# Chap. 17
# Designing Boundary Classes

*Object Oriented Systems Analysis and Design Using UML*, (4th Edition), McGraw Hill

## Topics Covered

- Architecture of Presentation Layer
- Prototyping User Interface
- Designing Classes
- Designing Interaction with Sequence Diagrams
- User Interface Design Patterns
- Modelling the Interface Using State Machines

## Architecture of the Presentation Layer

- We aim to separate the classes that have the responsibility for the interface with the user, or with other systems, (boundary classes) from the business classes (entity classes) and the classes that handle the application logic (control classes)
- This is the Three-Tier Architecture

## Presentation Layer

- Handles interface with users and other systems
- Formats and presents data at the interface
- Presentation can be for display as text or charts, printing on a printer, speech synthesis, or formatting in XML to transfer to another system
- Provides a mechanism for data entry by the user, but the events are handled by control classes

# 3-Tier Architecture

- Different authors have used different terms
  - Boundary, Control, Entity
  - Model, View, Controller
  - Human Interaction Component, Problem Domain Component, Task Management Component

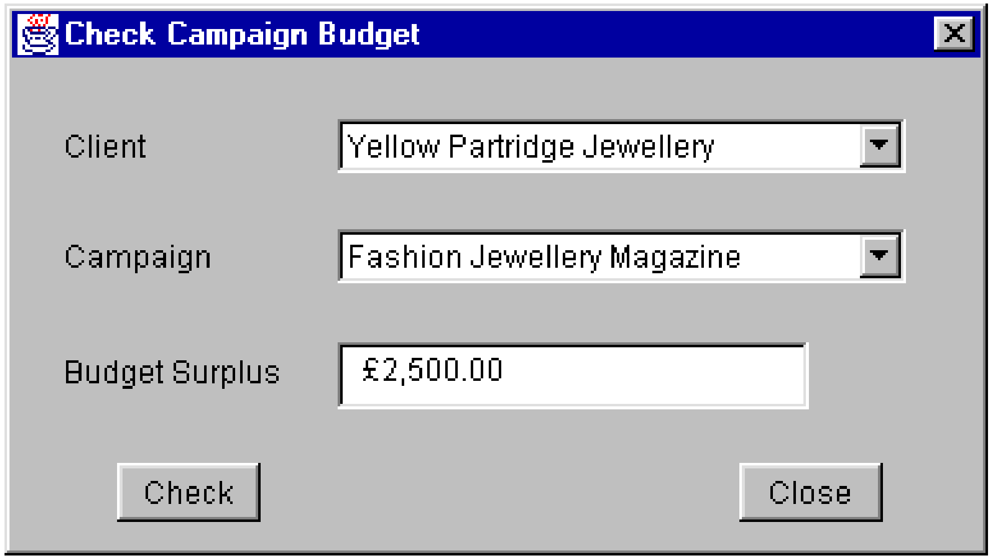
## Developing Boundary Classes

① Prototype the user interface
② Design the classes
③ Model the interaction involved in the interface
④ Model the control of the interface using state machines (if necessary)

## Prototyping the User Interface

- A prototype is a model that looks, and partly behaves, like the finished product but lacks certain features

## Check Campaign Budget Use Case Prototype


Check Campaign Budget

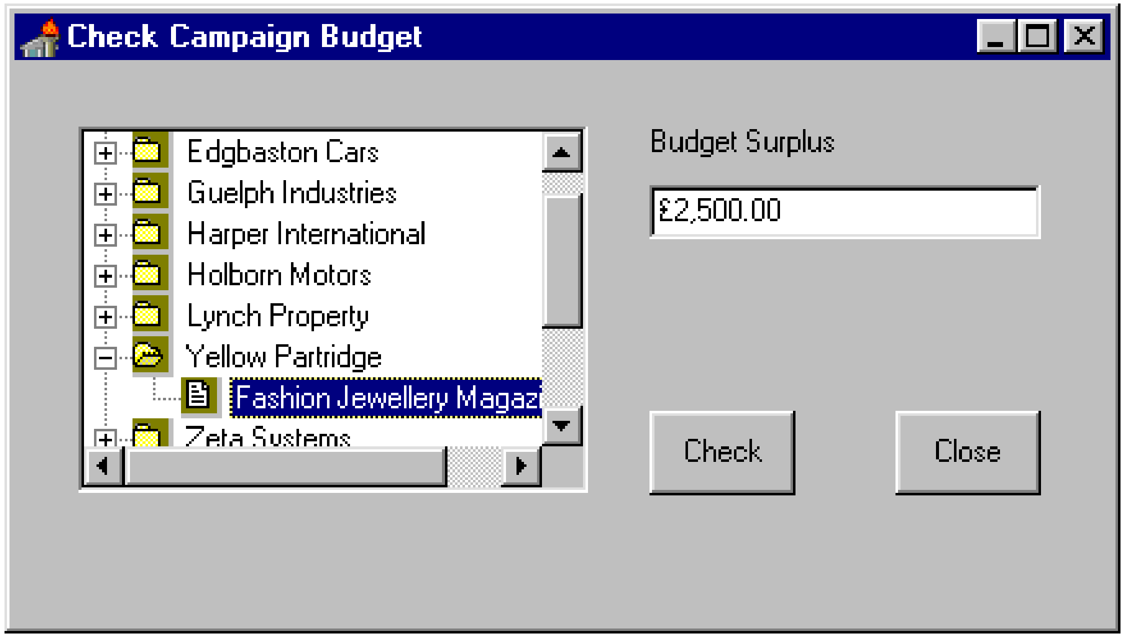
Client: Yellow Partridge Jewellery

Campaign: Fashion Jewellery Magazine

Budget Surplus: £2,500.00

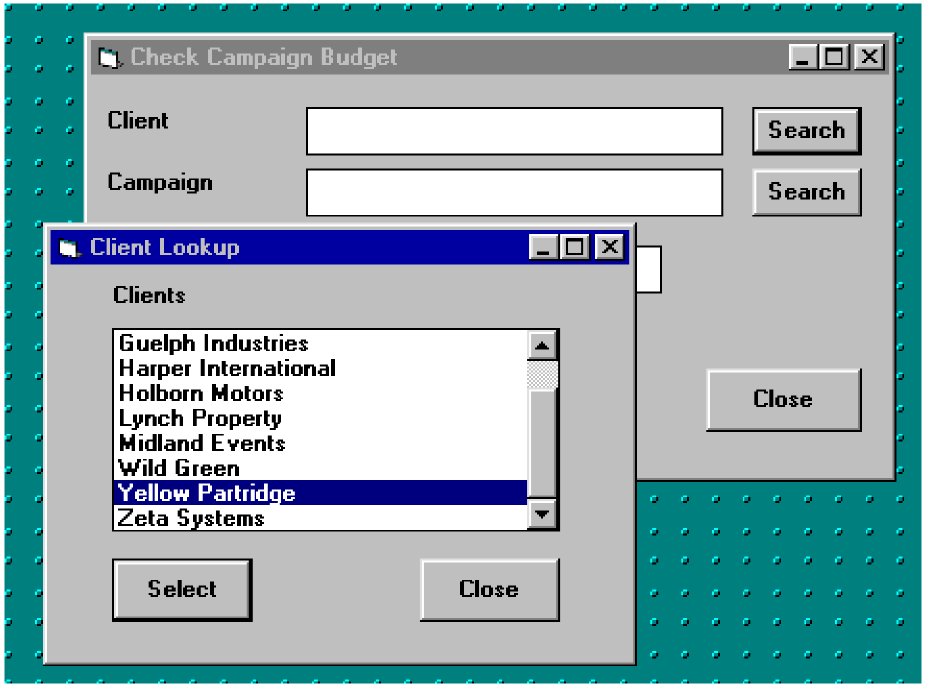
- In this prototype, Clients and Campaigns are selected in drop-down lists
- There are other ways…

# Check Campaign Budget Use Case Prototype



- Using a treeview control…

# Check Campaign Budget Use Case Prototype



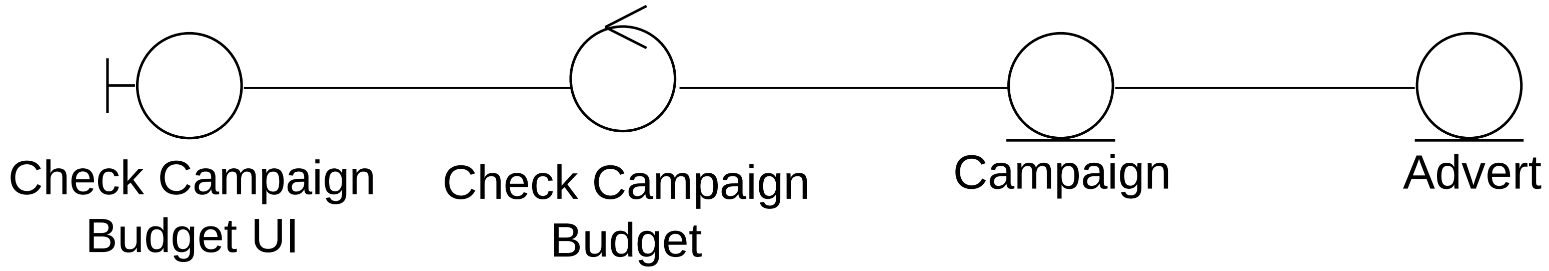
Guelph Industries
Harper International
Holborn Motors
Lynch Property
Midland Events
Wild Green
Yellow Partridge
Zeta Systems

- Using a separate look-up window

## Designing Classes

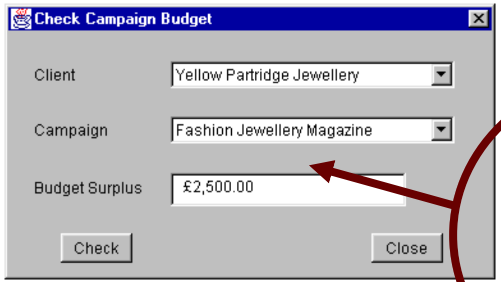
- Start with the collaborations from the analysis model
- Elaborate the collaborations to include necessary boundary, entity and control classes

