DIVISION OF COMPUTER SCIENCE

SEQUENTIAL ANIMATION OF A STRUCTURED ANALYSIS REQUIRED LOGICAL MODEL OF A VENDING MACHINE CONTROLLER

D. A. Fensome

Technical Report No.169

October 1993
Sequential Animation of a Structured Analysis Required Logical Model of a Vending Machine Controller

D.A. Fensome October 1993

1. Background
To give brief summary of research

This report summarises part of a research programme investigating prototyping methods for real-time systems. Previous work has examined Ada tasking as an animation tool for data flow models [Fensome 92], and the use of data flow to model real time system functional requirements [Fensome 93]. The present report describes work carried out in animating a Vending Machine case study using the sequential programming paradigm.

2. Structured Analysis (SA) requirement model
To summarise required logical model used for this report

The vending machine controller is described in [Fensome 93], where the requirements and logical context diagrams are shown. Appendix 1 of the present report shows the first level decomposition of the requirement model, and includes the required logical model of data flow and control flow, and the control machine.

3. Structured Analysis Control Centred Animation
To summarise the design and to present pros and cons

The animation is a conventional control centred design, shown in Appendix 2 as a top level structure chart and Ada styled program units. Only sample parts of the design are given, which are sufficient to demonstrate the important features of the design transformation. The design is carried through to asking for an event input and producing a model output, and a procedural abstraction has been used throughout.

At the top level of the control machine design, the first decisions to be made concerned the data flow between program modules, and how atomic execution of input events could be emulated. These decisions involved dealing with polled inputs, and in particular how null inputs (null events) and time outs were to be handled. I decided that the control data flow p2cev and p3cev (in cf0d0) would be a suitable coupling mechanism to the control module, if a 'null' event could be added to the design to signify no particular input event had occurred.

An automatic time-out event 'to' could be generated if time spent in, for example, the Input Control module was too long. Automatic time-out did assume an Ada environment with all the facilities of package Calendar (ie background real time clock which could be interrogated periodically). If this was not required or undesirable, then the time out could be generated by the animation user as just another input event.

The next problem was how to deal with the process activators pat3 and pat4 in cspec0. Should these be explicitly coded into the design or were these an unnecessary complication in the animation? In fact since the control machine knows whether p3 or p4 are active, and will only call the appropriate process if it is active, there seems little point in having additional variables to represent the process activators. An example of this is when the event 'mp' (minimum payment) takes the control machine from S1 to S2 and activates P3 via pat3.

One control event 'cg' ("change given") also appears to be redundant in the design. In the requirement model it is necessary to signal this event to the control machine from
P4 (see cspec0), but again is unnecessary in the sequential implementation because control will return to the control machine automatically when P4 has executed.

Overall the animation design was not as easy as might be assumed, and took much longer than expected (is any design easy?). How this might be quantified I am not sure, but I am now aware of the difficult areas i.e. polling of inputs, atomic execution emulation, relevance of requirement model communication primitives.

However the design did show that the animation user I/O was not so much a problem as expected. The design would produce a scrolled screen on the animation computer which typically might be -

Return Coins? Y/N = N  New Stock? Y/N = N  Customer Inserts = 10p
Customer selects = Cola

Product Available = Y  Product Given = Cola  Change = null

for each input event. This might seem crude, but would provide an adequate listing of event sequences and responses, in order to check for correct functionality.

4. Direct Animation with Visual Basic

To summarise design and to present pros and cons

The ability to quickly draw a user interface for the animation, and easily connect events into their related processing, was the reason why Visual Basic was used as an Animation Tool.

