
- behaviour of subsystems;
- interactions in boundary classes (GUIs);
- use case realizations;
- complex objects.



Symbols used in Statecharts

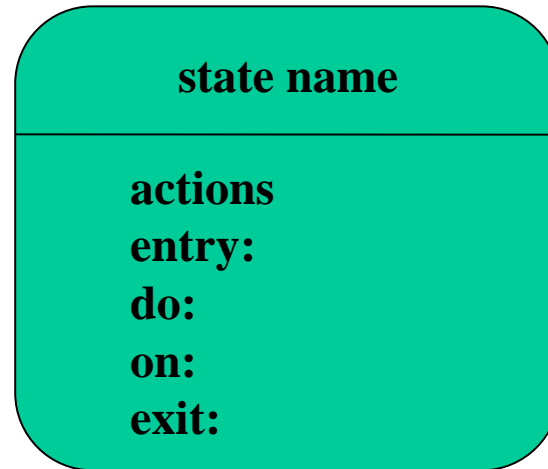




Multiple transactions from states
of the Account class.



A state can trigger actions which may be represented in the statechart diagram as follows.



- **entry** a specific action performed on the entry to the state
- **do** an ongoing action performed while in the state
- **on** a specific action performed as a result of a specific event
- **exit** a specific action performed on exiting the state



Note the syntax for a transition on a state diagram is:

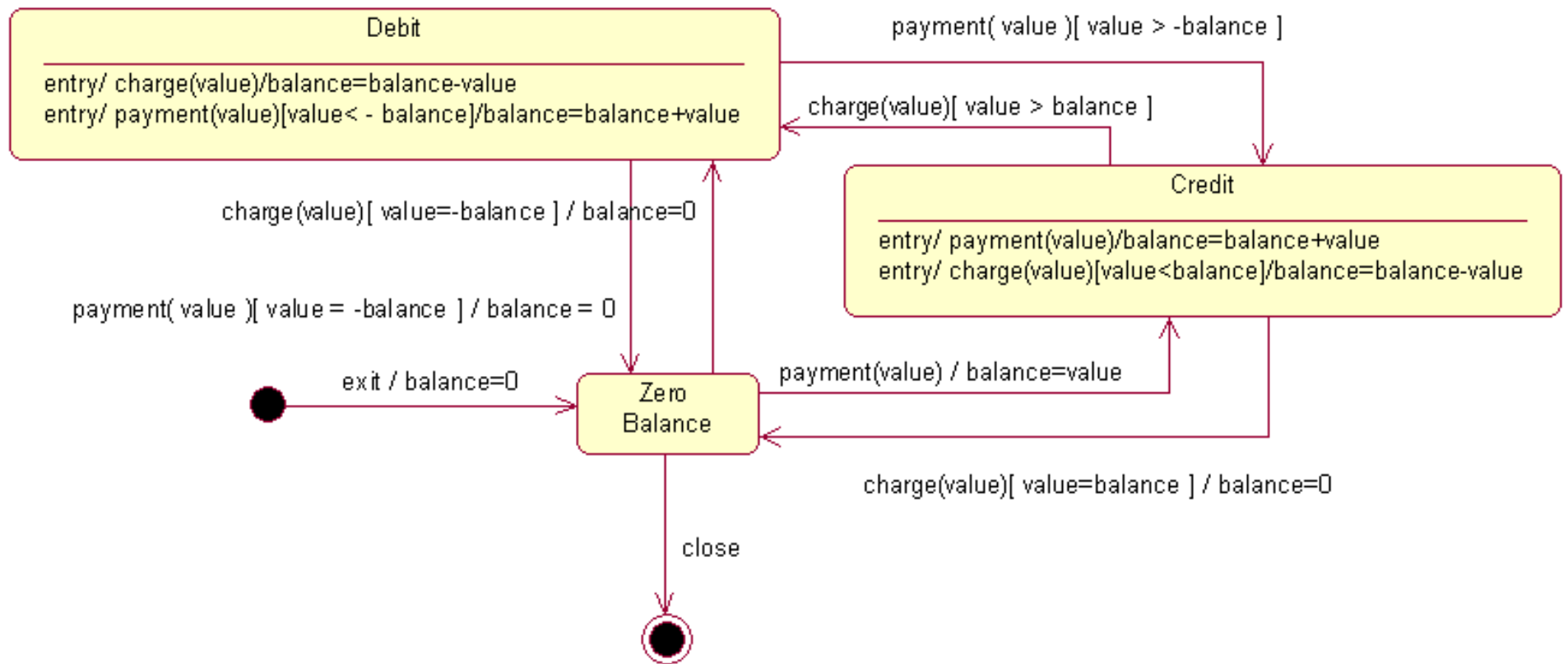
Event [Guard] / Action - where each part is optional.

The event signature takes the form **event-name** ' ('parameter-list') '

The guard condition is a boolean expression.

The action expression is executed when an event triggers the transition, and is the execution of one of an objects methods.





7. Modelling the dynamic behaviour of the interface

The

- user interface prototype
- class diagram, and
- sequence diagrams

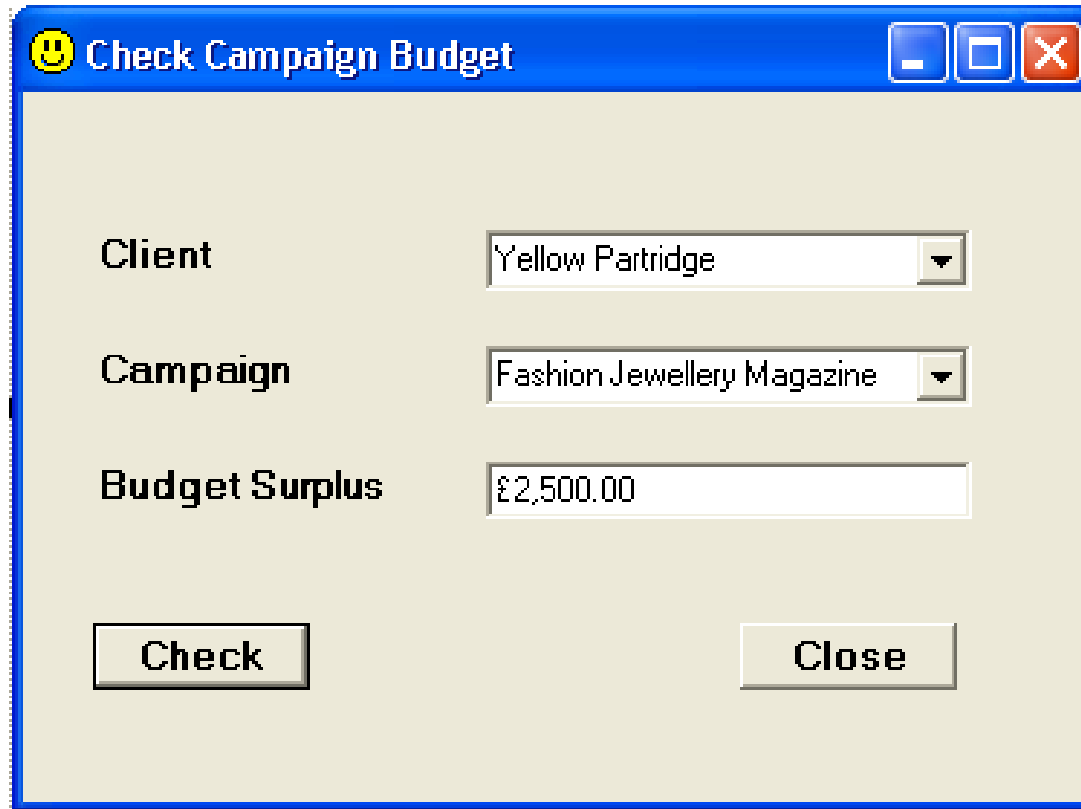
for the Check Campaign Budget Window tell us nothing about the permitted states of the components that make up the interface.

The very nature of such an interface will normally permit the fields to be filled and buttons to be pressed in any order.