# Collaboration for Check campaign budget



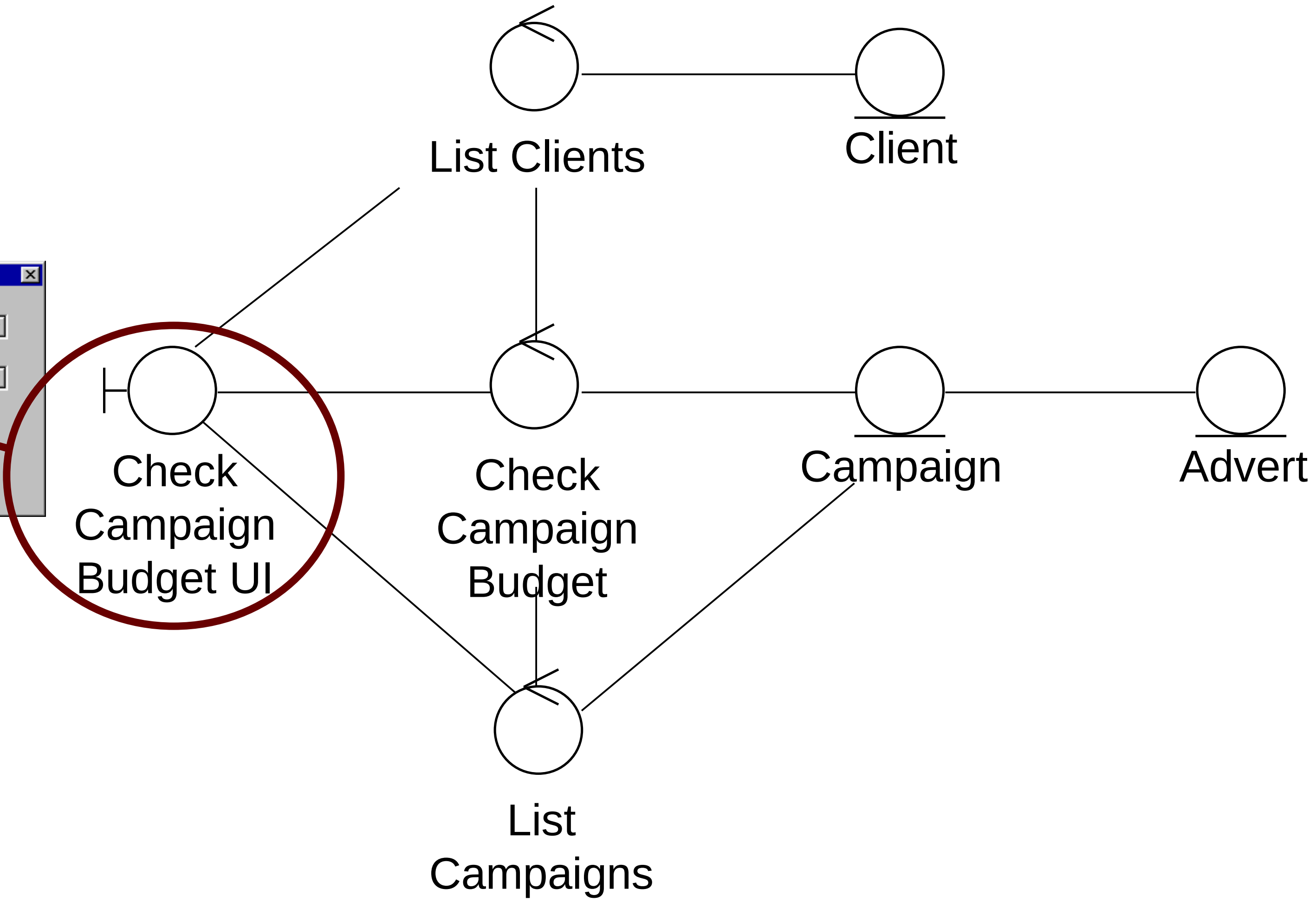
- In order to find the right Campaign, we also need to use the Client class, even though it doesn't participate in the real process of checking the budget
- We also add control classes for the respons-ibilities of listing clients and campaigns

# Collaboration for Check campaign budget

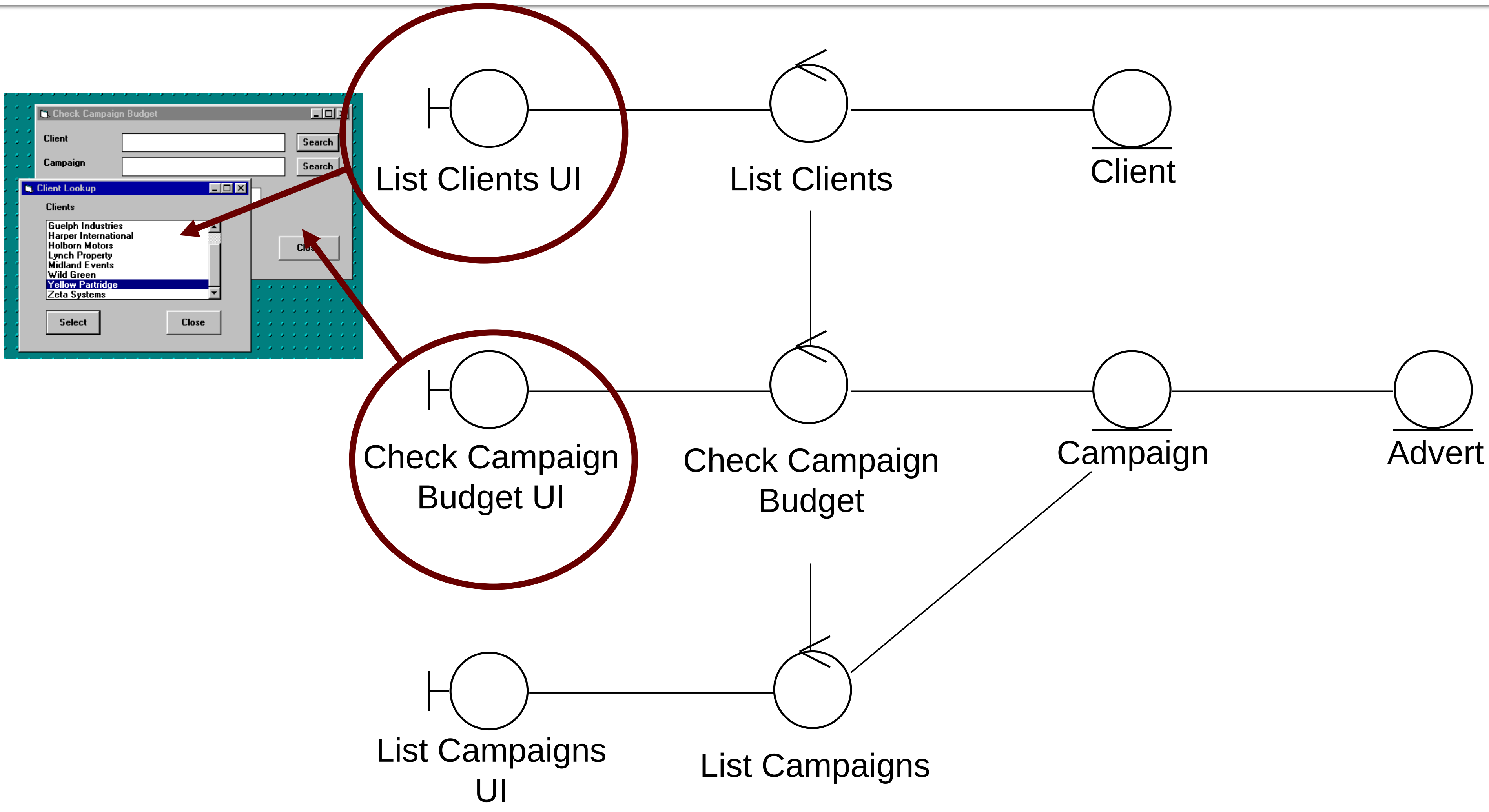


Check Campaign Budget

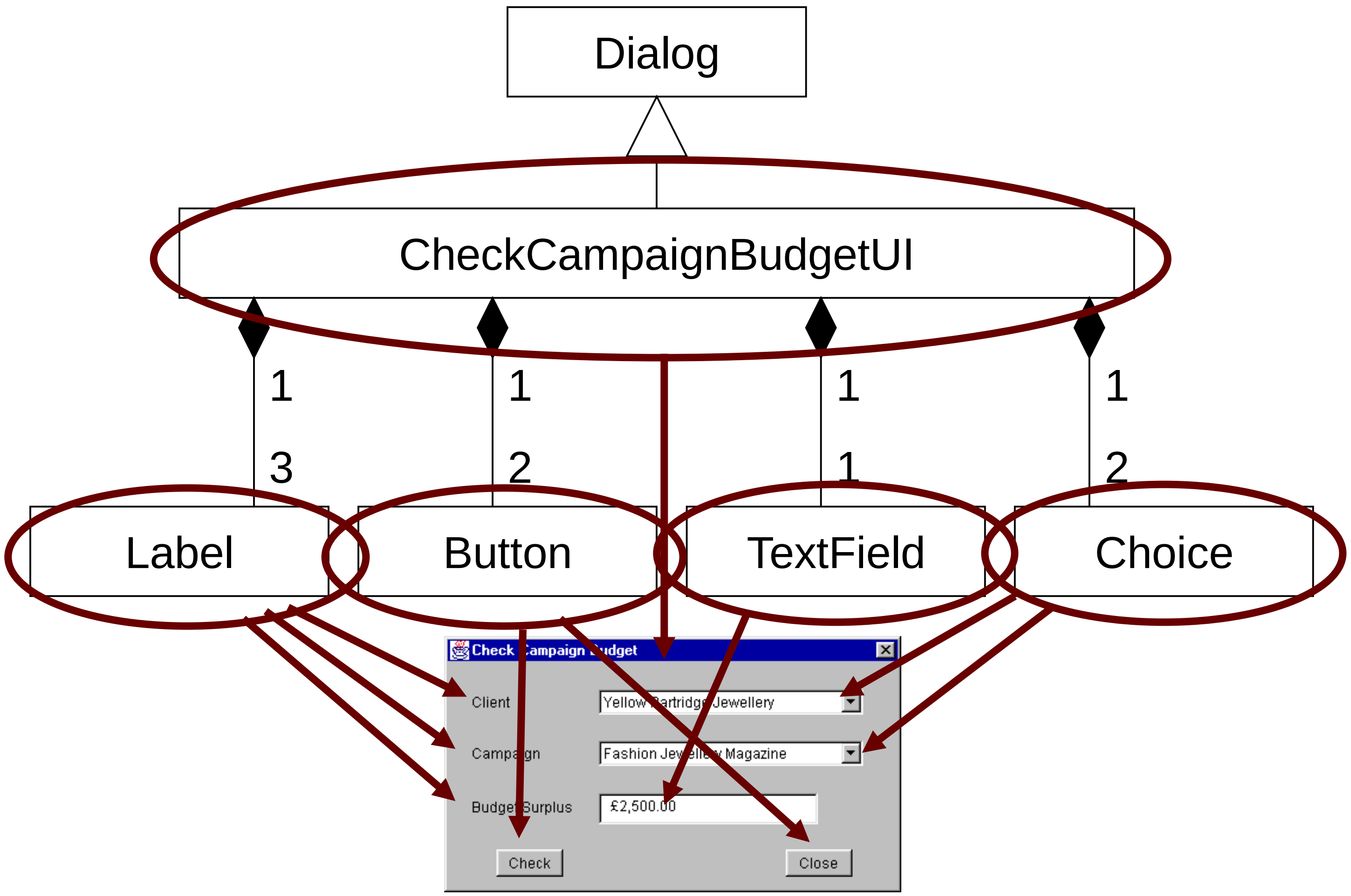
| | |
|---|---|
| Client | Yellow Partridge Jewellery |
| Campaign | Fashion Jewellery Magazine |
| Budget Surplus | £2,500.00 |



