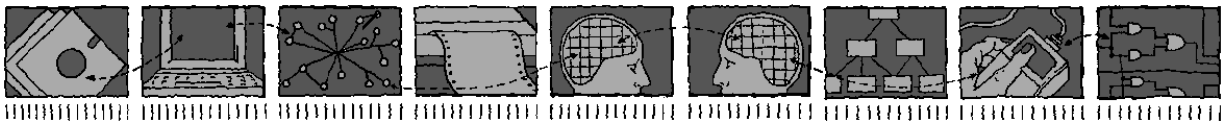
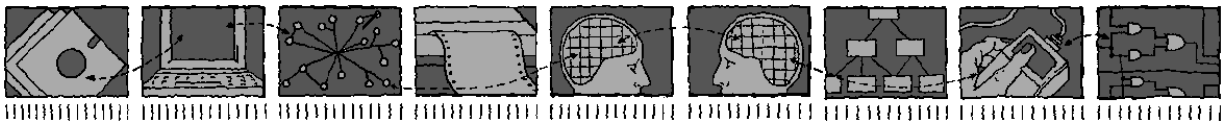


Department of Computing Science and Mathematics
University of Stirling



SOLVe: specification using an object oriented, LOTOS based, visual language

Ashley M^{c}Clenaghan

Technical Report CSM-115

January 1994

Department of Computing Science and Mathematics
University of Stirling

SOLVe: specification using an object oriented, LOTOS based, visual language

Ashley M^{c}Clenaghan

Department of Computing Science and Mathematics, University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44-786-467421, Facsimile +44-786-464551
Email amc@compsci.stirling.ac.uk

Technical Report CSM-115

January 1994

Contents

Abstract

SOLVe is a language and set of software tools developed by the SPLICE I project [Tur92]. SOLVe is experimental, it is not a mature system.

SOLVe is particularly suited for the formal specification and prototyping of interactive-systems. A SOLVe specification consists of a number of inter-communicating objects. Objects control on-screen icons. Clicking or dragging object icons send messages to the objects.

A SOLVe specification may be interactively animated. This is achieved by translating the SOLVe specification into LOTOS [ISO89], then executing the LOTOS using the `hippo` simulator. Events output by the LOTOS simulation drive icons in an X widget and, reciprocally, mouse manipulation of these icons drives the LOTOS simulation.

This document is a tutorial, user guide, and SPLICE project report on SOLVe.

Key phrases: requirements capture and prototyping, LOTOS generation, object-oriented, interactive-systems.

Chapter 1

Introduction to SPLICE

1.1 Context

The requirements capture and specification activity is a difficult, yet extremely important part of system development. Errors in the top-level specification have a major impact on later refinements. The SPLICE project conjectures that the requirements capture activity can benefit from an injection of formalism.

1.2 Objectives

The objective of the SPLICE project [Tur92] is to develop methods and software tools that support the use of LOTOS [ISO89, Tur93b, Tur89] for requirements capture activities in a number of selected application domains. The aim of the project is to make the benefits of formal specification accessible to non-formalists. The methods and tools will present LOTOS specifications in a fashion which is intelligible to users untrained in the use of LOTOS. SOLVe is one thread of SPLICE research work.

1.3 Related work

Previous projects on requirements capture methods include CORE (Controlled Requirements Expression), FOREST (Formal Requirements Specification, Alvey SE/015) and FORSITE (A Specification Support Environments, Alvey SE/065). CORE was concerned with a method rather than tools, and had no (mathematically) formal basis. FOREST and FORSITE were primarily concerned with developing tools that directly manipulated formal notation. The SPLICE conviction is to have the tool user indirectly manipulate the formal representation of the requirements via a familiar, front-end language. In this way the user can generate formal requirements specifications without having to learn an esoteric formal notation.

1.4 Progress

SPLICE has chosen to look at three application domains: OSI services, neural networks and interactive-systems. Prof. Kenneth J. Turner has been investigating the OSI services thread. He presented some results from his investigation in an invited paper presentation [Tur93a] and has developed an experimental tool which supports OSI service concepts, in the m4-macro language. Dr. J. Paul Gibson has been developing a tool, using the Sather language and Sunview, for the LOTOS based specification of neural networks [Gib93b]. This document reports on the work done by the author on the specification and prototyping of interactive-systems using LOTOS.