What happens if

- A user clicks on the check button before client and campaign selected?
- A new client is selected after a campaign selected?



A screenshot of a Windows-style dialog box titled "Check Campaign Budget" with a yellow smiley face icon. The dialog box has a blue title bar with standard minimize, maximize, and close buttons. The main area is light beige and contains three labels with corresponding input fields: "Client" with a dropdown menu showing "Yellow Partridge", "Campaign" with a dropdown menu showing "Fashion Jewellery Magazine", and "Budget Surplus" with a text box containing "£2,500.00". At the bottom, there are two buttons: "Check" on the left and "Close" on the right.



What is needed is a set of interface rules.

User interface objects with constant behaviour.

- Whenever a client is selected, a list of campaigns is loaded into the campaign dropdown.
- The budget textfield is initially empty. It is cleared whenever a new client or a new campaign is selected. It is not editable.
- The close button may be pressed at any time to close the window.



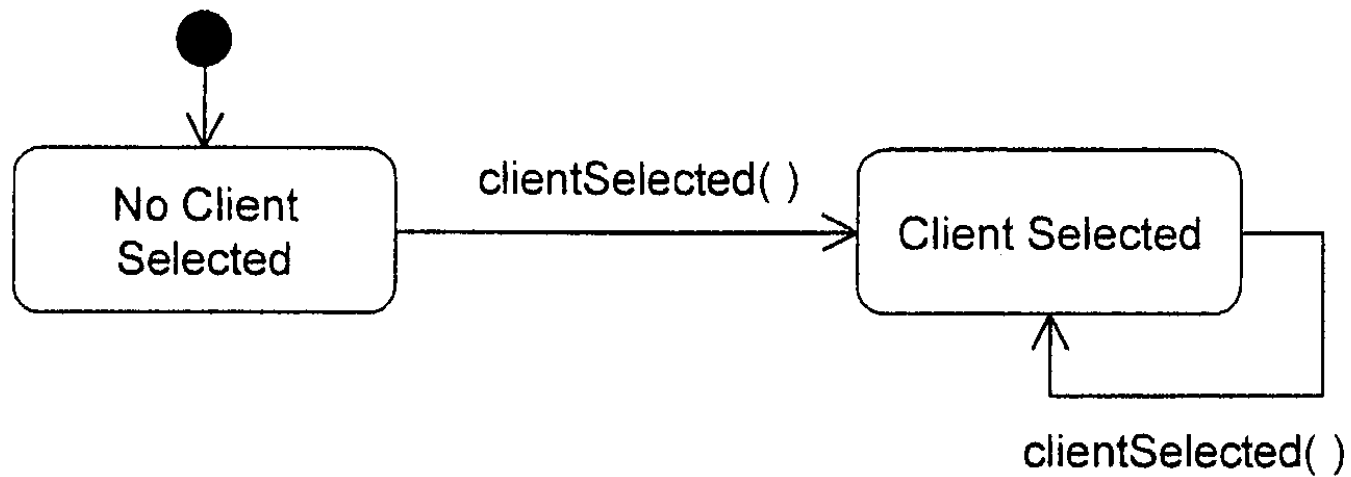
User interface objects with varying behaviour.

- The campaign dropdown is initially disabled. No campaign can be selected until a client has been selected. Once it has been loaded with a list of campaigns it is enabled.
- The check button is initially disabled. It is enabled when a campaign is selected. It is disabled whenever a new client is selected.

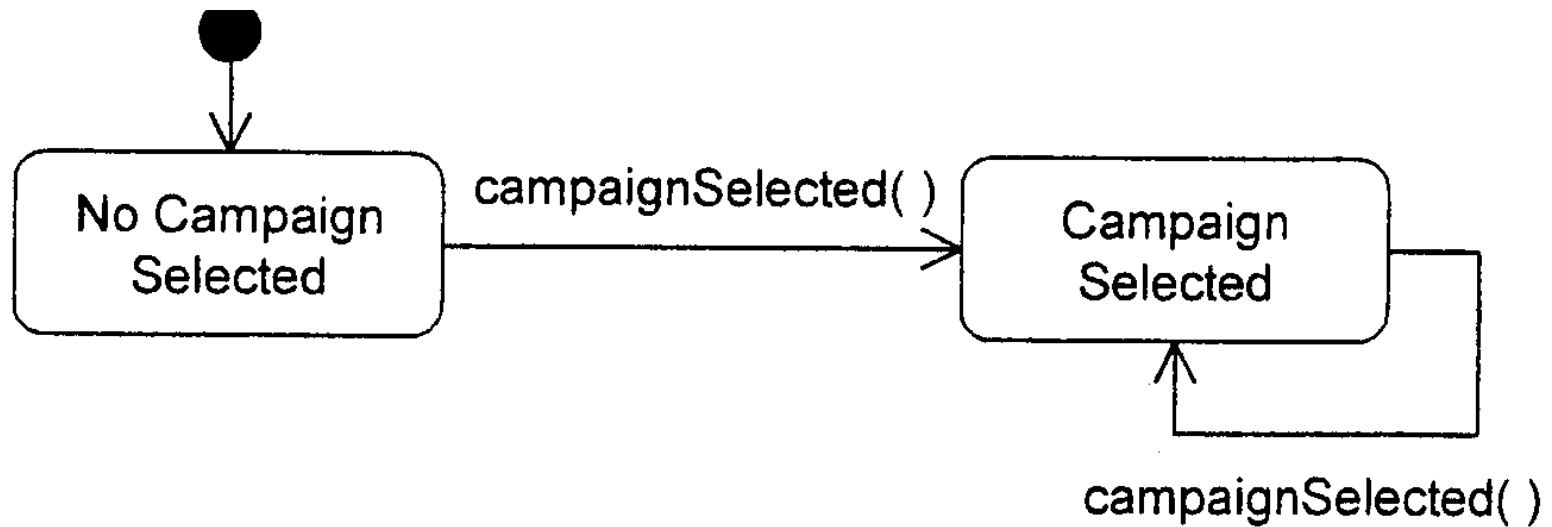
These rules can be modelled using a state chart diagram.



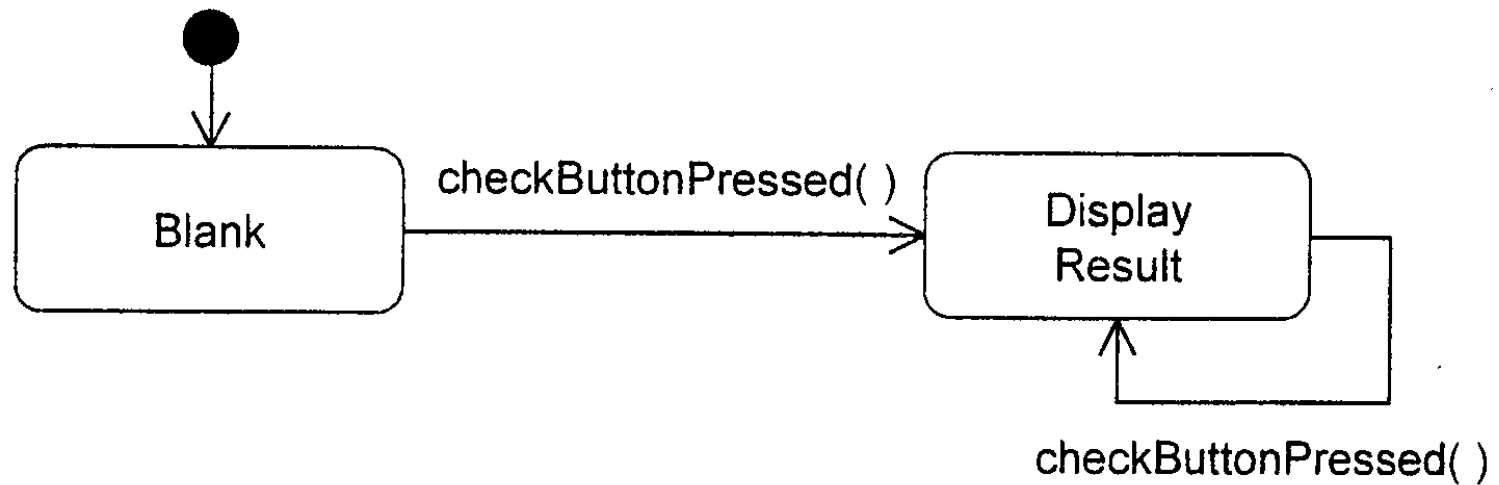
State of the client dropdown.