# Alternative Collaboration for Check campaign budget

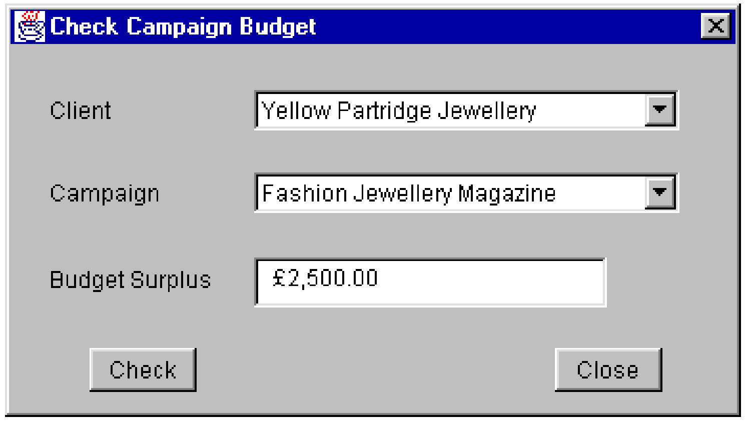


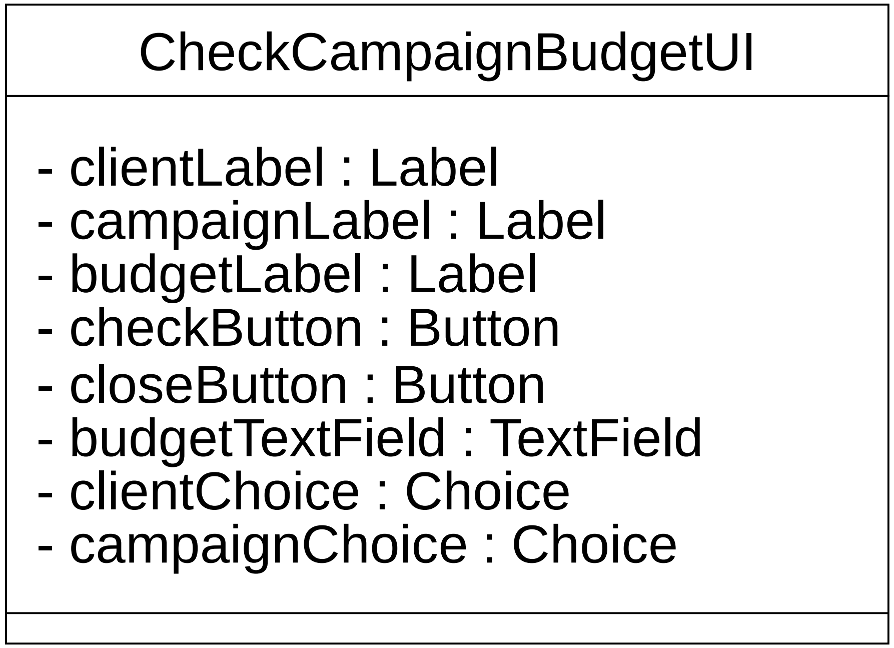
# Class Diagram for CheckCampaignBudgetUI



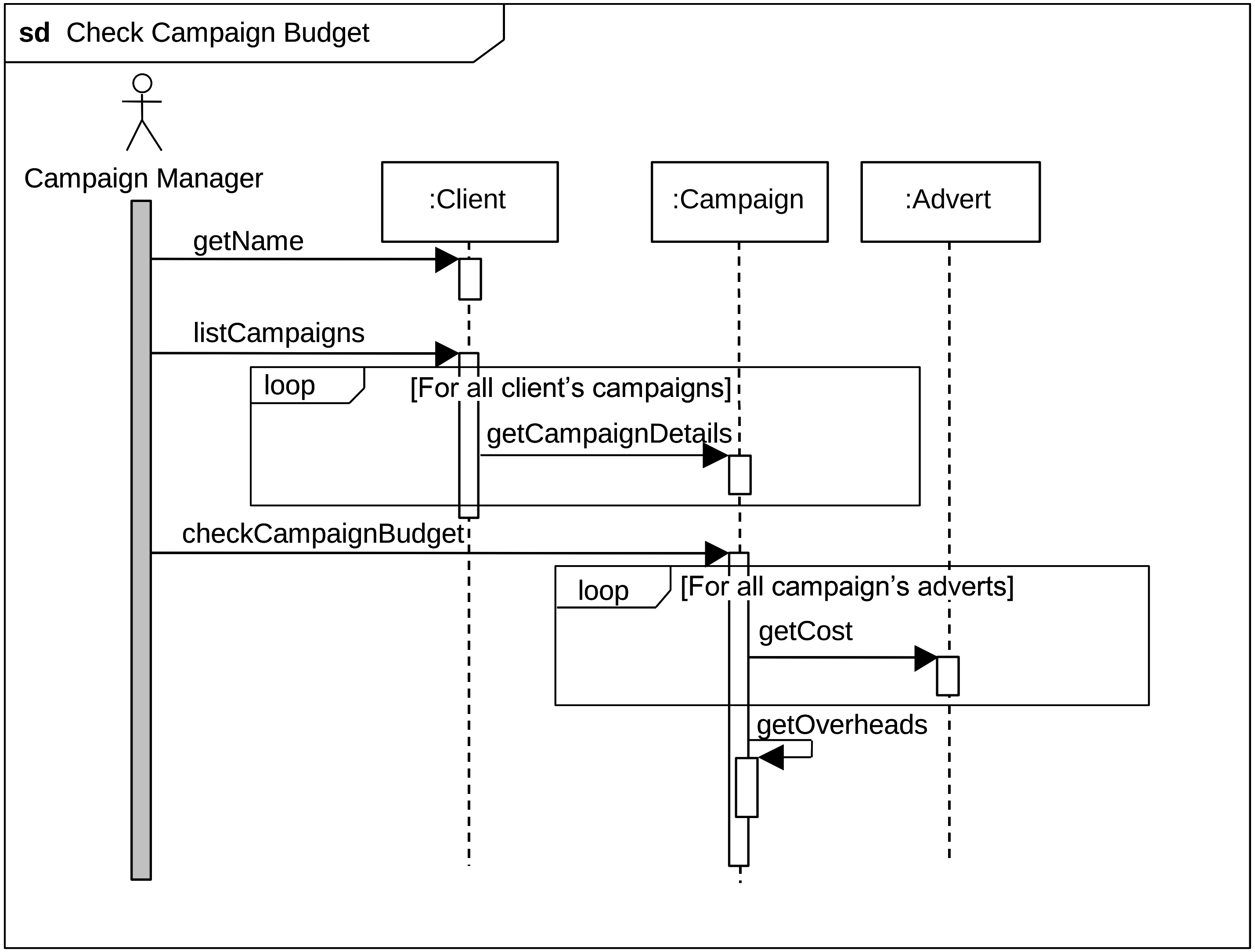
# Single Class for CheckCampaignBudgetUI

- Draw in your own lines to show which attribute is which element of the interface





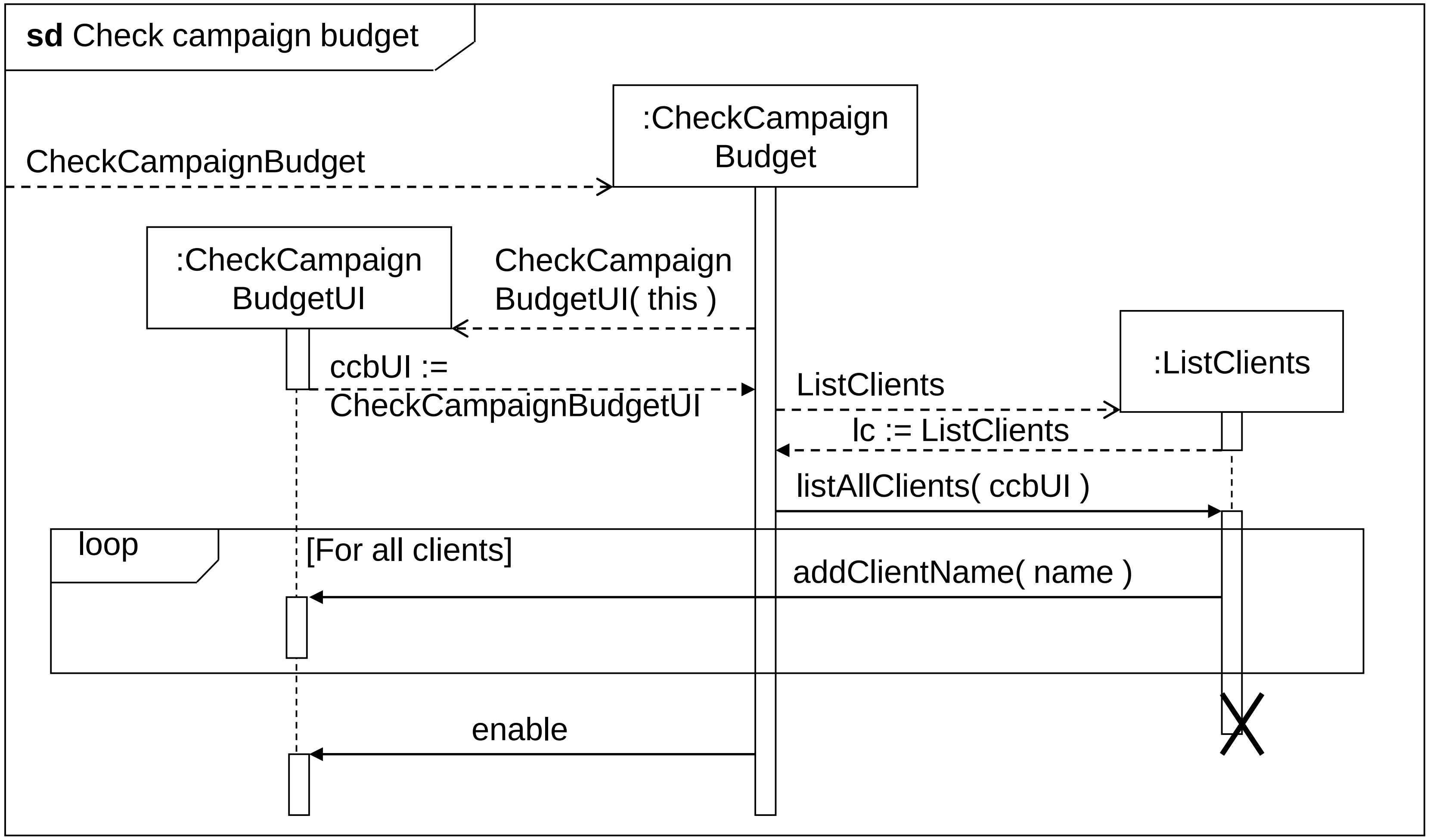
# Designing Interaction with Sequence Diagrams