Chapter 2

What SOLVe is

SOLVe is:

- A language for specifying and prototyping the requirements of interactive-systems. When we talk about interactive-systems we mean the human-oriented interface-systems to machines such as VCRs, hi-fi systems and TVs.
- A set of software tools for manipulating and animating SOLVe specifications.

SOLVe incorporates three important concepts:

- object-oriented specification
- interactive animation
- formal specification.

This is a snapshot of an example SOLVe session where SOLVe has been used to specify a simple system consisting of a switch and a light.
Graphical icons represent the switch object and the light object. An object's icon may reflect its state. The SOLVe user may drag or click on an icon to effect a change in the object's state (e.g. clicking the switch icon will toggle the switch object between the states on and off).
The specified system's behaviour unfolds as sequences of messages between objects. These messages are represents by suitably structured LOTOS events.

Chapter 3

Related work on interactive-systems

3.1 Background

Commercially available tools for requirements capture for interactive-systems tend to be sophisticated graphics editors. These systems help users collect and organize their thoughts about the visual attributes of the interactive-systems. However the descriptions which these tools produce are often inadequate models of the behavioural requirements, cannot be used as executable prototypes, and lack the rigour needed for testing and refinement.

The production of tractable specifications from the requirements capture activity is a desirable precursor for a formal development strategy. Moreover, in a world ever more cluttered by "push-button" devices (the TV/hi-fi/... interactive-systems targeted by SOLVe), the importance of generating analysable models at the early, requirements capture stage takes on particular significance. Thimbleby [Thi93a, Thi90] describes the ramifications of poorly conceived "push-button" devices and demonstrates how building formal models of these leads to early problem identification and promotes better designs.

3.2 Existing tools

The QUICK system [DDN92] inspired and shaped this work. QUICK (Quick User Interface Construction Kit) is a toolkit which allows non-programmers to construct and exploring graphical direct manipulation interfaces. In particular we have borrowed the ideas nurtured by QUICK of "prototypical objects", animation language, and response to graphical manipulation. QUICK concentrates on graphical presentation and manipulation. SOLVe uses graphical presentation and manipulation to convey the meaning of a specification but, unlike QUICK, SOLVe's primary concern is to deal with formal specifications. Other related work includes: XIT [HHR92, HHR93], STATEMENT [HLN+90] (also [HdR91]), Hyperdoc [Thi93b, Thi93a, Thi90] and work by [Naf91]. SOLVe has evolved from work under the name ReCap-IS [McC93] done by the author. (Also see [McC94].)

Chapter 4

The SOLVe system

SOLVe is designed to be used by people who are not familiar with formal languages (in particular LOTOS). SOLVe is a system for building formal requirements specifications using a simple object-oriented language, and for exploring these specifications using *interactive animation*.
SOLVe is suitable for requirements capture in design domains in which systems are interactive (producing feedback in response to user input) and can be represented by graphical animation. The author has applied SOLVe to a number simple example systems, including: a VCR on-screen controller, a database access abstraction, a light switch system, and a visualization of a message passing communications protocol.

The challenge faced by SOLVe is how to provide users with little training in LOTOS a means to:

- establish a (formal) LOTOS specification of the requirements
- explore requirements specifications.

SOLVe meets this challenge by:

- allowing the user to write the requirements specification in a simple, intuitive object-oriented language which is then automatically translated into LOTOS
- graphically animating the LOTOS translation, allowing the user to explore the behaviours of the specification by interacting with the animation (e.g. by clicking or dragging object icons).

Chapter 5

Object-orientation

The SOLVe notion of object-orientedness includes:

- an object is an autonomous entity with well defined interface methods
- objects communicate via blocking or non-blocking message passing
- an object can decide its future behaviour dependent on internal values and communications with other objects
- a *simple* object has an *icon* — a visual representation maintained by the object
- interaction with the environment is in terms of message passing, e.g. when the user clicks on an *icon* this is interpreted as a message being sent to the `IconClicked` method of the object responsible for the icon
- a *composite* object is a set of interworking *simple* or *composite* objects.

A key feature of the SOLVe system is *visualization*. Each object is visualized as an *icon* — a displayed bitmap. An object is responsible for displaying and modifying its own object icon. An object icon can be used to represent an abstraction of the state of an object or some part of the total system under design. The object icons visually convey to the SOLVe user the behaviour of the system under design.