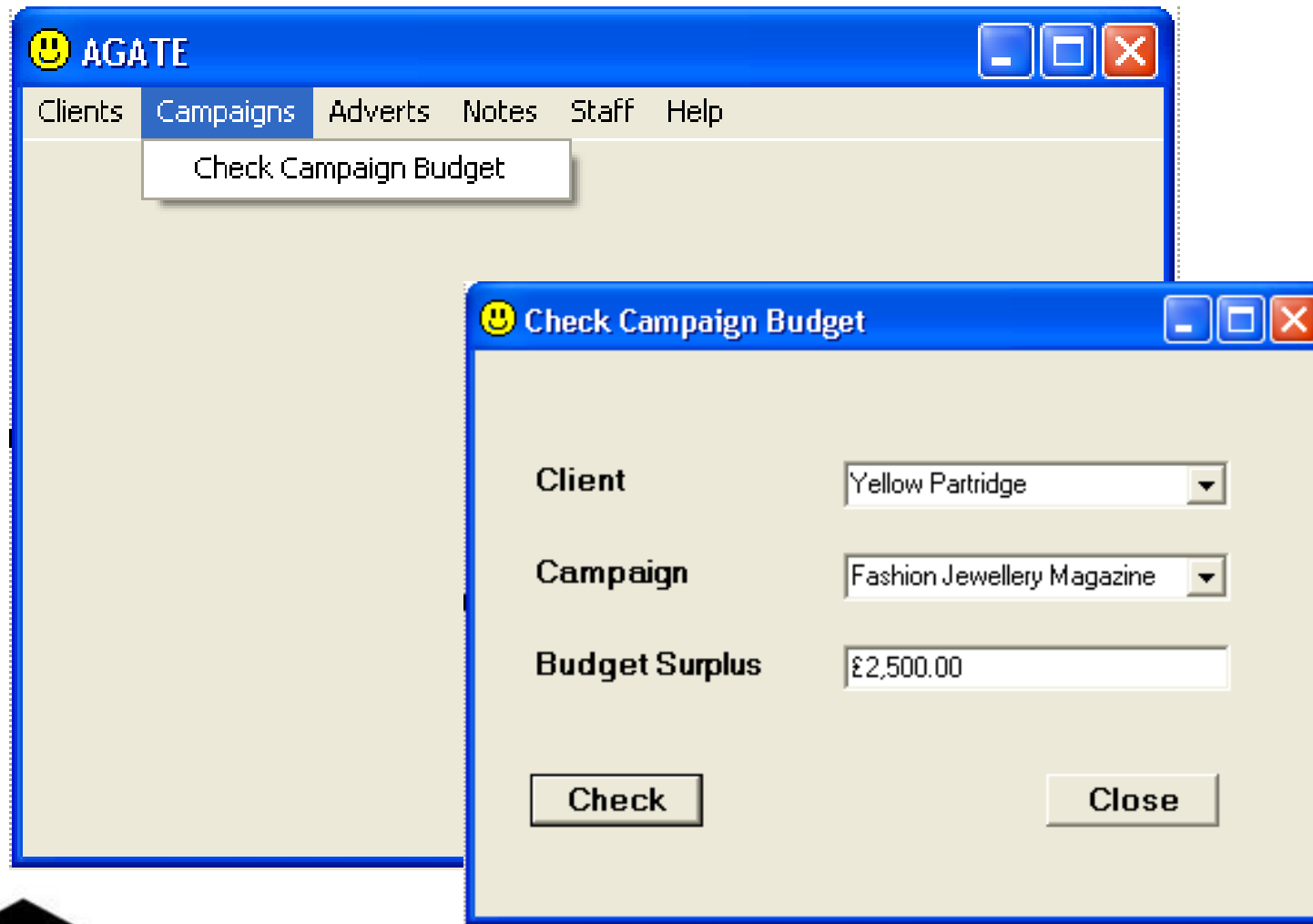
State of campaign drop down.



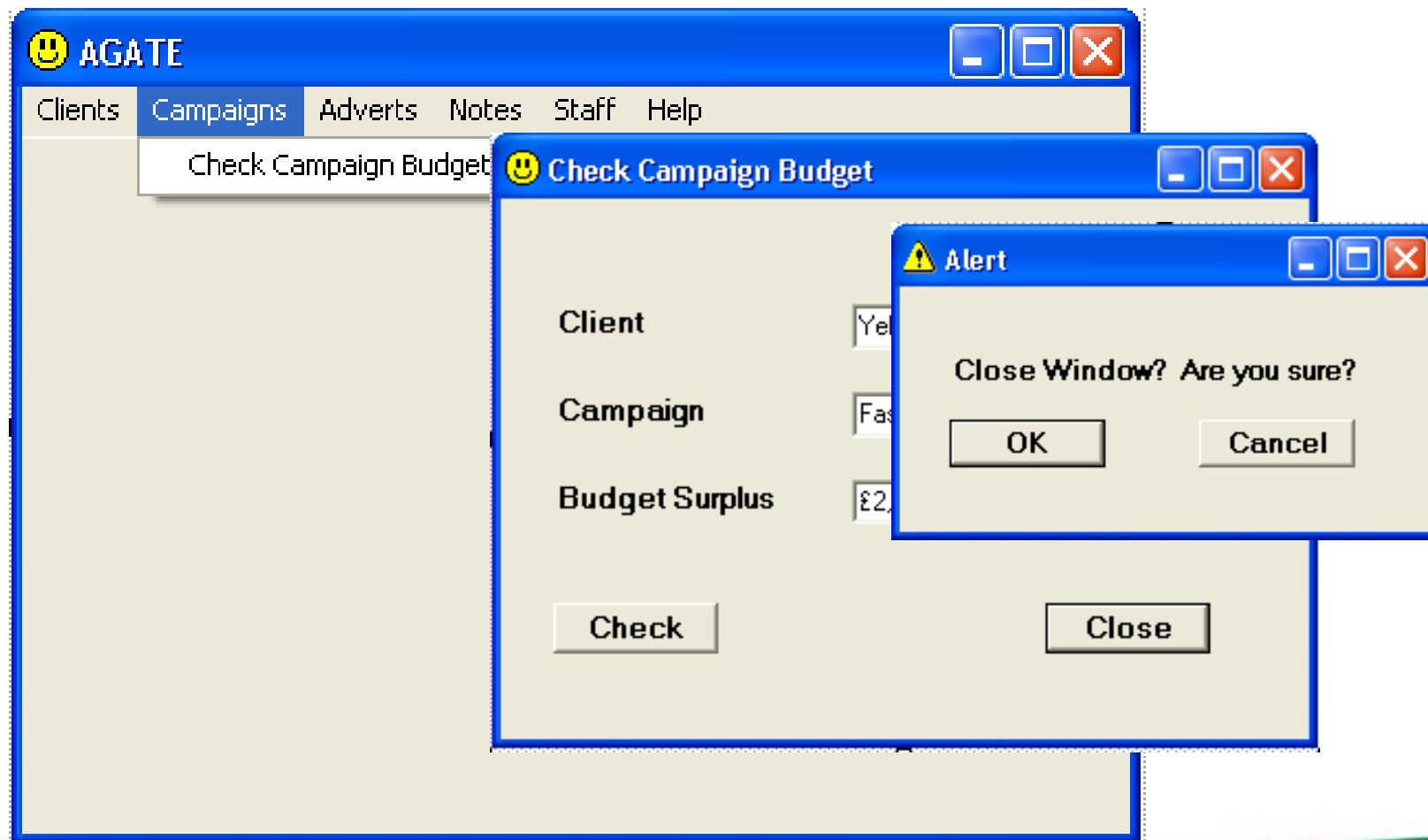
State of Budget Surplus textfield.