## Sequence Diagrams

- The sequence diagram on the previous slide just shows the entity classes
- We also have the collaboration diagram from an earlier slide showing the control and boundary classes
- We now need to model the interaction more detail

# First Part of Sequence Diagram



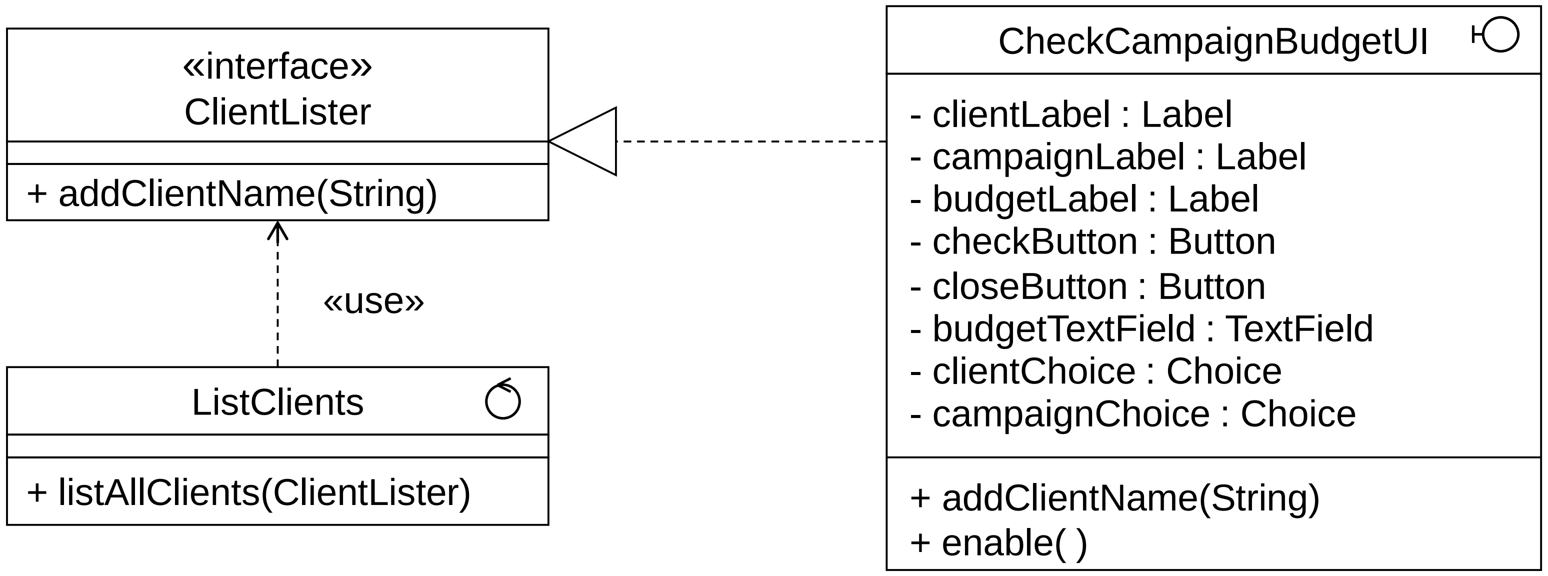
## First Part of Sequence Diagram

- The control class
  - creates the instance of the boundary class
  - creates the instance of the **`ListClients`** control class
  - passes to **`:ListClients`** a reference to the boundary class
  - **`:ListClients`** then sets the name of each client in turn into the boundary class by calling **`addClientname(name)`** repeatedly in the loop

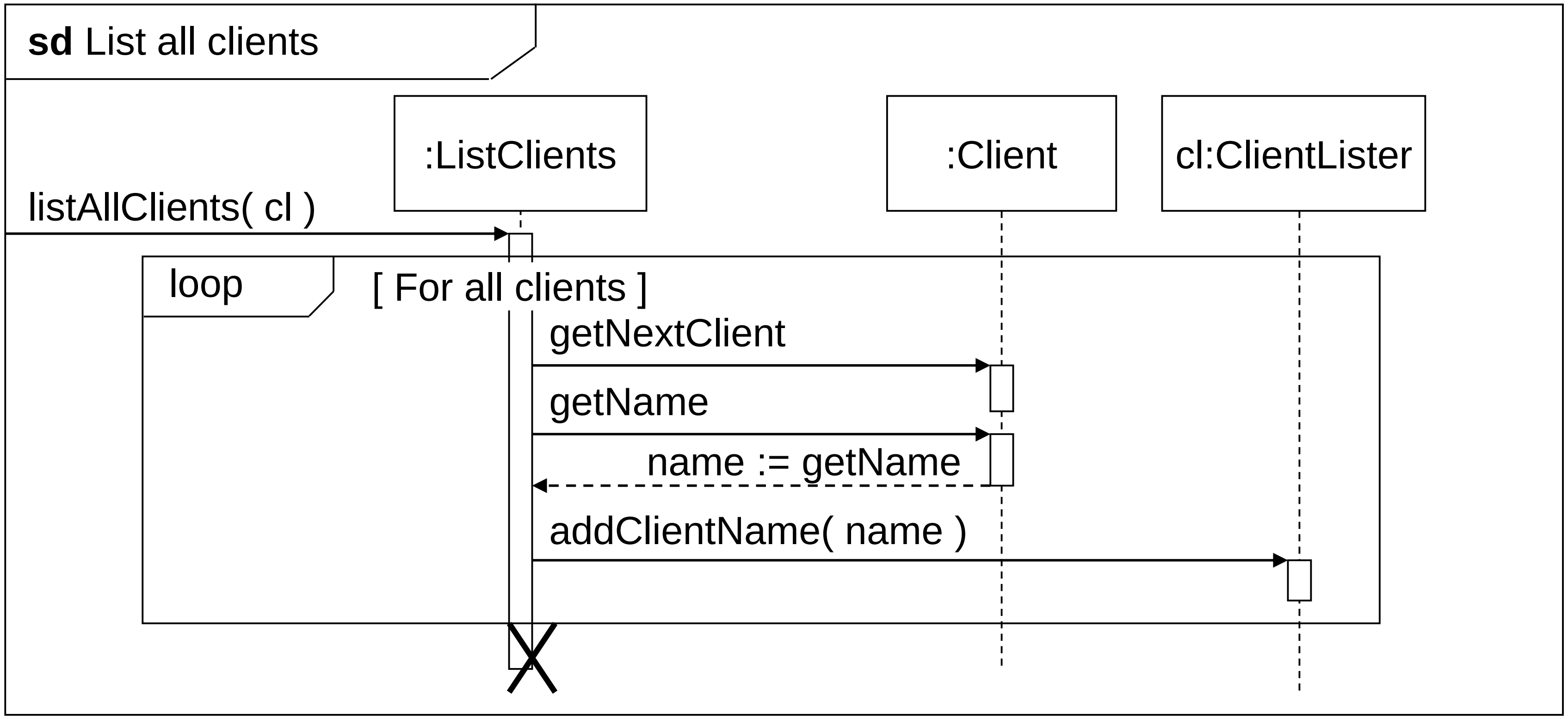
## Using Interfaces

- We don’t mean user interfaces!
- Many boundary classes will need to list clients to allow the user to select a client
- The **`ListClients`** control class doesn’t need to know how the boundary class lists them
- The boundary class needs to implement the **`ClientLister`** interface and provide an implementation of the operation **`addClientName(String)`**