Object-orientation sometimes supports "class" or "type" based objects [CW85], as opposed to the "prototypical" objects [DDN92] supported by SOLVe object-orientation. (The distinction between these two schools is clearest when looking at how objects are coded.) The key idea of class-based inheritance is top-down specialization, whereas the key idea of prototypical objects is bottom-up composition of objects from simpler objects. Each SOLVe simple object is created with all the characteristics of a basic prototypical object. A created object can be edited to customize it. Using the SOLVe editor, the user may amass a construction kit of predefined simple and composite objects. [DDN92] point out that prototypical objects support the two often quoted advantages of inheritance: abstraction and reuse. In a prototypical paradigm objects may be aggregated and the aggregate labelled to form a new abstract "class" or "type". Objects may be duplicated thus supporting reuse.

Other works relating object-oriented ideas and LOTOS include [vH89, Rud92, Gib93a, MC93].

Chapter 6

The SOLVe object-oriented model

This chapter outlines the SOLVe object-oriented model.

6.1 Objects and messages

A SOLVe specification consists of a number of objects which inter-communicate via messages. A message either invokes an object method, or returns the value results of an invoked method. Objects execute concurrently.

One of the objects in an executing SOLVe specification is an object called `Interface`. The `Interface` object is implicit — it is declared and defined by the SOLVe system. In contrast, all the other objects in a SOLVe specification have to be declared and defined explicitly by the user. Figure 1 illustrates this architecture.

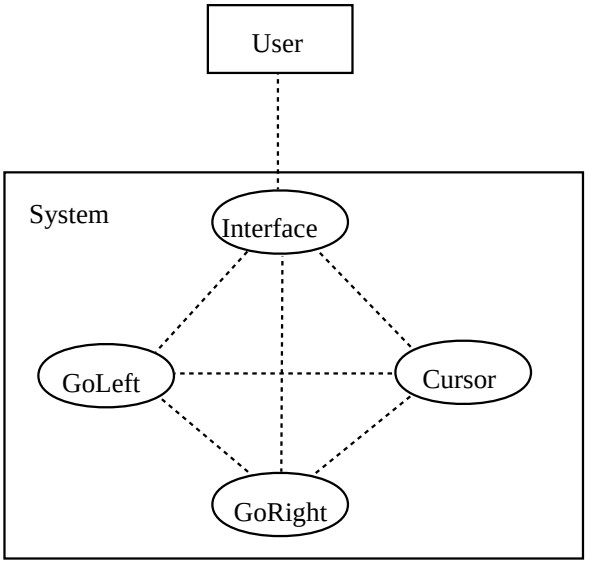


Figure 1: Inter-communicating objects in a SOLVe specification

The specification in figure 1 consists of the implicit object `Interface` and the 3 user-declared objects `GoLeft`, `GoRight` and `Cursor`. Potential message communication paths are depicted by

the dotted lines. These show that all objects may communicate directly with one another. (This is at the SOLVe language level; in chapter 10.1 we shall see how these direct message paths are translated into indirect paths – via the `ObjectComms` process – at the LOTOS language level).

All communication between an executing SOLVe specification and the user is via the `Interface` object which is responsible both for displaying object icons to the user, and for accepting click and drag requests from the user.

Figure 2 provides a typical example of an interaction between an executing SOLVe specification an its user.