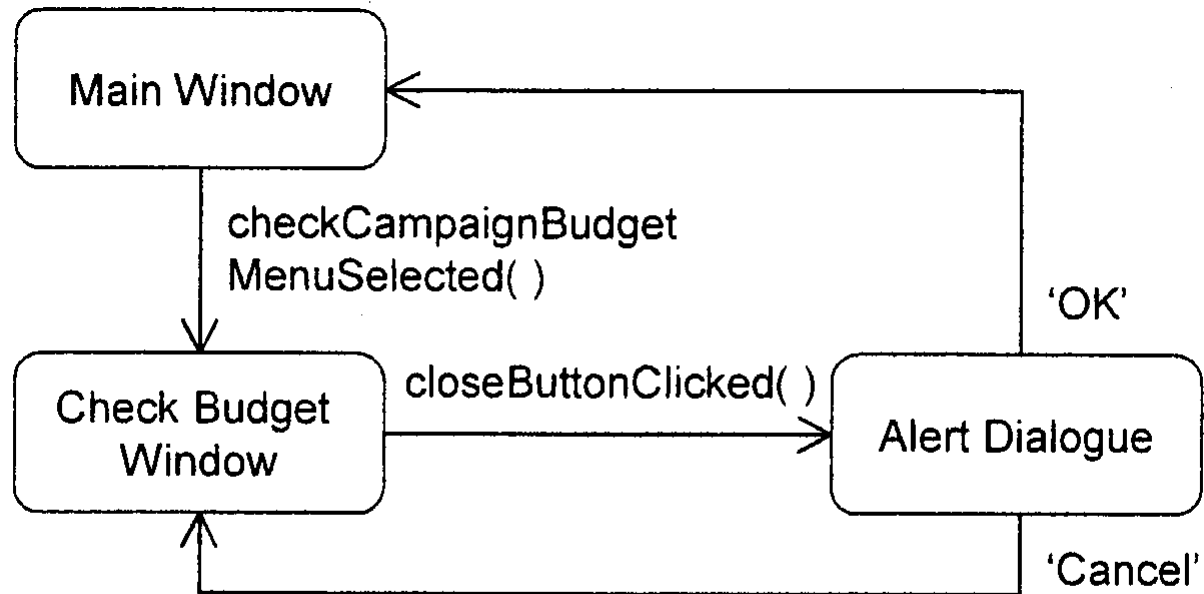


The window is entered from the main window when the check campaign budget menu item is selected.



When the close button is clicked, an alert dialogue is displayed. If OK is clicked the window is exited; if cancel is clicked then it carries on in the state it was in before the close button was clicked.

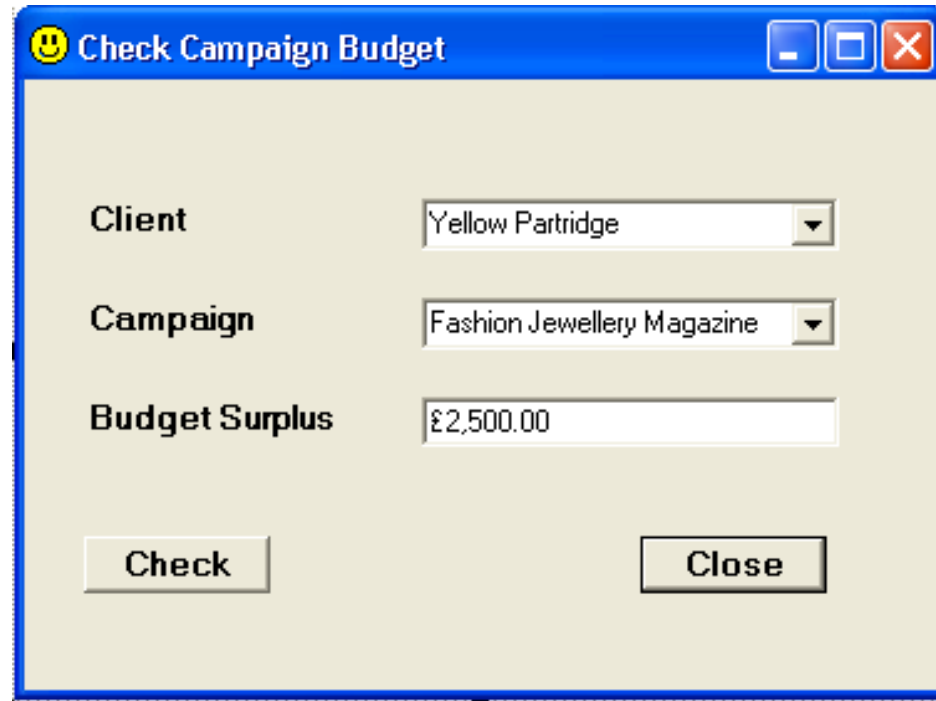




Top-level states.



Describe any of the states the user interface Check Campaign Surplus could be in prior to the Close button being clicked.



The image shows a Windows-style dialog box titled "Check Campaign Budget" with a yellow smiley face icon. The dialog has a blue title bar with standard minimize, maximize, and close buttons. The main area has a light beige background. It contains three labels on the left: "Client", "Campaign", and "Budget Surplus". To the right of "Client" is a dropdown menu showing "Yellow Partridge". To the right of "Campaign" is a dropdown menu showing "Fashion Jewellery Magazine". To the right of "Budget Surplus" is a text input field containing "£2,500.00". At the bottom of the dialog are two buttons: "Check" on the left and "Close" on the right.

How do we show on a state diagram which state to return to if the Cancel button is clicked in the Alert window?