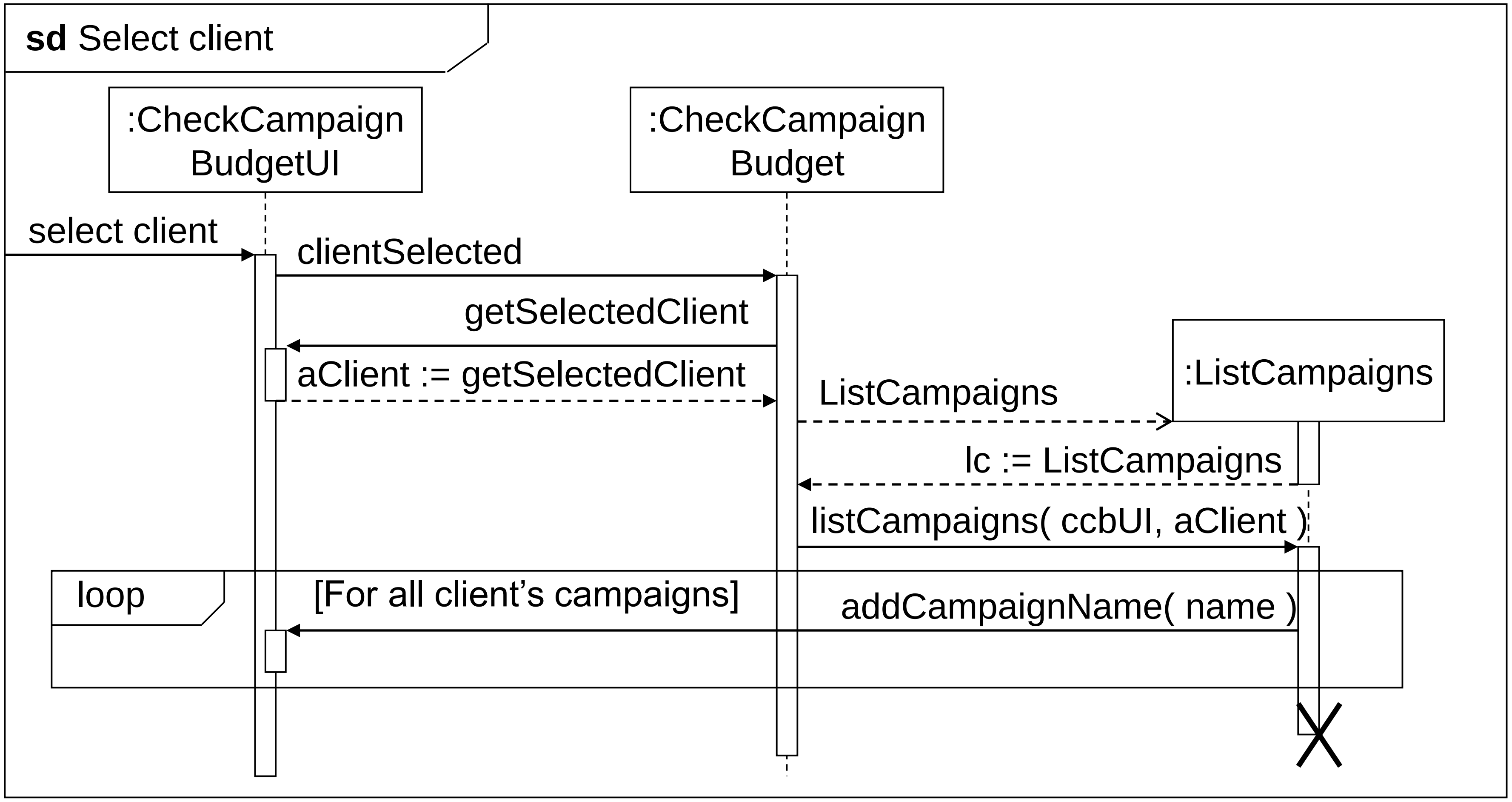
## Using Interfaces



# listAllClients Operation

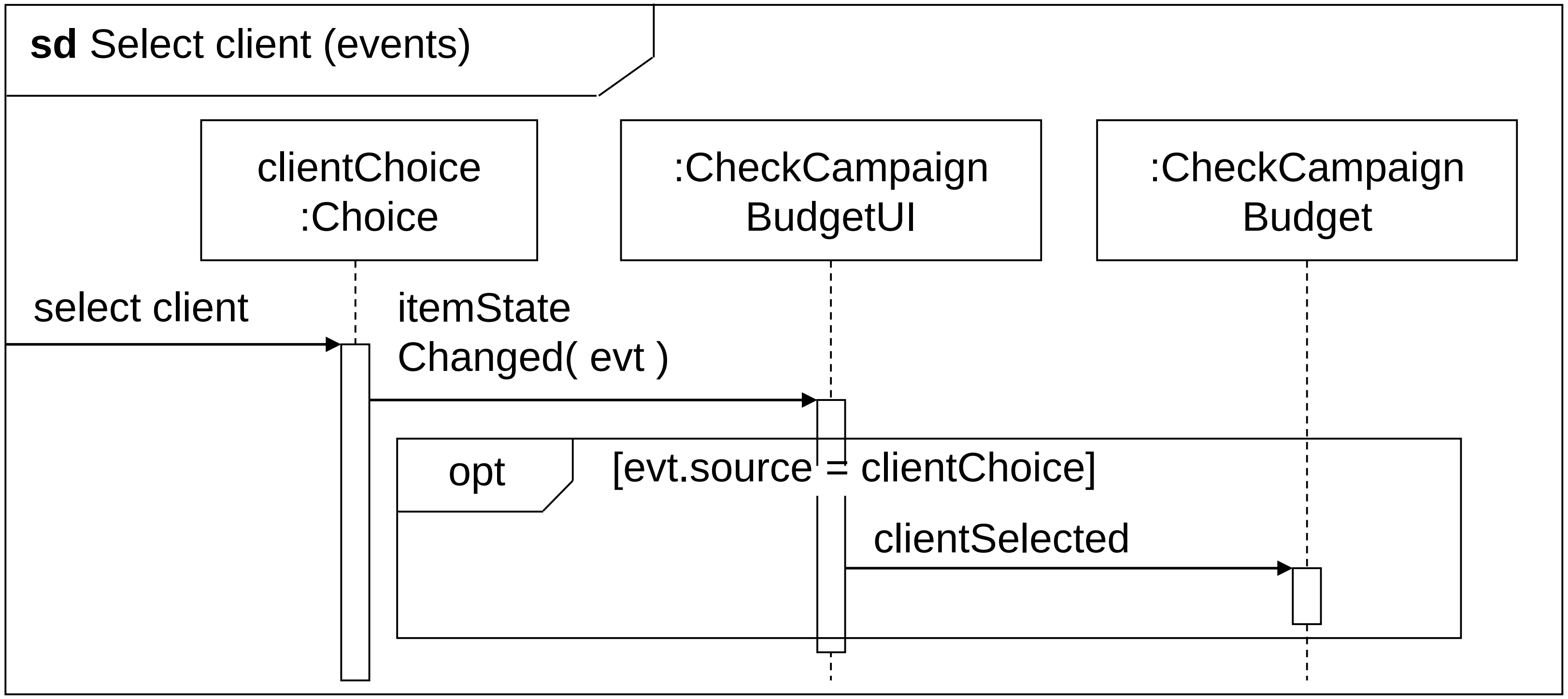


## Second Part of Sequence Diagram

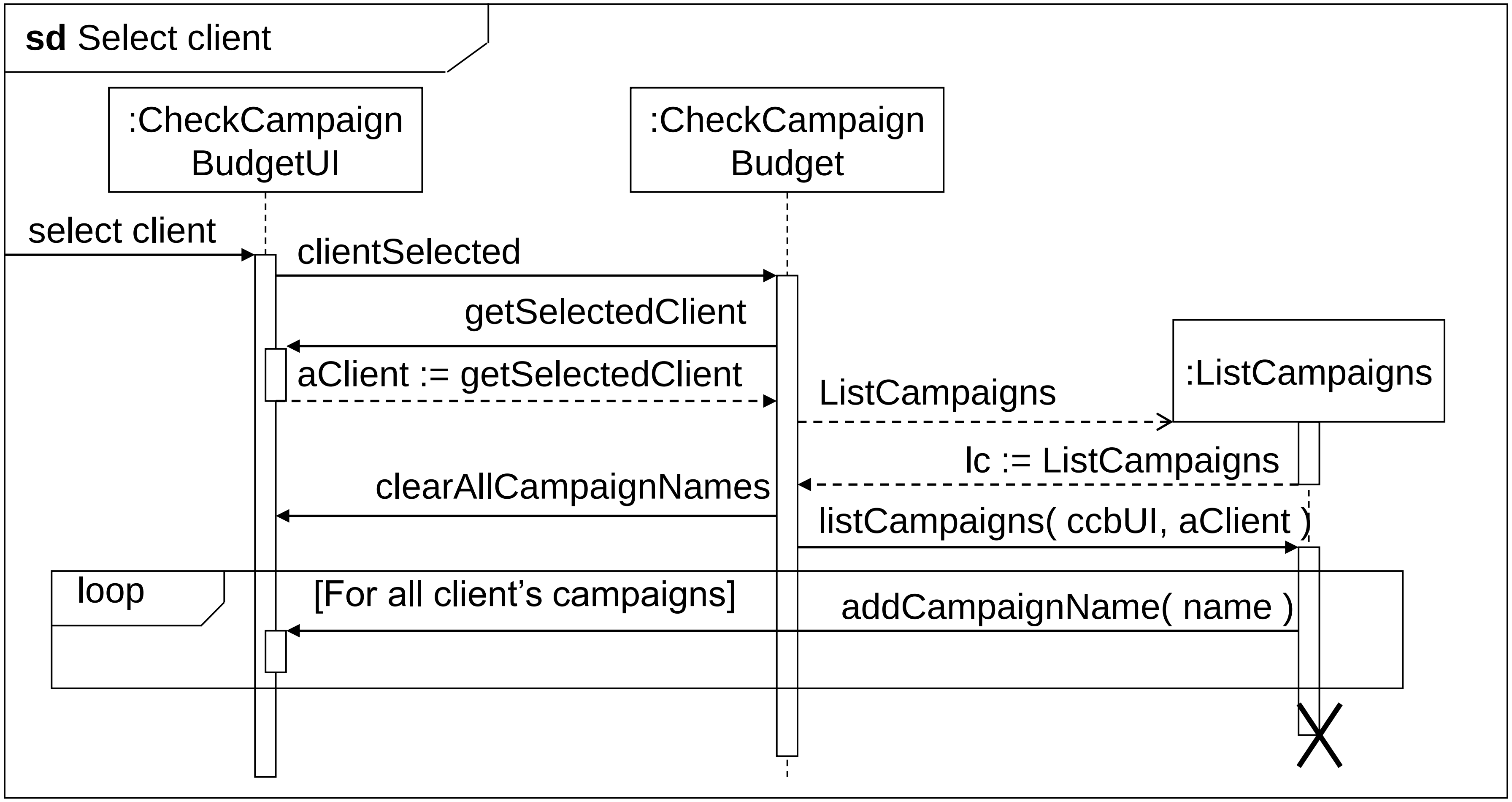


# Event-driven User Interface

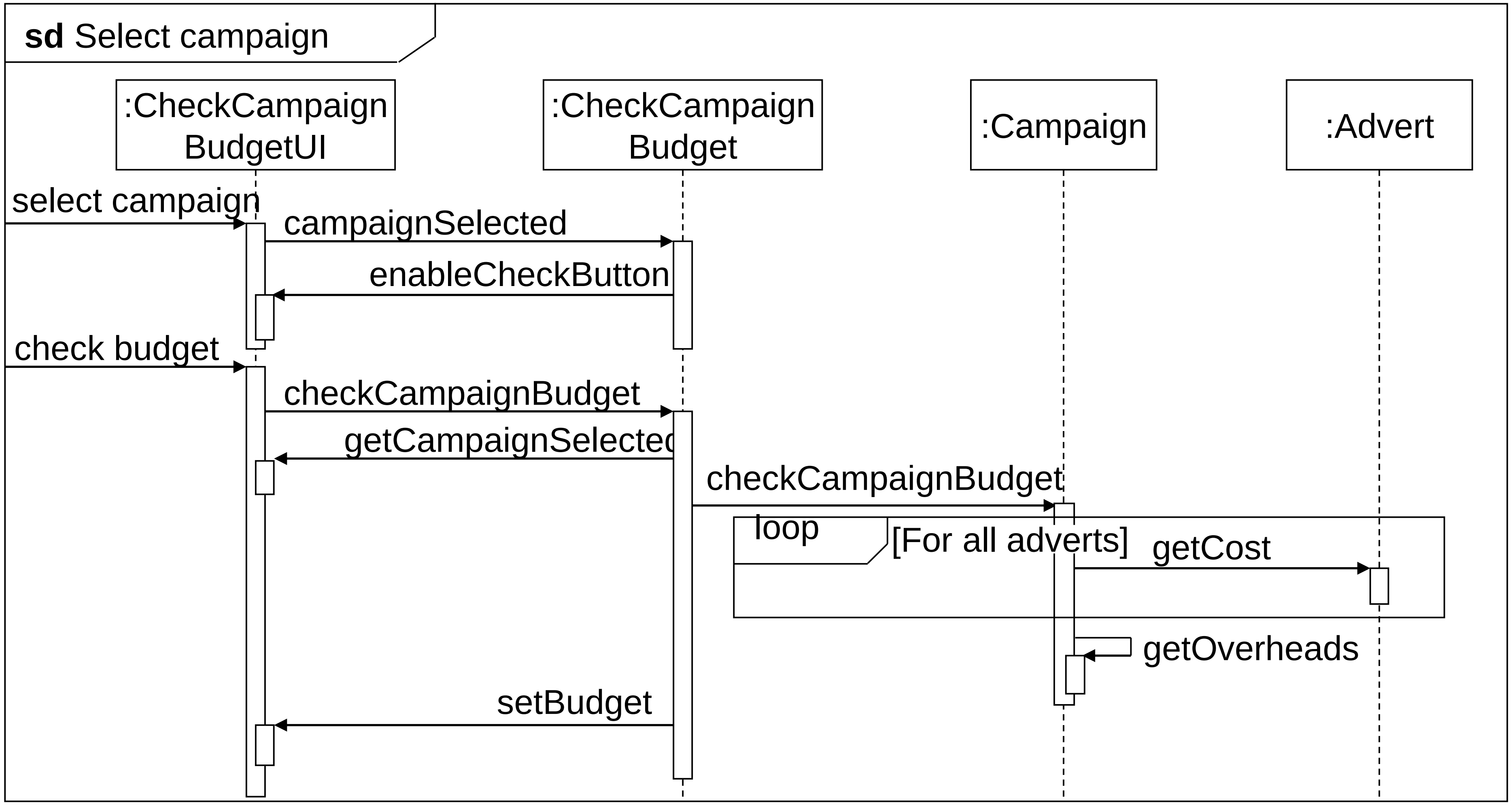
Event in <u>`:clientChoice`</u> is passed through to the main boundary class