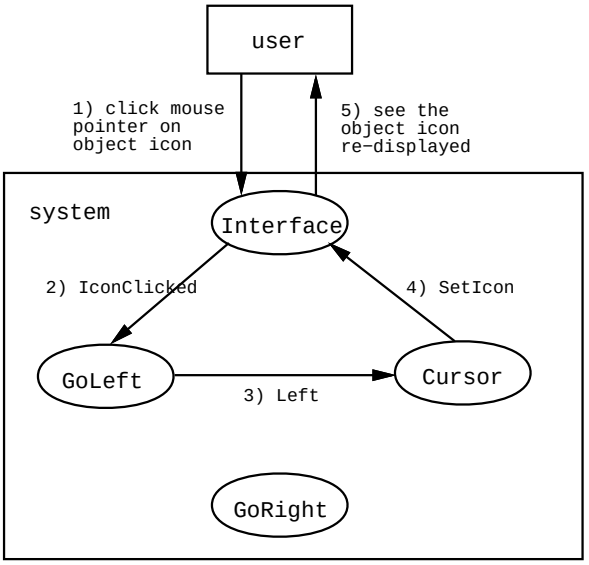


Figure 2: A typical message sequence

1. The user initiates the interaction by clicking the mouse over the icon representing the `GoLeft` object.
2. This X display event is handled by a method within the `Interface` object which sends an `IconClicked` message to the `GoLeft` object, notifying the `GoLeft` object that its icon has been clicked.
3. Then `GoLeft`'s `IconClicked` method sends a `Left` message to the `Cursor` object.
4. The `Cursor`'s `Left` method responds issuing a `SetIcon` message to the `Interface` object, requesting the `Interface` object to redisplay the `Cursor` icon one place to the left.
5. Responding to this request the `Interface` object instructs the X display to redisplay the `Cursor` icon in its new position.

6.2 Instance parameters and methods

In a SOLVe specification all objects must be first declared and then defined (with the exception of the `Interface` object). The object declaration block provides sort information used to check assignment statements, method call parameters, etc. occurring within the object definition block.

Each object has a list of instance parameters and a set of methods. When users declare an object they declare the sorts of its instance parameters. They also declare the names of the object's methods together with the invocation and return parameter sorts of these methods.

In addition to the declared instance parameters, each object has three implicitly declared instance parameters called `xPos`, `yPos`, and `iconPic` of sort `Int`. These three parameters are used to represent the x,y co-ordinates and the bitmap picture of the object's icon.

In addition to the declared methods, each object has three implicitly declared methods called `Initialize`, `IconClicked` and `IconMoveRequest`. The `Initialize` method of each object is invoked immediately the SOLVe specification starts to execute. The `Initialize` method may be used to assign initial values to an object's instance parameters and to initialize the object's icon. The `IconClicked` and `IconMoveRequest` methods of an object are invoked when the user either clicks on or attempts to drag the object's icon. Although these three methods are implicitly declared for each object, the specifier must explicitly define these methods for each object.

Chapter 7

The SOLVe language

The SOLVe language is simple to use. We have tried give it the feel of a simple, intuitive object-oriented programming language, deliberately moving away from the esotericness and algebraic feel of LOTOS. SOLVe object-oriented descriptions can be automatically translated into LOTOS specifications. Some of the mechanics of the SOLVe object-oriented system are coded in the LOTOS specification resulting from the translation. This is a disadvantage with respect to the abstractness of the resulting LOTOS specification, but it supports object-oriented requirements capture and it allows interactive animation of the resulting LOTOS specifications. (See appendix C for a formal definition of the syntax of the SOLVe language.)
An outline of a SOLVe description looks like this:

```
System <name> Is

   PicDecls
      -- icon picture declarations
      leftArrow, rightArrow
   EndPicDecls

   ObjectDeclarations
      <object declarations>
   EndObjectDeclarations

   ObjectDefinitions
      <object definitions>
   EndObjectDefinitions

EndSystem
```

The user is expected to declare a list of the icon picture (bitmap) identifiers to be used in the specification. These identifiers ought to be file names (without their `.btm` extensions) from the SOLVe bitmap directory (see appendix B.1). The identifiers declared in the `PicDecls` statement are given the sort `Int`. They are implicitly assigned successive `Int` values starting at `0` for the first declared `PicDecls` identifier.

An object declaration describes the instance parameter sorts, names and method parameters sorts of each object.

An object definition describes the inner details of the object. The definition must satisfy the object's declaration and define the declared methods including the implicitly declared methods `Initialize`, `IconClicked` and `IconMoveRequest`.

```
Object Cursor(Bool) Is Left()() Right()() QueryXPos()(Int) EndObject
Object GoLeft() Is EndObject
```

Above are two example object declarations. The object `Cursor` has the three explicitly declared methods `Left`, `Right` and `QueryXPos` as well as the three implicitly declared methods `Initialize`, `IconClicked` and `IconMoveRequest`. The object `GoLeft` has only the three implicitly declared methods.

The two sets of parentheses after each method declaration are used to declare the sorts of the method's invocation and return parameters. The methods `Left` and `Right` have no invocation or return parameters, while the method `QueryXPos` has no invocation parameters but has one return parameter of the sort `Int`.