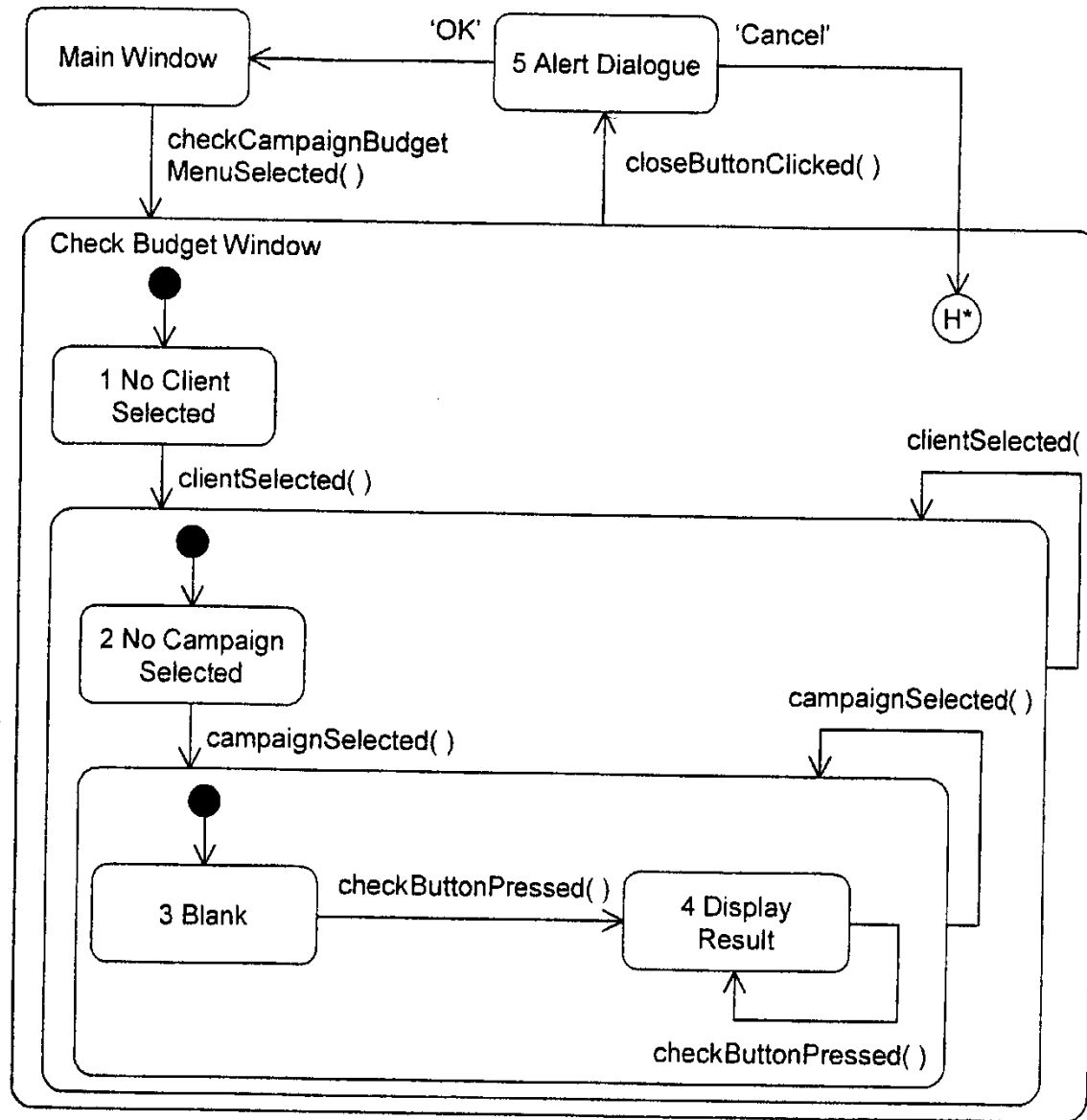


The deep history indicator $\textcircled{H^*}$ shows that when a transition takes place, it will return to the exact same state that it was in before the transition to the Alert dialogue state, however, far down the nested hierarchy of states that it was.

This works like a memory. The state of the user interface before the Close button was clicked is recorded, and the Cancel event returns it back to that recorded state.

We can now combine all the state diagrams for the components of the interface into one statechart.





deep history indicator

numbered states

Simplified version of statechart



8. Event-Action Table.

Often it is clearer to construct an event-action table than attempt to display all the actions in a statechart. The columns in the event-action chart represent:

- the current state of the object being modelled;
- the event that can take place;
- the actions associated with the combination of state and event;
- the next state of the object after the event has taken place.

The following event-action table relates to the check campaign budget interface statechart.



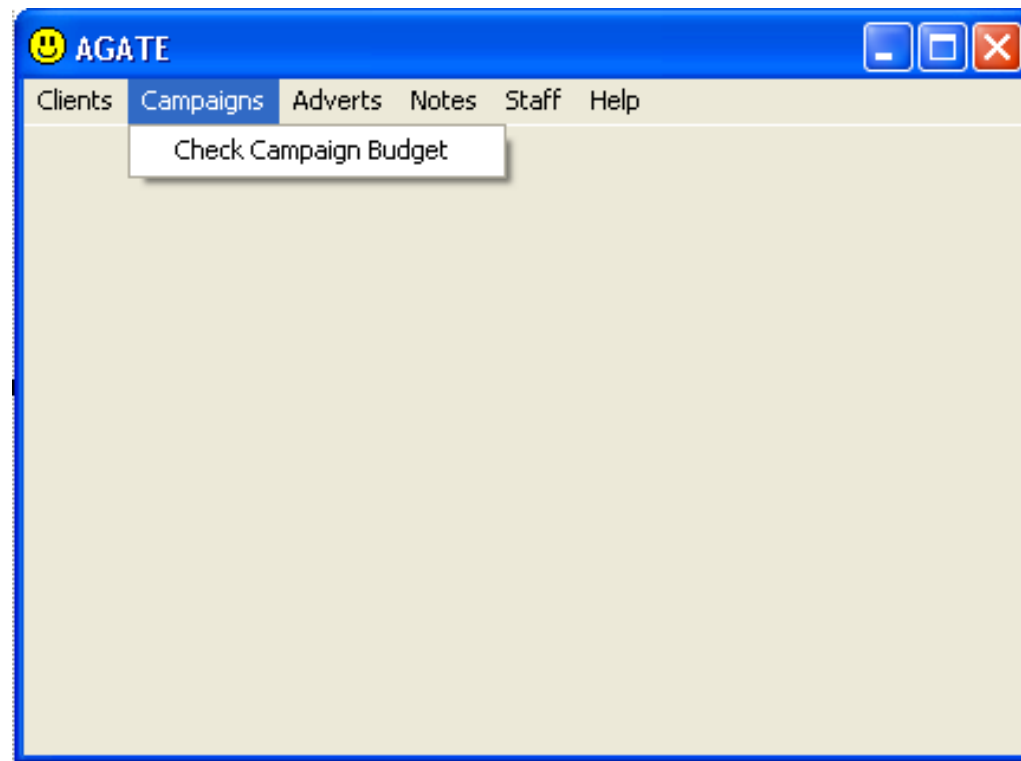
Current State	Event	Action	Next State
–	Check Campaign Budget menu item selected.	Display CheckCampaignBudgetUI. Load client dropdown. Disable campaign dropdown. Disable check button. Enable window.	1
1	Client selected.	Clear campaign dropdown. Load campaign dropdown. Enable campaign dropdown.	2
2, 3, 4	Client selected.	Clear campaign dropdown. Load campaign dropdown. Clear budget textfield. Disable check button.	2
2	Campaign selected.	Clear budget textfield. Enable check button.	3
3	Check button pressed.	Calculate budget. Display result.	4
3, 4	Campaign selected.	Clear budget textfield.	3
4	Check button pressed.	Calculate budget. Display result.	4
1, 2, 3, 4	Close button clicked.	Display alert dialogue.	5
5	OK button clicked.	Close alert dialogue. Close window.	–
5	Cancel button clicked.	Close alert dialogue.	H*