## Revised Second Part of Sequence Diagram

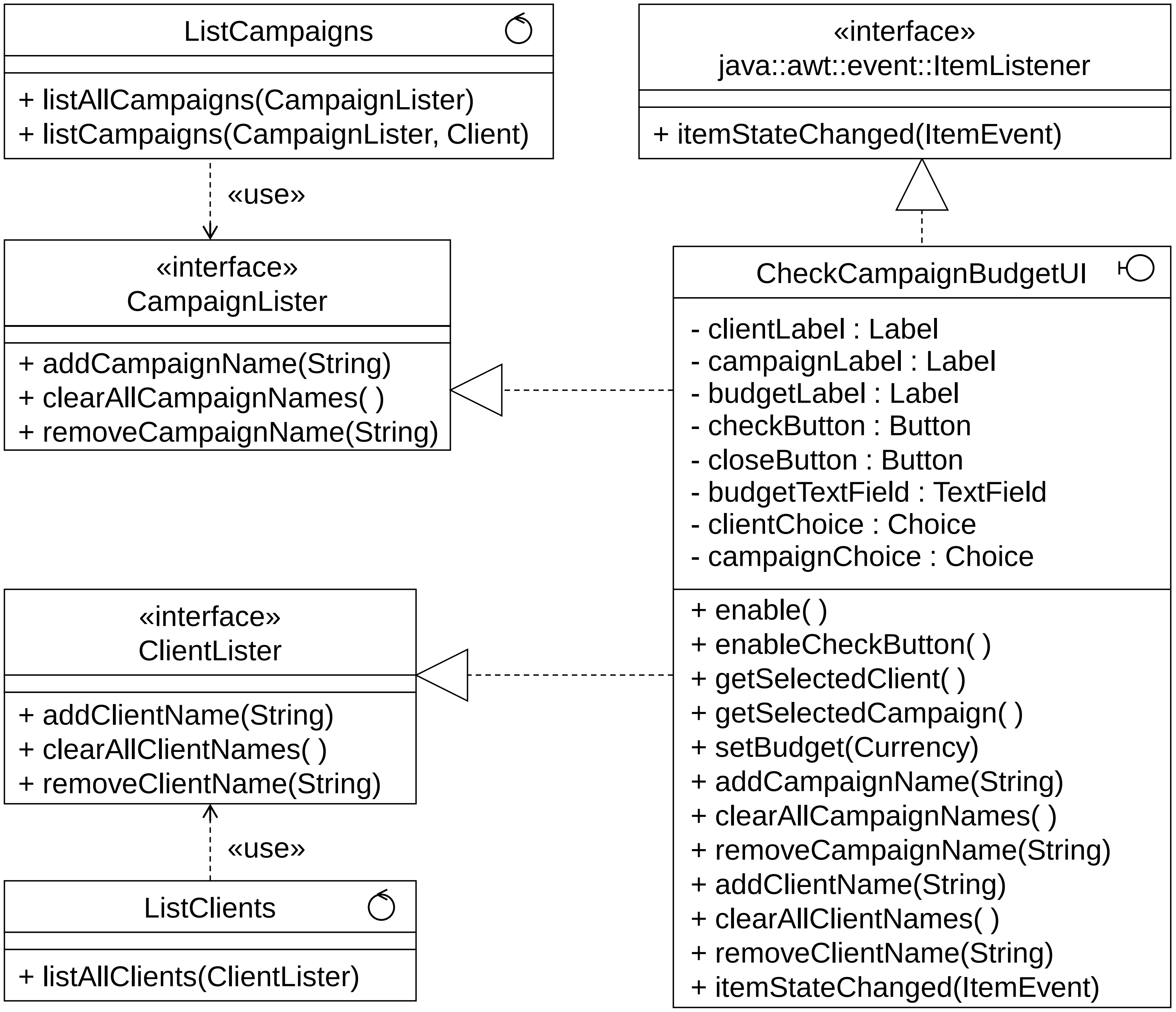


# Final Part of Sequence Diagram



## Adding to the Class Diagram

- From the sequence diagrams, we can see that the `CheckCampaignBudgetUI` class needs to implement both the `ClientLister` and the `CampaignLister` interfaces
- There are also additional operations that have been introduced, some of which will apply to any class that implements these interfaces (and therefore belong to the interfaces), and some of which belong to the `CheckCampaignBudgetUI` class


ListCampaigns
+ listAllCampaigns(CampaignLister)
+ listCampaigns(CampaignLister, Client)
«use»
«interface»
CampaignLister
+ addCampaignName(String)
+ clearAllCampaignNames( )
+ removeCampaignName(String)
«interface»
java::awt::event::ItemListener
+ itemStateChanged(ItemEvent)
CheckCampaignBudgetUI
- clientLabel : Label
- campaignLabel : Label
- budgetLabel : Label
- checkButton : Button
- closeButton : Button
- budgetTextField : TextField
- clientChoice : Choice
- campaignChoice : Choice
+ enable( )
+ enableCheckButton( )
+ getSelectedClient( )
+ getSelectedCampaign( )
+ setBudget(Currency)
+ addCampaignName(String)
+ clearAllCampaignNames( )
+ removeCampaignName(String)
+ addClientName(String)
+ clearAllClientNames( )
+ removeClientName(String)
+ itemStateChanged(ItemEvent)
«interface»
ClientLister
+ addClientName(String)
+ clearAllClientNames( )
+ removeClientName(String)
«use»
ListClients
+ listAllClients(ClientLister)

## Modelling the User Interface in State Machines

- Five tasks of Horrocks’ approach:
  - describe the high-level requirements and main user tasks
  - describe the user interface behaviour
  - define user interface rules
  - draw the state machine (and successively refine it)
  - prepare an event action table

## High-level Requirements

- The requirement here is that the users must be able to check whether the budget for an advertising campaign has been exceeded or not.
- This is calculated by summing the cost of all the adverts in a campaign, adding a percentage for overheads and subtracting the result from the planned budget.
- A negative value indicates that the budget has been overspent. This information is used by a campaign manager.

# User Interface Behaviour

- The **client dropdown** displays a list of clients. When a client is selected, their campaigns will be displayed in the campaign dropdown.
- The **campaign dropdown** displays a list of campaigns belonging to the client selected in the client dropdown. When a campaign is selected the check button is enabled.
- The **budget textfield** displays the result of the calculation to check the budget.
- The **check button** causes the calculation of the budget balance to take place.
- The **close button** closes the window and exits the use case.

## Define User Interface Rules

- User interface objects with constant behaviour
  - The client dropdown has 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 is selected or a new campaign is selected. It is not editable
  - The close button may be pressed at any time to close the window

## Define User Interface Rules

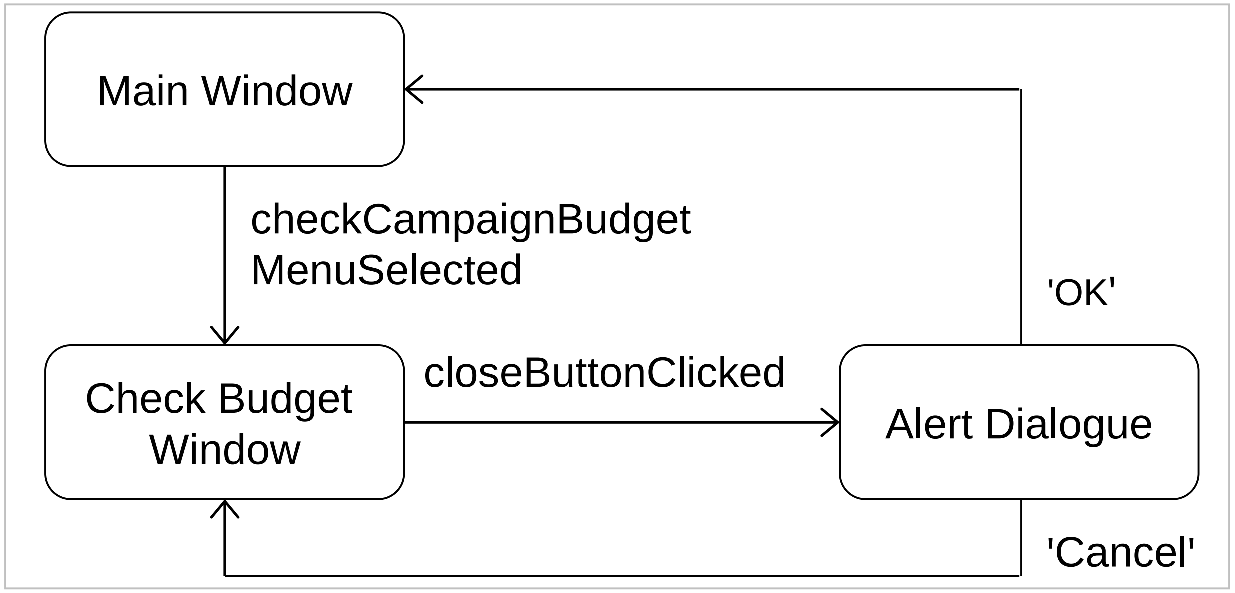
- User interface objects with varying behaviour
  - The campaign dropdown is initially disabled. No campaign can be selected until a client has been selected. Once it has been loaded with a list of campaigns it is enabled
  - The check button is initially disabled. It is enabled when a campaign is selected.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

## Define User Interface Rules

- Entry and exit events
  - 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. This asks ‘Close window? Are you sure?’ and displays two buttons labelled ‘OK’ and ‘Cancel’. 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