The parenthesis after an object name are used to declare the types of the object's instance parameters. The `Cursor` object has one explicitly declared instance parameter of type `Bool`, as well as the three implicitly declared instance parameters `xPos`, `yPos` and `iconPic` all type `Int`. The `GoLeft` object has only the three implicitly declared parameters.
Listed below is the minimum definition for the `Cursor` object given its declaration above.

```
Object Cursor(Bool:flashingOn) Is

   Method Initialize() Is
      -- This is a minimal definition of this method. Other SOLVe language
      -- statements may be used in place of this comment to define more
      -- functionality for this method. (This applies to the below methods also.)
      Return()
   EndMethod

   Method IconClicked() Is
      Return()
   EndMethod

   Method IconMoveRequest(Int:newXPos,Int:newYPos) Is
      Return()
   EndMethod

   Method Left() Is
      Return()
   EndMethod

   Method Right() Is
      Return()
   EndMethod

   Method QueryXPos() Is
      Return(xPos)         -- xPos is of sort Int which satisfies the
   EndMethod                --          declaration of the return parameter types
                                  --          of this method.
EndObject
```

The variable `flashingOn` appears in the object's instance parameter list satisfying the declaration of one explicit instance parameter of type `Bool`. The implicit instance parameters `xPos`, `yPos` and `iconPic` should not occur in the instance parameter list of the definition. All the instance parameters (explicit and implicit) of an object may be referenced or assigned to within the object's methods. An instance parameter maintains its assigned value between methods invocations until it is reassigned some other value.

Behaviour within any single simple object is sequential. This means that an object may only be executing one method at any time.

The invocation and return parameter lists of the methods also satisfy the object's declaration information.

The above definition provides the minimum definition of the `Cursor` object given its declaration. The method definitions may be customized using any of the SOLVe language statements listed below.

```
Assign(<variable>,<value>)
```

The `Assign` statement is used to assign a value to a variable. The SOLVe language supports values and variables of the sorts `Int` (integer) and `Bool` (boolean). The `Int` operators are `Succ`, `Pred`, `Plus`, `Minus`, `Eqi`, `Nei`, `Le`, `Lt`, `Gt` and `Ge`. The `Bool` operators are `Not`, `Eqb`, `Neb`, `And` and `Or`. The `Int` values are `..., -2, -1, 0, 1, 2, ...`. The `Bool` values are `True` and `False`. SOLVe pre-defines the four `Int` constants `XMIN`, `XMAX`, `YMIN`, `YMAX`. These define the limiting co-ordinates of the icon screen, and may be used like `Int` values.

```
AskWaitCall <object name>.<method name>(<invocation values list>)
                                                (<return variables list>)
```

The `AskWaitCall` statement is used to invoke a particular method of a particular object with a list of particular invocation values. An `AskWaitCall` blocks execution in the invoking object until the call returns to assign values to the variables in the return variables list (these variables must have been declared earlier). In effect the execution of an `AskWaitCall` statement sends a request message to an object and waits for a reply message. An object can use an `AskWaitCall` if it wants to invoke a method and ensure the completion of this method before continuing its behaviour (see figure 3).