Visual Basic is a programming environment which provides a set of standard interface objects (e.g. boxes, buttons) on screen windows called forms, so that input events on the forms can be easily processed. The interface objects are connected into the programming environment by defined subprograms. For example the button labelled '10p' in the customer form shown in Appendix 3, is given a name Temp_In and is connected via subprogram Temp_In_Click, where 'Click' is the predefined name for the mouse click event on the button. Appendix 3 gives the complete Visual Basic listing for the CPVM animation, and shows Sub Temp_In_Click ()

There are very few design decisions using Visual Basic as an animation tool because the programming paradigm is one of input event and subsequent processing. The atomic execution semantics of the data flow model are automatically provided by Visual Basic. This is an advantage because it makes the animation easy to implement from the data flow model i.e. the programmer just provides a connection into the Basic programming environment by coding 'Click' subprograms. The control machine cspec0 can easily be implemented as a subprogram which is called when each input event occurs. Also the process activators can be implemented as global variables which can be inspected by other subprograms which implement processes. For example see process 3 implemented as Sub validate_selection. Therefore there is an easy translation mechanism from a data flow model to the animation in Visual Basic.

The CPVM Visual Basic listing in Appendix 3 does not show any time-out events, but these could have been added as 'user generated' from the customer form. Visual Basic also provides a background idle loop facility which could be used to code automatic timeout events using the general timer facilities. The Timer function however was used to generate a delay function, which was then used to display a sequence of change coins in the 'Returned' box of the customer form.

However there are some disadvantages in using Visual Basic as an animation tool, mainly in the programming facilities offered by the language. Visual Basic has some
data typing facilities but these are somewhat crude, particularly for discrete types. For example there are no integer subtypes or enumerated types, each being implemented as a standard integer type (as Booleans are too).

The modularisation facilities in Visual Basic are also somewhat crude compared to the standard Ada facilities. Global modules, modules related to forms, and general purpose modules are provided, but there is no attempt to encapsulate data or provide anything which could remotely be called object oriented facilities. This is a very negative feature of Visual Basic, especially as part of my research objectives is to look for a prototyping method which allows carry over of prototype components into production software.

Finally the syntax for subprogram calls, whilst not difficult to use or learn, also caused problems. This is because procedure calls do not use brackets around the actual parameters, whereas function calls do have brackets. However if you call a procedure with brackets by mistake, it is misinterpreted as an array reference, a local array is set up (by default), and no error is reported. This is most unfortunate!

5. Conclusions

Emulation of the requirement model semantics with a sequential animation was certainly easier than the parallel equivalent (see technical report 131) where there was a significant tasking overhead. The programming tool used for the sequential animation did make a significant difference however.

Programming in an Ada environment was relatively difficult compared with the Visual Basic environment, because

- input polling had to be designed
- some of the requirement model communication primitives were redundant

In Visual Basic however, the animation was straightforward at the design level because processing is directly connected to input events, and the programming paradigm directly mirrors the requirement model semantics. However the language as a programming tool has some disadvantages in its abstraction and encapsulation facilities. Also the benefits of the graphical interface are questionable for prototyping event oriented systems, where a tabulated list of events and consequences are required.

For future work it is proposed that the prototype programming environment should provide

- only sequential programming
- input event scanning with easy facilities to connect into the required processing
  - scrolled textual I/O
  - good data typing including subtypes and enumerated types
  - good abstraction facilities; at least data abstraction, but preferably object oriented (for ease of design transfer to production software).
References


Appendix 1

Structured Analysis Required Logical Model

Level 1 DFD

Level 1 CFD

Cspec0
entered price

Get Product Price 1

Validate Coins 2

product_data:Catalogue_Set
coins: Coinhopper
current_payment: Money

Validate Selection 3

Give change 4

total
price

selection
product available

the product

return:Object

CPVM DFD 0 Level 1

Required Logical Model
EVENT = \{ ci, mp, rc, pg, op, cg, to \}

where
- ci = coin inserted
- mp = minimum payment
- rc = return coins
- pg = product given
- op = over payment
- cg = change given
- to = time out

CPVM CFD 0    Level 1
Appendix 2

Control Centred Structured Design

a) Top Level Structure Chart

b) Ada Style Control Centred Design
CPVM Animation
Top Level Control Centred Design