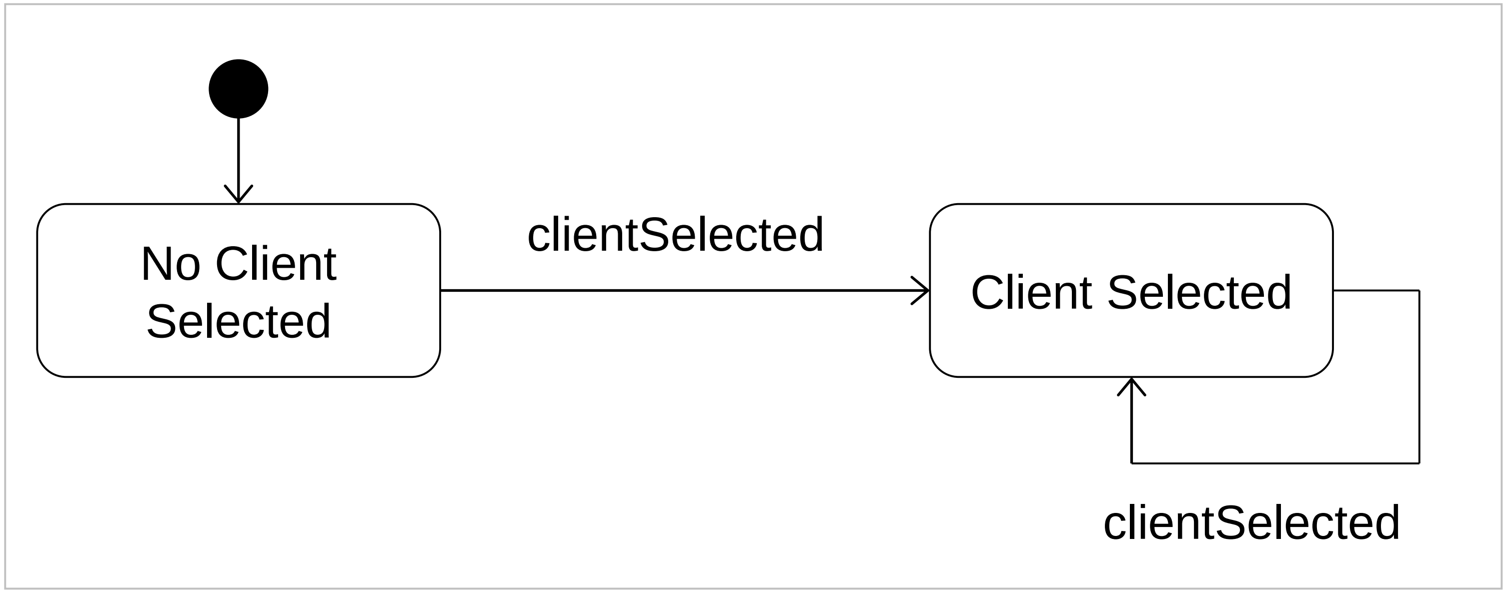
## Draw the State Machine

- We start with the top-level state machine for movement between the windows and dialogues



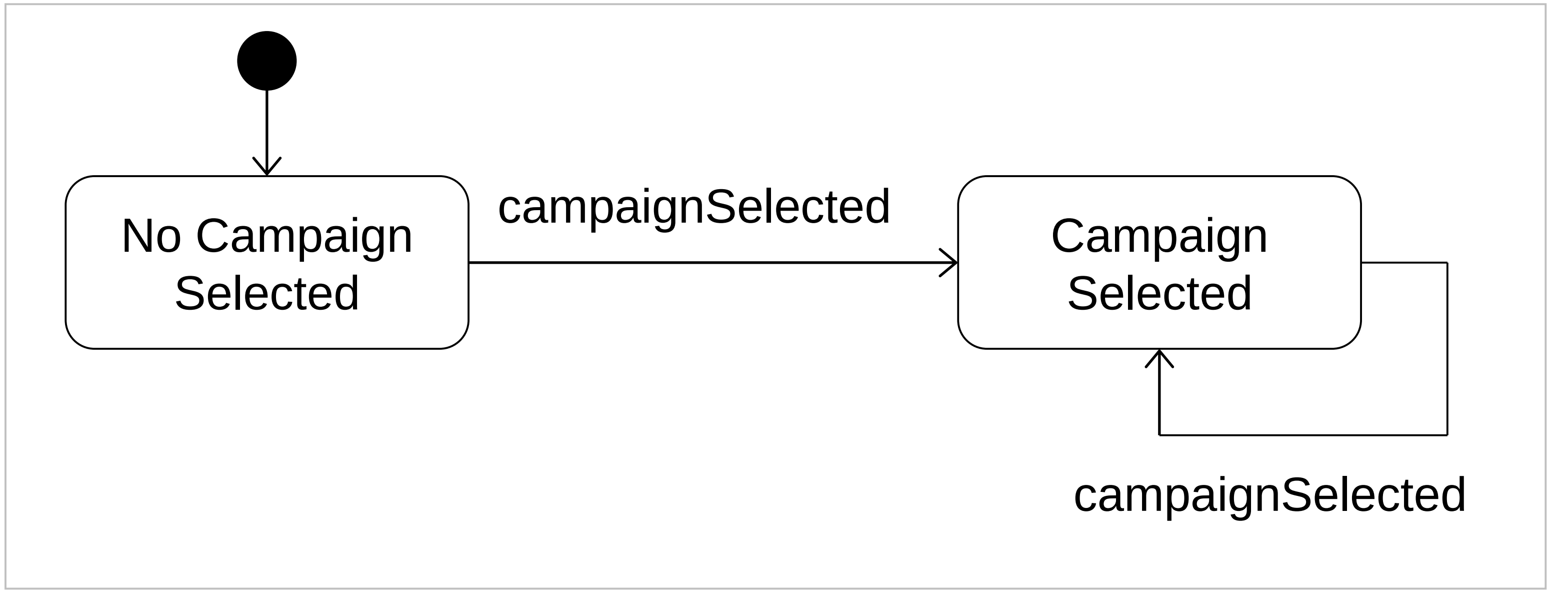
# Draw the State Machine

- Client selection states are nested within the **`Check Budget Window`** state



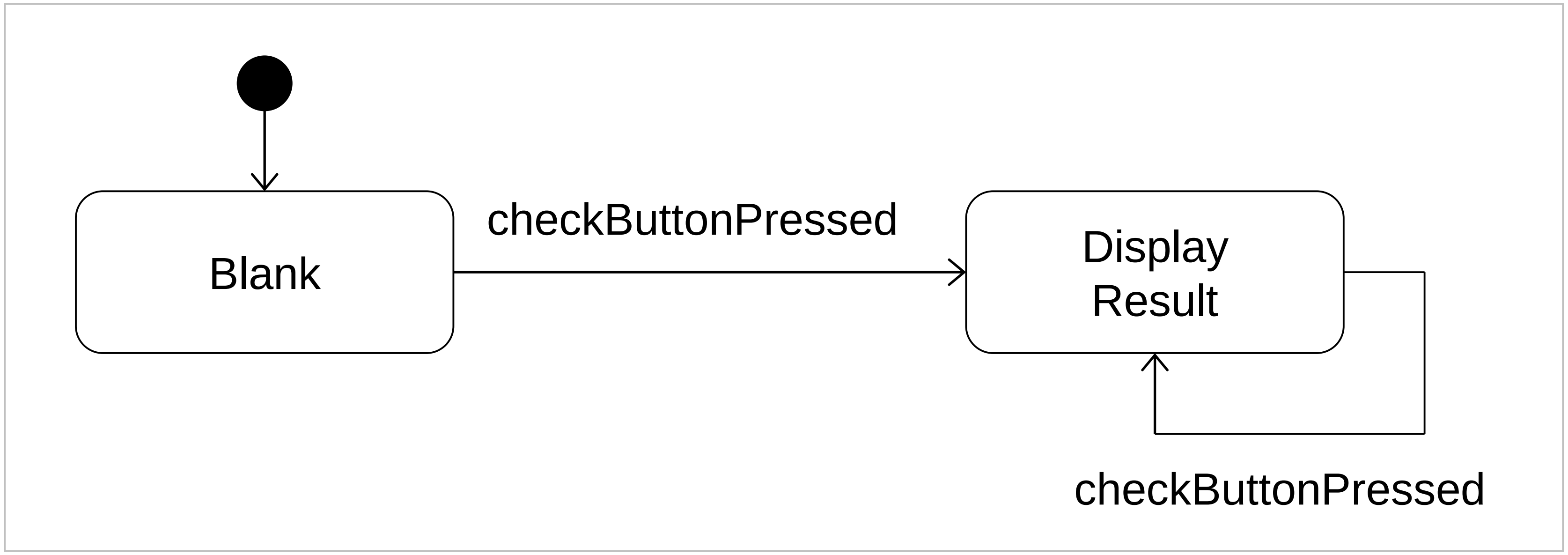
## Draw the State Machine

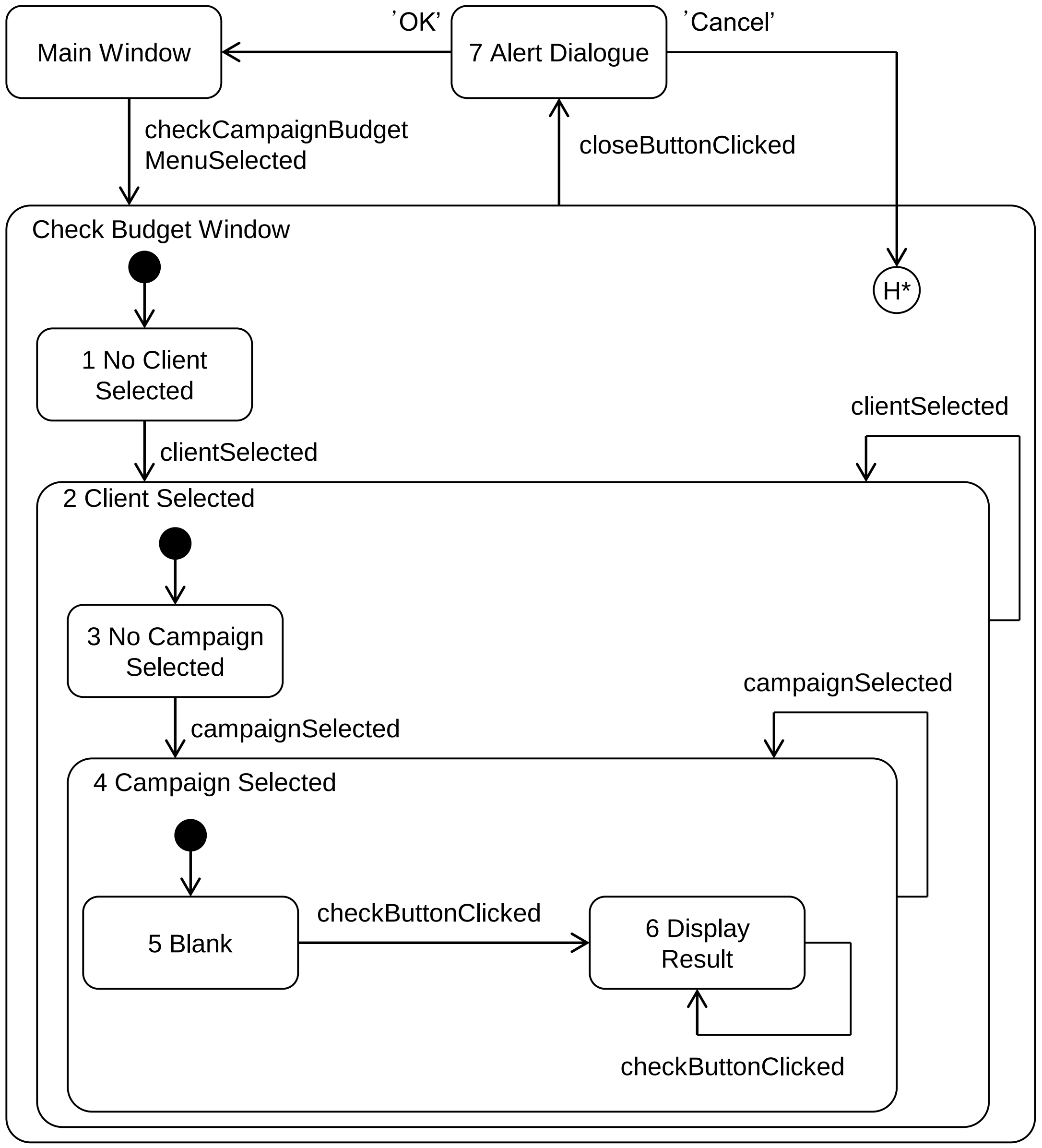
- Campaign selection states are nested within the **`Client Selected`** state