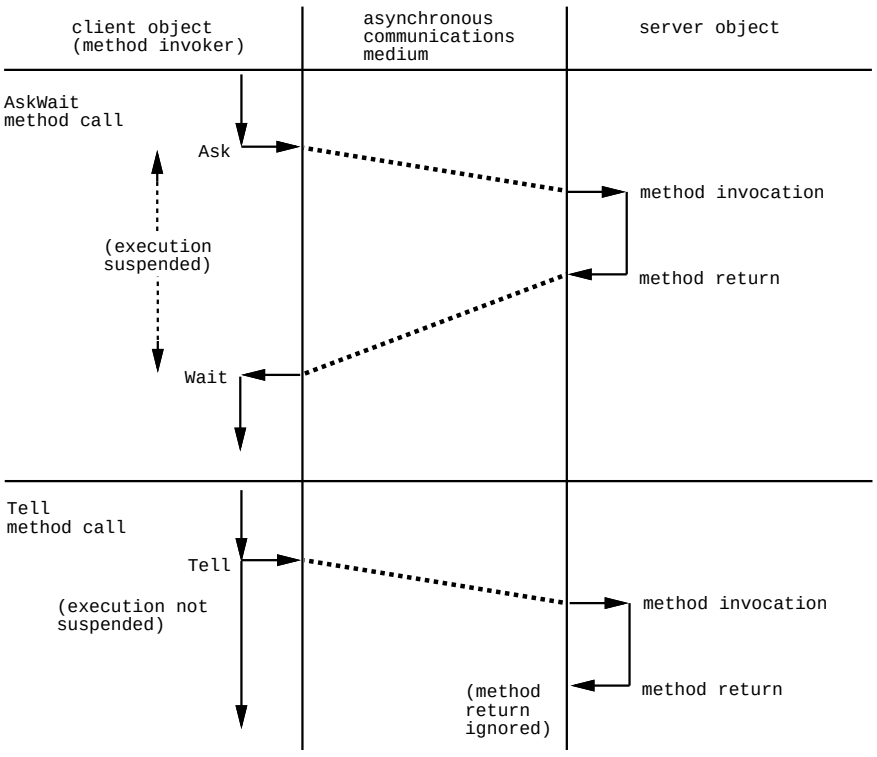


Figure 3: A blocking `AskWaitCall` and a non-blocking `TellCall`

```
TellCall <object name>.<method name>(<invocation values list>)
```

The `TellCall` statement is used in a similar way to the `AskWaitCall` statement except that it does not block execution in the invoking object. Any parameters returned by the invoked method are ignored. An object can use a `TellCall` if it wants to invoke a method and immediately continue its behaviour, not waiting for the completion of the invoked method (see figure 3).

```
If <condition> Then <statement> Else <statement> EndIf
```

The `If` statement provides simple conditional branching.

```
While <condition> Do <statement> EndWhile
```

The `While` statement provides a simple iterative loop structure.

```
Variables Int:exampleVarOne, Bool:exampleVarTwo EndVariables
```

The `Variables` statement provides a means to declare local variables with a lifetime of that of the enclosing method definition and with a scope confined to the statements following within the method definition.

```
-- <any comment here>
```

Comments are delimited by `--` and the end-of-line character.

Chapter 8

Using SOLVe: an example

This chapter describes how to use SOLVe by applying it to a example. SOLVe is used to capture the requirements of a simplified on-screen control system for a VCR.

8.1 Informal requirements

The informal requirements for the VCR on-screen control are as follow:

1. The system allows the VCR user to set an on-screen 24 hour clock using an on-screen cursor manipulated via cursor control buttons. (Initial requirements statement.)
2. The 24 hour clock is represented by two on-screen digits representing tens of hours and units of hours. (Simplifying assumption.)
3. An increment (decrement) button allows the data value (clock digit) pointed to by the cursor to be incremented (decremented). (Derived requirement.)
4. It should not be possible to move the cursor to unmeaningful screen positions where the cursor is pointing at unmodifiable data. (Refined requirement.)
5. It should not be possible to modify the clock to a value outside the range 0–24. (Refined requirement.)
6. Incrementing the least significant clock digit when the clock displays, say, `09` will result in the new display `10` (and vice versa for decrementing). (Refined requirement.)

8.2 The SOLVe specification

The SOLVe specification listed in appendix D captures the informal requirements of chapter 8.1.

Normally the SOLVe editor tool (described in chapter 9.1) would be used to write SOLVe descriptions. This tool would provide an environment for checking the syntax of the description incrementally as it is written, and would provide access to a construction kit of objects predefined for the particular problem domain at hand. However, at present this tool is still to be developed, and we have edited this simple SOLVe description using `vi`.

The mapping between the informal requirements and the SOLVe specification is straightforward. The following table shows the objects, attributes, methods, etc. which have been identified from analysing the informal requirements.[1]

Object	Category	Attributes	Methods provided	Methods called
Units	on-screen entity	digit value	Inc Dec	TensDigit.QueryValue
Tens	on-screen entity	digit value	Inc Dec QueryValue	
Cursor	on-screen entity	X position	Left Right QueryXPos	
GoLeft	push-button entity		IconClicked	Cursor.Left
GoRight	push-button entity		IconClicked	Cursor.Right
IncNum	push-button entity		IconClicked	Cursor.QueryXPos Tens.Inc Units.Inc
DecNum	push-button entity		IconClicked	Cursor.QueryXPos Tens.Dec Units.Dec

The manual operation of any of the four push-button objects is represented by the user clicking on the icon of the object. For instance, to make the on-screen cursor move to the right, the VCR user would push the `GoRight` button. In executing the SOLVe specification, this is represented by the user clicking on the `GoRight` object icon. This invokes the `IconClicked` method of the `GoRight` object which sends a `Right` message to the `Cursor` object requesting it to move itself to the right. The `Cursor` object may respond by moving its icon one position to the right depending on its current position. If its already under the `UnitsDigit` then it will not move any further right, thus complying with requirement 4.