Fig 1
-- Ada Style CPVM Animation
-- Control Centred Design, module control machine
-- Oct 1993    D. A. Fensome

procedure main is

  p2cev, p3cev : EVENT;
  the_product : PRODUCT;
  vendmc : STATE;

  loop
    case venmc is
      when S0 => -- Idle state
        p2cev := Input_Control;
        if p2cev = ci then
          vendmc = S1;
        end if;
      when S1 => -- Serious Customer
        p2cev := Input_Control;
        if p2cev = rc then
          vendmc = S0;
          Change_Control(null); -- Give coins back!
        elsif p2cev = mp then
          vendmc = S2;
        elsif p2cev = to then
          vendmc = S0;
        end if;
      when S2 => -- poll all inputs
        loop
          p3cev := Selection_Control (the_product);
          p2cev := Input_Control;
          if p3cev = op then -- overpaid, give change
            vendmc = S3;
            exit loop;
          -- time out, or return coins
          elsif p3cev = to or p2cev = rc then
            Change_Control(null);
            vendmc = S0;
            exit loop;
          elsif p3cev = pg then
            vendmc = S0;
            exit loop;
          end if;
        end loop;
      when S3 =>
        Change_Control(the_product);

    end case;
  end loop;

end main;
function Input_Control return EVENT is

rcev : EVENT;
new_stock : CATALOGUE;
the_object : OBJECT;
the_money : MONEY;
entry_time : TIME;

begin
  entry_time := CLOCK; -- note the time
  loop
    get_return_coins (rcev);
    if rcev /= null then
      return (rcev);
    end if;
    get_new_stock (new_stock); -- has stocker restocked/repriced?
    if new_stock /= null then
      put_new_stock (new_stock);
    end if;
    get_the_object (the_object);
    if the_object /= null then -- anything input in coin chute?
      if the_object = Slug then
        P4.put_return (Slug) -- return slug to user, not for us!
      else
        object_to_money (the_money, the_object);
        add_to_current_payment (the_money);
        add_to_coins (the_money);
      end if;
      return (is_mp); -- has minimum payment been made?
    end if;
  until Is_Time_out (entry_time, timeout);
  return (to);
end Input_control;

-- Animation user I/O
-- Ada style

function get_return_coin return EVENT is

the_char : CHARACTER;

begin
  text_io.put ("Return coins ? Y/N = ");
  text_io.get (the_char);
  return (if the_char = 'Y' then rc else null; end);
end get_return_coin;
-- Part of P4 change control

procedure put_return (the_object : OBJECT);
begin
    object_io.put (the_object);
    text_io.putline ("Returned to Customer");
end put_return;
Appendix 3
Visual Basic Animation
Customer Form
Visual Basic code
Product Available

Y

Product

Cola

50p

Slug

Returned

10p

Cola

Return Coins

Visual Basic - Customer Form
Selection Case state
Case s0
  If cev = ci Then 'coin inserted
    state = s1
  End If
  Pat3 = False
  Pat4 = False
Case s1
  If cev = mp Then 'minimum payment
    state = s2
    Pat3 = True
  ElseIf cev = rc Then 'return coins
    state = s0
    Pat4 = True
  ElseIf cev = to Then 'turn off
    state = s0
  End If
Case s2
  If cev = op Then 'over paid
    state = s3
    Pat4 = True
  ElseIf cev = pg 'product given
    state = s0
    Pat3 = False
  ElseIf cev = rc Then
    state = s0
    Pat4 = True
  End If
Case s3
  If cev = cg Then 'change given
    state = so
  End If
End Select

End Sub

Sub Return_coins_Click ()
'************
  cspec0 rc
  Give_change null
End Sub
'process 2

Sub get_coins (ByVal the_object)

***************

Select Case the_object
    Case Tenp
        current_payment = current_payment + 10
        Coins.TenpCount = Coins.TenpCount + 1
    Case Twentyp
        current_payment = current_payment + 20
        Coins.TwentypCount = Coins.TwentypCount + 1
    Case FiftypCount
        current_payment = current_payment + 50
        Coins.FiftypCount = Coins.FiftypCount + 1
    Case slug
        returned.text = "SLUG...urghh"
End Select

If current_payment >= Min_payment Then
    cspec0 mp
End If

End Sub

Sub Tenp_In_Click ()

***************
    cspec0 ci
    get_coins Tenp
    Tenp_In.Value = False

End Sub

Sub Twentyp_In_Click ()