## Draw the State Machine

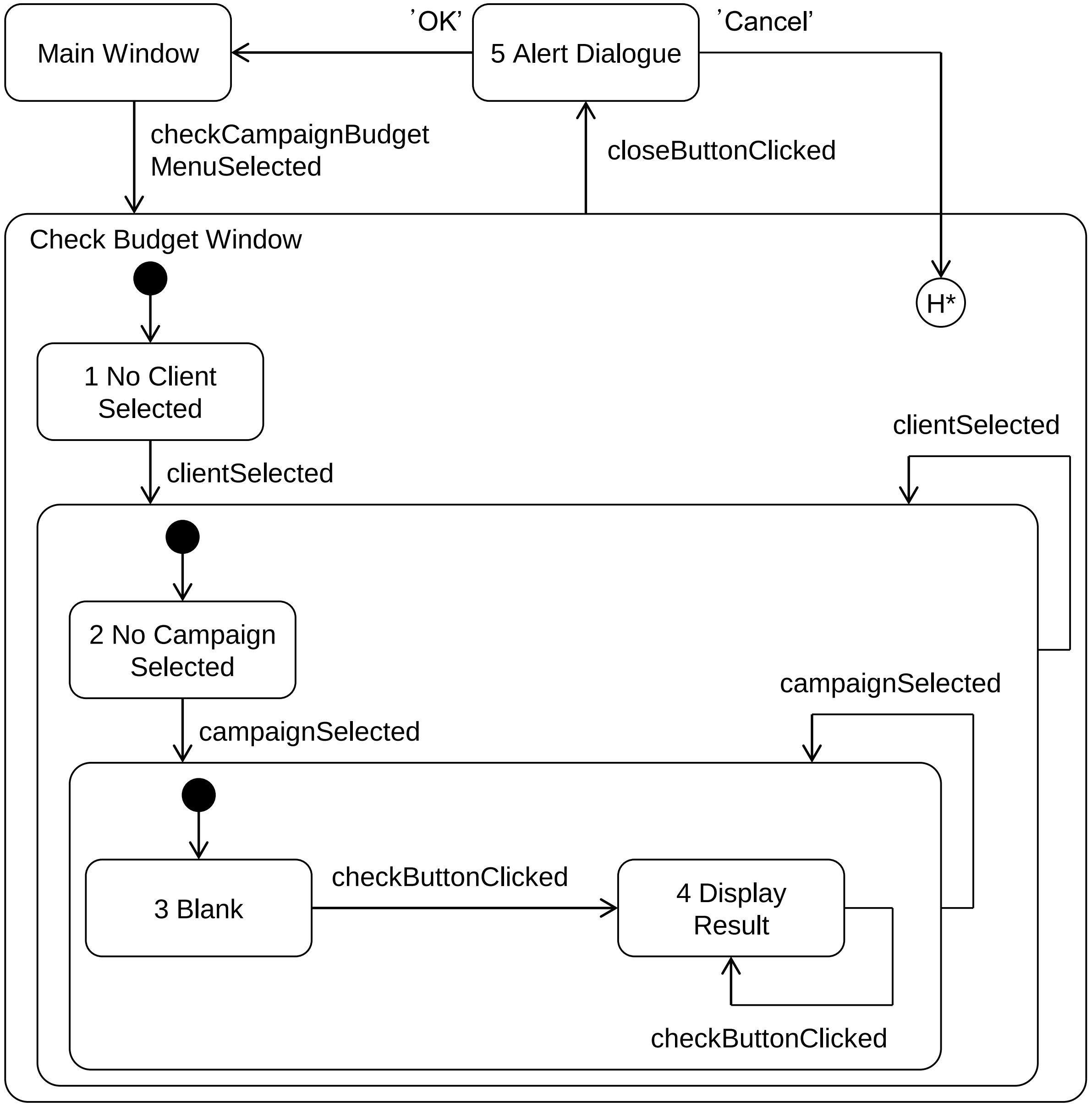
- Display of result states are nested within `Campaign Selected` state




Main Window
'OK'
7 Alert Dialogue
'Cancel'
checkCampaignBudget MenuSelected
closeButtonClicked
Check Budget Window
H*
1 No Client Selected
clientSelected
clientSelected
2 Client Selected
3 No Campaign Selected
campaignSelected
campaignSelected
4 Campaign Selected
5 Blank
checkButtonClicked
6 Display Result
checkButtonClicked

## Draw the State Machine

- State machine can be simplified (as on next slide)
- Rather than try to add all messages associated with transitions into the diagram, an Event–Action table can be used


Main Window
'OK'
5 Alert Dialogue
'Cancel'
checkCampaignBudget MenuSelected
closeButtonClicked
Check Budget Window
H*
1 No Client Selected
clientSelected
clientSelected
2 No Campaign Selected
campaignSelected
campaignSelected
3 Blank
checkButtonClicked
4 Display Result
checkButtonClicked

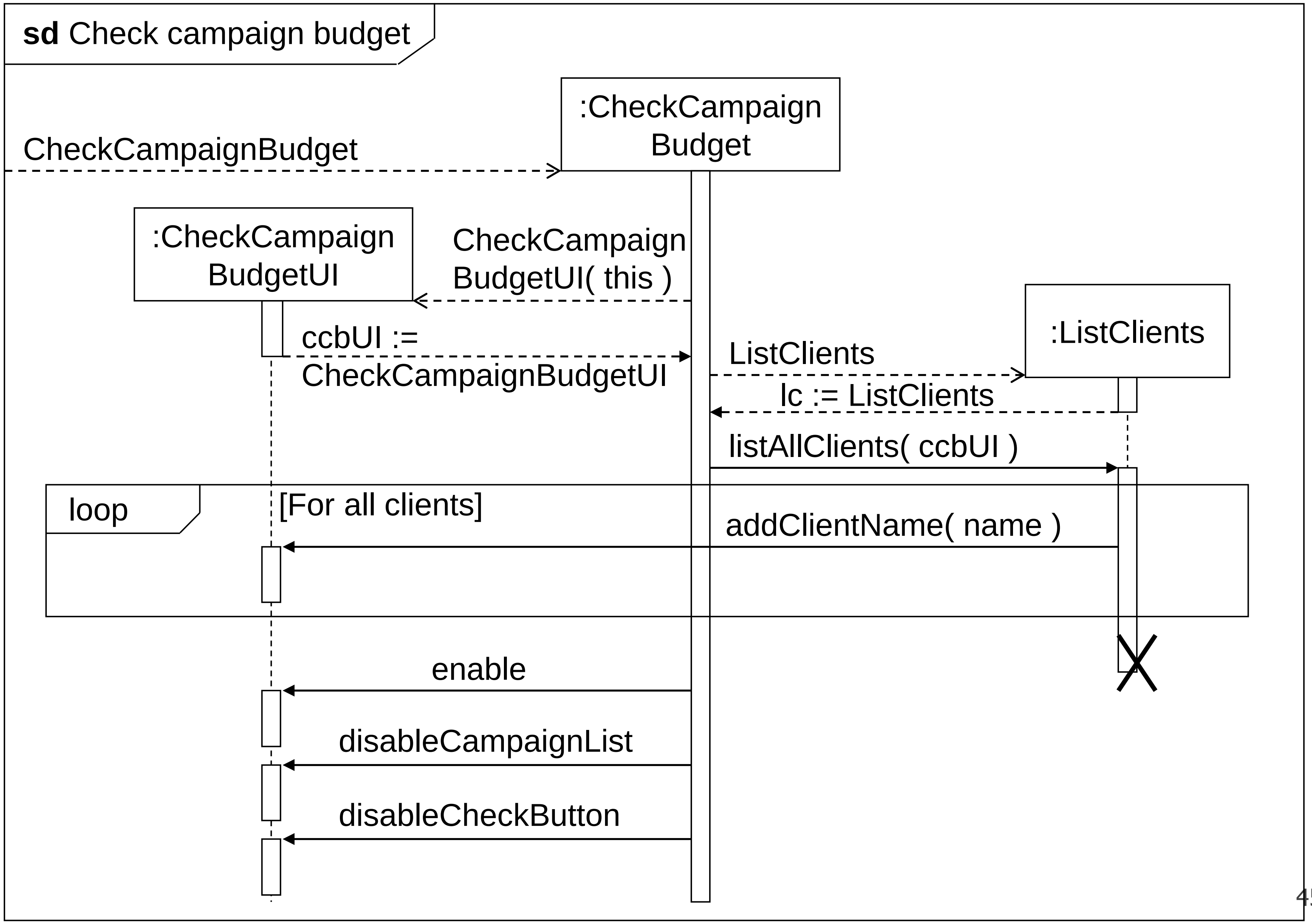
# Event–Action Table

| 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 clicked. | Calculate budget. Display result. | 4 |
| 3, 4 | Campaign selected. | Clear budget textfield. | 3 |
| 4 | Check button clicked. | 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* |

## Revising the Sequence Diagrams and Class Diagrams

- Producing the state machine and the event–action table has identified some additional messages that will be sent to the user interface to control it
- These will need to be added to the sequence diagrams and to the class diagram as operations of the UI class or of the lister interfaces

# Revised First Sequence Diagram


sd Check campaign budget
CheckCampaignBudget
:CheckCampaign Budget
:CheckCampaign BudgetUI
CheckCampaign BudgetUI( this )
ccbUI := CheckCampaignBudgetUI
:ListClients
ListClients
lc := ListClients
listAllClients( ccbUI )
loop
[For all clients]
addClientName( name )
enable
disableCampaignList
disableCheckButton