This is an example of what we mean by interactive-animation. The graphically animated SOLVe specification reacts to direct manipulation (via the graphical interface) from the user.

Bye way of an explanatory example, we study what might happen if the VCR user were to push the IncNum button. Consider figure 4 which shows one possible sequence of messages resulting from the IncNum button being pressed.

1. The user clicks on the `IncNum` object icon (representing the VCR user pushing the IncNum button).

2. The `Interface` object sends a `IconClicked` message to the `IncNum` object to tell it that its icon has been clicked.

3. The `IncNum` object solicits the `Cursor` object for its current position. Assume that `Cursor` responds with the value `2` which indicates that it is pointing at the `Units` digit.

4. The `IncNum` object requests the `Units` object to increment the units digit.

[1] Only the objects, methods, etc. which are directly relevant to the VCR requirements (and are not just artifacts of the SOLVe specification) have been included in the table.

5. The `Units` object solicits the `Tens` object for the value of the tens digit. Assume that the `Tens` responds with the value `1`.

6. Assume that the current value of the units digit is `9`. Then, an attempt to increment the value of the units digit ought to result in the tens digit being incremented and the units set to `0`, i.e. the clock value goes from `19` to `20`. The `Units` object sends an `Inc` message to the `Tens` object to tell it to increment the value of the tens digit.

7. The `Units` object sends an `SetIcon` message to the `Interface` object to tell it to redisplay its icon (using a bitmap which represents the value `0`).

8. The user sees the `Units` digit change to `0`.

9. The `Tens` object sends an `SetIcon` message to the `Interface` object to tell it to redisplay its icon (using a bitmap which represents the value `2`).

10. The user sees the `Tens` digit change to `2`.

The `If` statements in the `Inc` and `Dec` methods of the `Units` object ensure that a clock value of `24` cannot be incremented and that a clock value of `00` cannot be decremented (see the SOLVe listing in appendix D).