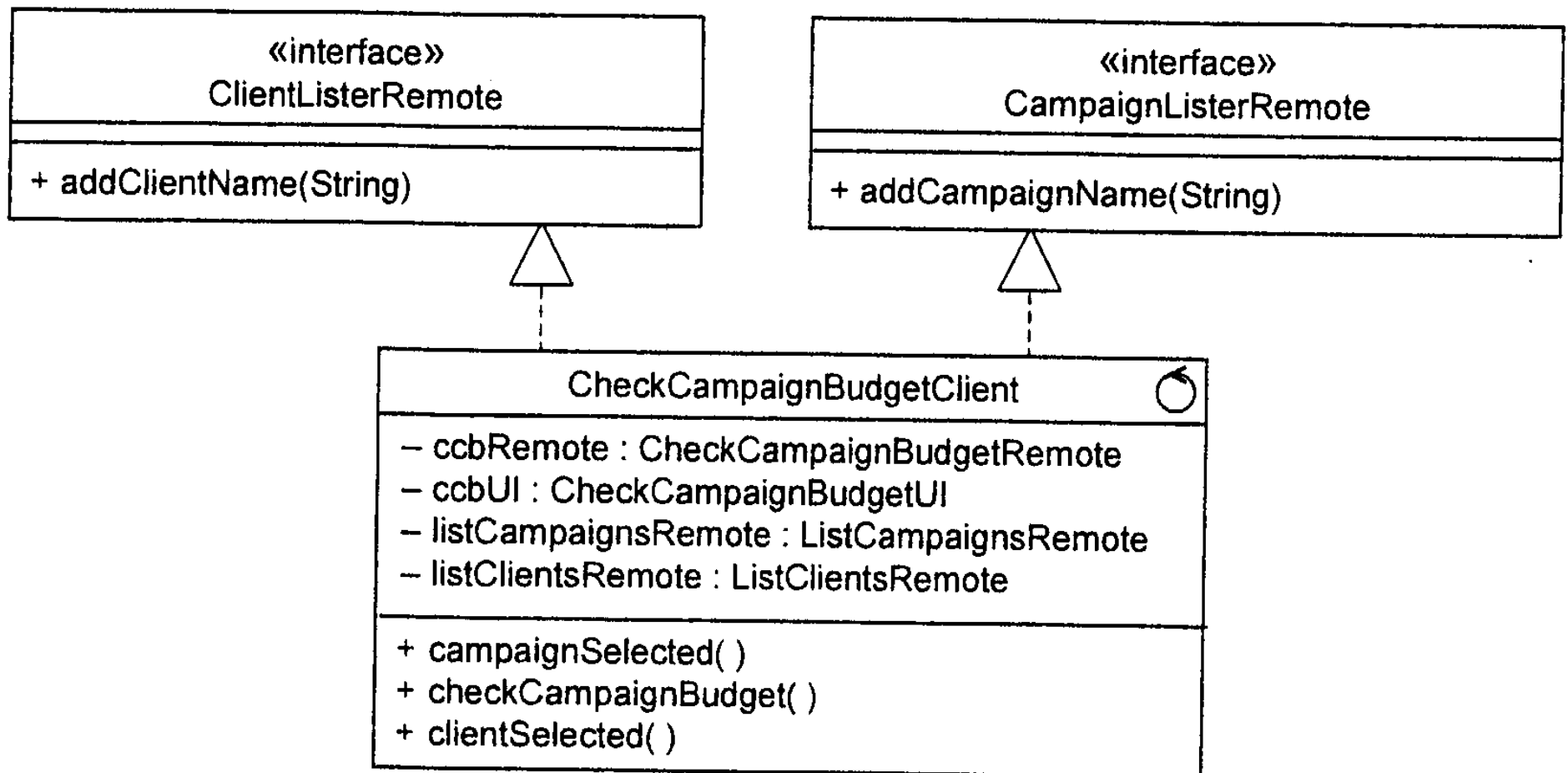
Event-action table



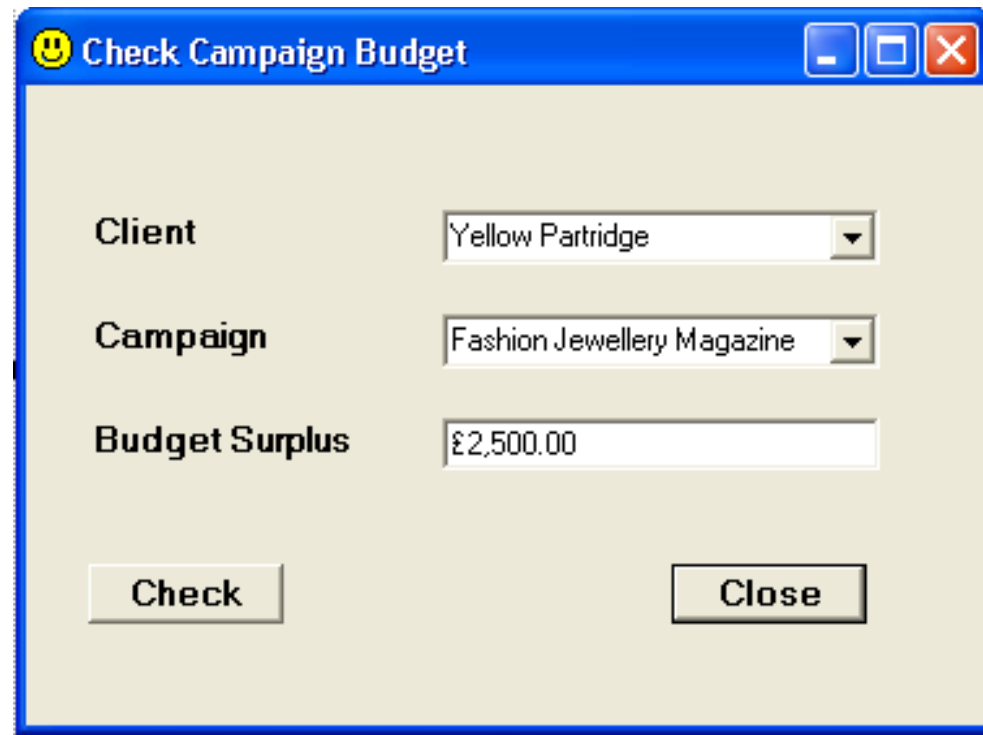
9. Designing classes.

The control window shown below is an object, therefore must be defined by a class – **CheckCampaignBudgetClient**.





The Check Campaign Budget window shown below is also an object, therefore must be defined by a class - **CheckCampaignBudgetUI**.



Check Campaign Budget

Client: Yellow Partridge

Campaign: Fashion Jewellery Magazine

Budget Surplus: £2,500.00

Check Close



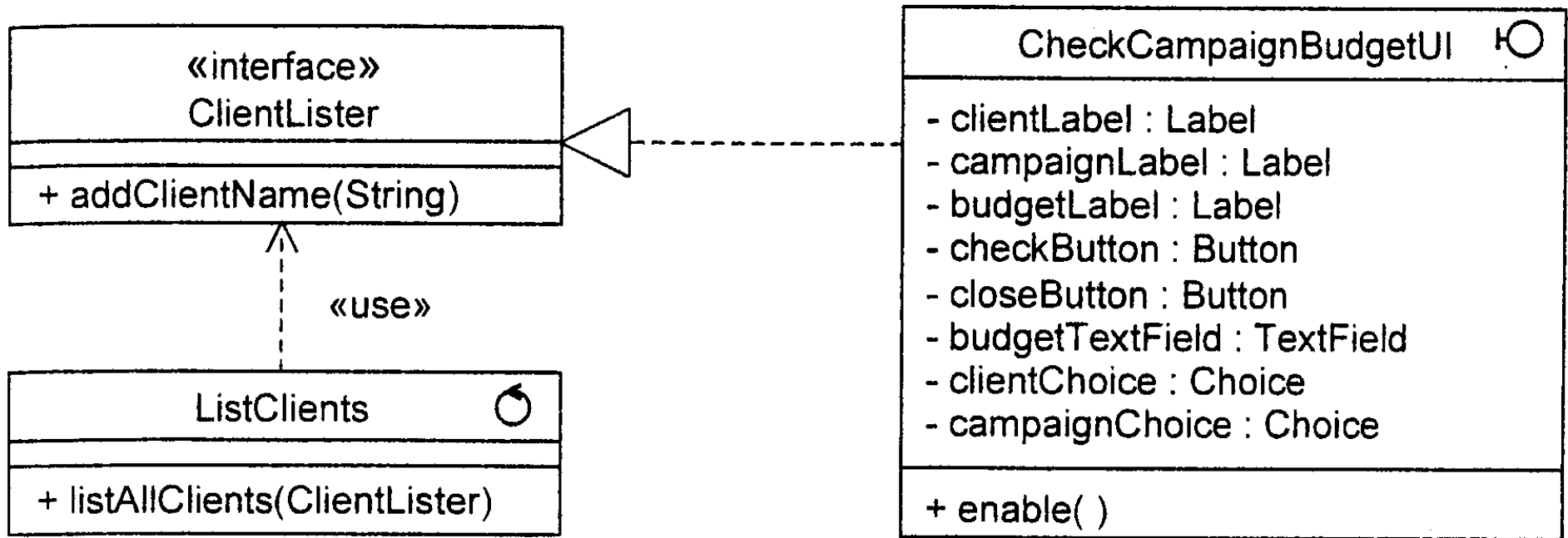
A list of clients may need to be used in other parts of the system. For this reason it makes sense to have a separate control class to List Clients.

We cannot expect List Clients to know about all the different boundary classes to which it could send the message `addClientName(name)`.

We use the idea of an interface to specify the operations that all these boundary classes must respond to.

Design of other use cases may identify other operations that must also be part of the interface, for example `clearAllNames()`, `removeClientName(name)`, etc.