***************
    cspec0 ci
    get_coins Twentyp
    Twentyp_In.Value = False

End Sub
Sub Fiftyp_In_Click ()
  '**************************
    cspec0 ci
    get_coins Fiftyp
    Fiftyp_In.Value = False
End Sub

Sub Slug_In_Click ()
  '**********************
    get_coins slug
End Sub

Sub Beer_button_Click ()
  '**************************
    validate_selection Beer
End Sub

Sub validate_selection (ByVal the_selection%)
  '**************************

'process 3

  Dim enough_paid%, some_there%

  If Pat3 Then
    enough_paid = Check_enough(the_selection)  'process 3.1
    some_there = Is_Product_Available(the_selection)  'process 3.2
    put_product the_selection, enough_paid, some_there  'process 3.3
  End If

End Sub

Function Check_enough% (ByVal the_selection%)
  '**************************

'process 3.1

  Dim the_cost%

    the_cost = Product_data(the_selection).CostPart
    If current_payment > the_cost Then
      cspec0 op
      Check_enough = True
    ElseIf current_payment = the_cost Then
      Check_enough = True
    Else
      Check_enough = False
    End If

End Function
Function Is_Product_Available% (ByVal the_selection)
'******************************************************************************
'Process 3.2
    If Product_data(the_selection).GoodsPart > 0 Then
        product_available.text = "YES"
        Is_Product_Available = True
    Else
        product_available.text = "NO"
        Is_Product_Available = False
    End If
End Function

Sub put_product (ByVal the_selection%, ByVal enough_paid%, ByVal product_available%)
'******************************************************************************
'Process 3.3
    If (enough_paid And product_available) Then
        the_product.text = Product_data(the_selection).ProductPart
        Give_change (the_selection)
    End If
End Sub

Sub Give_change (ByVal the_product%)
'******************************************************************************
'Process 4
    Dim payment%
    If Pat4 Then
        payment = Calc_change(the_product)  'process 4.1
        Convert_to_Coins payment       'process 4.2
    End If
End Sub

Function Calc_change% (ByVal this_product)
'******************************************************************************
'Process 4.1
    If this_product = null Then
        Calc_change = current_payment
    Else
        Calc_change = current_payment - Product_data(this_product).CostPart
    End If
End Function
Sub Convert_to_Coins (ByVal the_payment%)
'***************************************************************************

'Process 4.2
'Return change for 10/20/30p depending on coins available

    Select Case the_payment
    Case 0
        Exit Sub
    Case 10
        If Coins.TenpCount > 0 Then
            returned.text = "10p"
            Coins.TenpCount = Coins.TenpCount - 1
        End If
    Case 20
        If Coins.TwentypCount > 0 Then
            returned.text = "20p"
            Coins.TwentypCount = Coins.TwentypCount - 1
        ElseIf Coins.TenpCount > 1 Then
            Convert_to_Coins 10
            Blank_return
            Convert_to_Coins 10
        End If
    Case 30
        Convert_to_Coins 20
        Blank_return
        Convert_to_Coins 10
    End Select

cspec0 cg

End Sub

Sub Blank_return ()
'**************************
    delay 5
    returned.text = ""
    delay 5
End Sub

Sub delay (ByVal secs%)
'**************************
    start! = Timer    'gets the current time since midnight
    Do
        timenow! = Timer
    Loop While timenow - start < secs
End Sub
Sub Cola_button_Click()
    '**************************
    validate_selection Cola
End Sub

Sub Form_Click()
    '**************

' set up globals

    current_payment = 0
    Product_data(Cola).ProductPart = "Cola"
    Product_data(Cola).CostPart = 20
    Product_data(Cola).GoodsPart = 5
    Product_data(Beer).ProductPart = "Beer"
    Product_data(Beer).CostPart = 30
    Product_data(Beer).GoodsPart = 5
    Coins.FiftypCount = 5
    Coins.TwentypCount = 5
    Coins.TempCount = 5

    state = 0
    Pat3 = False
    Pat4 = False

End Sub