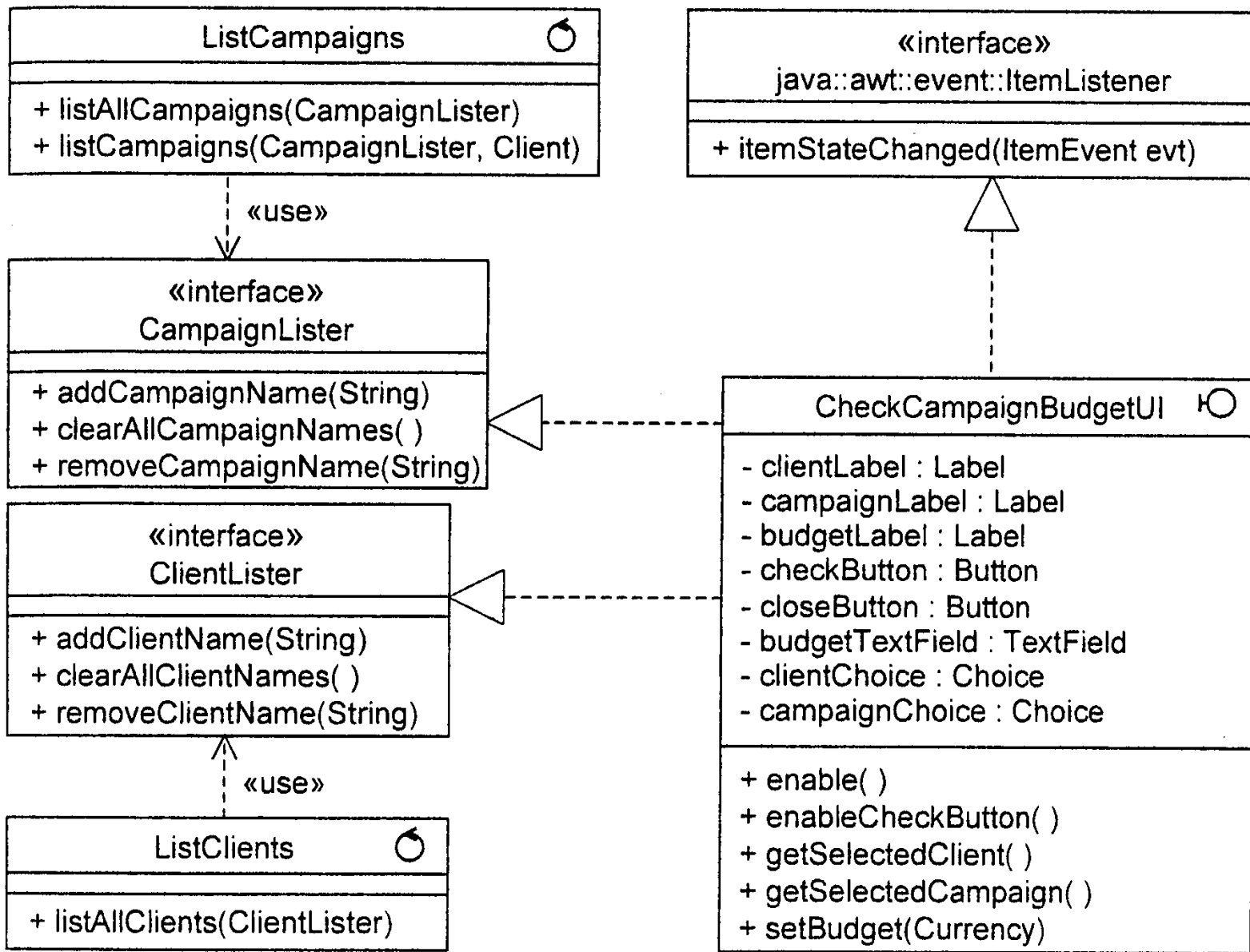




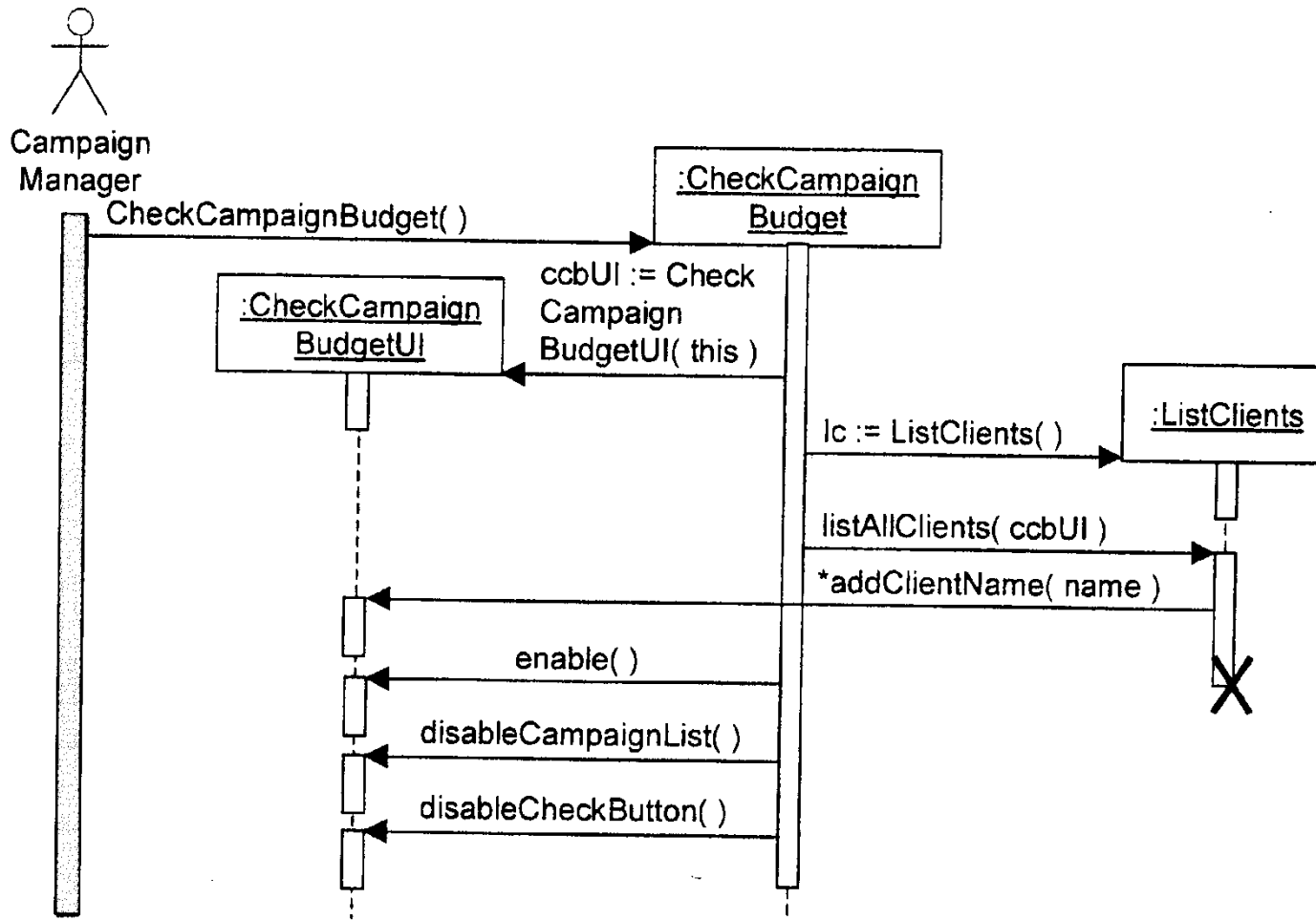
Notice that the interface `CheckCampaignBudgetUI` must implement the method `addClientName` in a specific manner to the 'choice box' in this interface.

A similar interface exists to include the method `addCampaignName`.



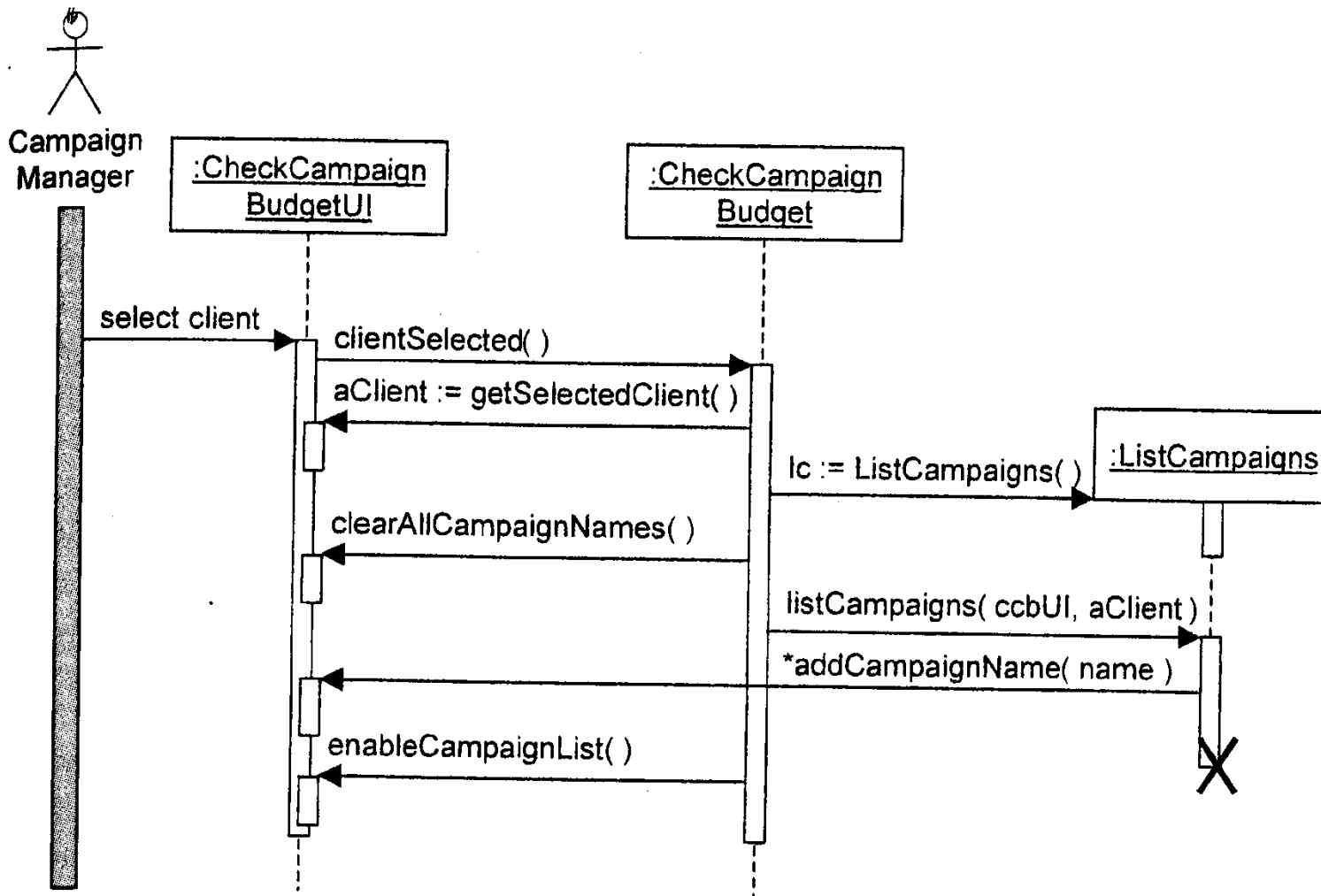


10. Sequence diagrams revisited.



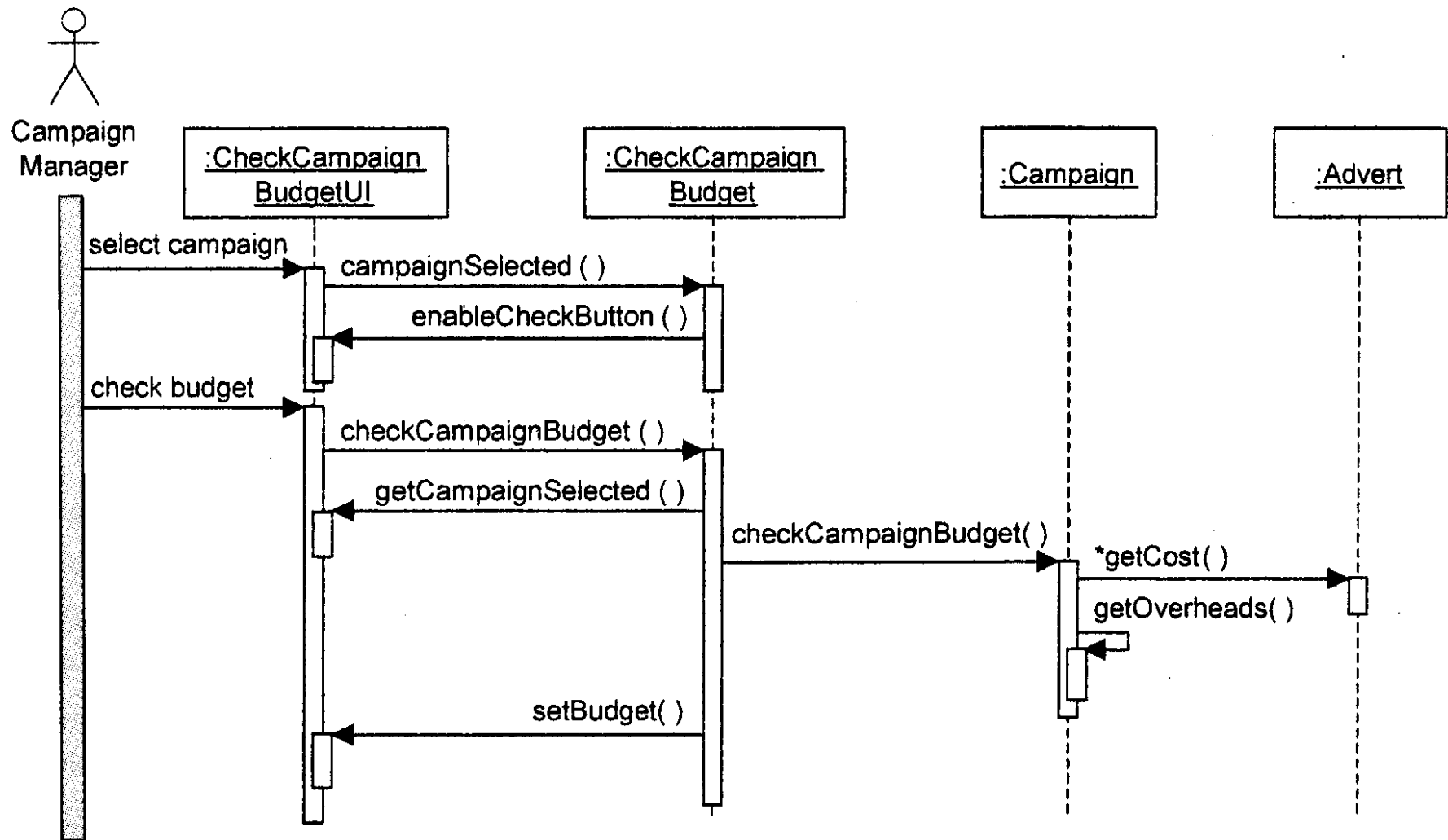
Revised sequence diagram for first part of interaction.





Revised sequence diagram for second part of interaction.





Final part of interaction for use case Check campaign budget.



11. Further Reading:

Lunn, K - Software Development with UML
– **Chapter 13.**

Quatrani, T - Visual Modeling with Rational Rose 2002 and UML
– **Chapter 9.**

Maciaszek, L – Requirements Analysis and System Design
– **Chapter 7.**



Bibliography:

Object-oriented Systems Analysis and Design using UML
-Bennett, McRobb and Farmer (Second Edition)

Chapter 12 Moving into Design

Chapter 13 System Design

Chapter 16 Human-Computer Interaction

Chapter 17 Designing Boundary Classes