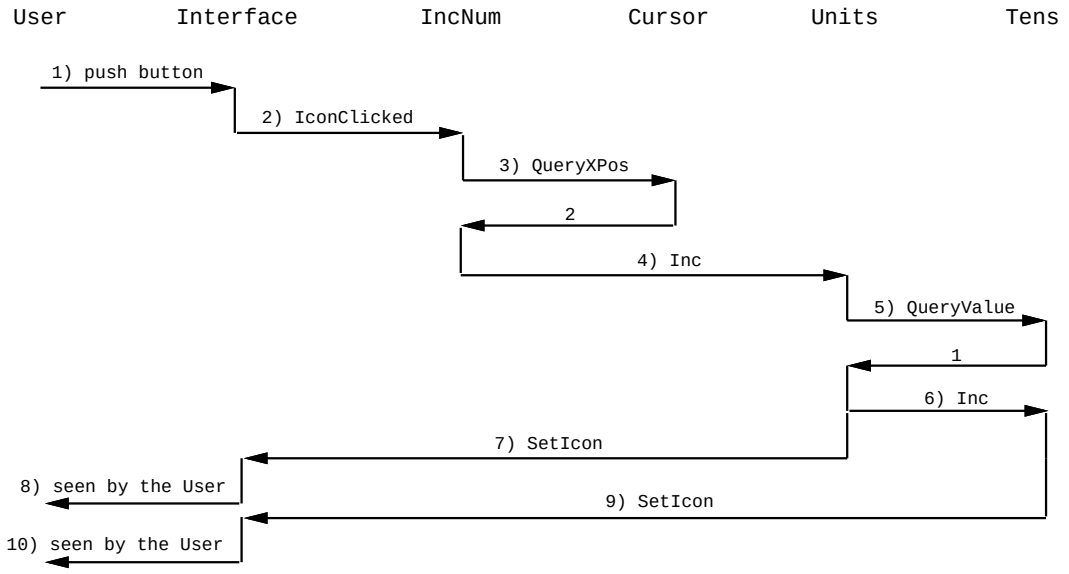


Figure 4: An example sequence of messages from the VCR system

8.3 The LOTOS specification

If a syntactically correct version of the VCR SOLVe specification is stored in the file `vcr.solve.oo` then executing the command `solve vcr` will generate the LOTOS specification `vcr.solve.lot` and then visually animate this specification. This process will generate various other `vcr.solve` files — see chapter 9.6. The LOTOS specification of our VCR example is listed in appendix E.

To specify our VCR example in object-oriented SOLVe takes approximately 250 lines of SOLVe code. The equivalent LOTOS specification, automatically generated from the SOLVe specification by the `parser`, consists of approximately 1700 lines of LOTOS. This example shows that SOLVe yields a favourable *user-effort to productivity ratio* (i.e. the amount of SOLVe code written by the user compared to the amount of equivalent, object-oriented styled, LOTOS code).

8.4 Animating the specification

In a real situation the analyst will want to check that a SOLVe specification is correct with respect to the informal requirements. To do this the analyst may animate the specification using the SOLVe toolset. Together the analyst and customer may explore and assess the animated behaviour of the specified system to establish its correctness and completeness. The SOLVe `animator` represents, and provides interaction with the specified system in a way which is intelligible and intuitive to the customer.

The command `solve vcr` will invoke the `animator` once it has parsed the SOLVe specification and generated the equivalent LOTOS specification. The following sections provide a storyboard account of an example session of using the SOLVe `animator` to interact with the VCR specification.

8.4.1 Start-up

When the `animator` is invoked an `animator` window and a `displayer` window appear.
The `animator` window provides a menu of possible next events. When the VCR starts seven internal events are possible. These seven events comprise of the `Tell` calls within the `Initialize` methods of the seven objects. These `Tell` calls initialize the icons for each of these objects.
Before these *Tell* calls occur the `animator` displays default bitmaps for the object icons at a default location — the result is the unintelligible graphic at the left of the `displayer` window. Initially this graphic is unintelligible because the icons overwrite each other since they are all written to the same default location.

8.4.2 Displaying event offers

The `animator` user may select a `view option` which displays the event offers coming in from the `hippo` tool and the event offers coming in from the `displayer` tool. Section 9.4 describes how the menu of possible next events is established by matching the `hippo` event offers with the `displayer` event offers.
Once this `view option` has been selected, the `animator` displays a total of four windows.

This window displays the event offers from `hippo`. (This window has the label `event offers from the specified system` because, in effect, `hippo` is acting the rôle of the specified system.) At present, the VCR is offering seven internal events (the `TellCall` icon initialization events) and two observable event offers (corresponding to any of the objects accepting either an `IconClicked` or an `IconMoveRequest` operation[a]). The seven internal events are listed in the top window because these are possible next events (being internal events they need not be synchronized with the event offers from the `displayer`). The two observable event offers are not listed in the top window because they cannot be matched with any event offers from the `displayer` tool.

This window displays the event offers from the `displayer tool`. (This window has the label `event offers from the specified system's environment` because, in effect, the `displayer` is acting the rôle of the specified system's user.) At the moment there are no `displayer` event offers.

[a] Although any of the objects can accept these operations, only the `GoLeft`, `GoRight`, `IncNum` and `DecNum` objects act on receiving `IconClicked` messages.

8.4.3 Choosing events from the menu

By default the `animator` tool assumes the the selection of an event from the menu of possible next events is user prompted. To select the next event the user simply clicks the mouse pointer on the desired event which will be highlighted as shown. This event will occur, the simulated system will be moved into its next state, and the `animator` window displays will be updated.

The chosen event corresponds to the `Units` object sending a `Tell` message to the `Interface` object.

The chosen event corresponds to the `Interface` object receiving a `Tell` message from the `Units` object.

The chosen event corresponds to the `Interface` object displaying a particular object icon. The structure of this event is:

```
Interface!SetIcon
            !<client object ID>
            !<x coordinate of icon>
            !<y coordinate of icon>
            !<bitmap ID of icon>
```

(If, as in this case, the text of an event is too long of the window then the window can be scrolled or enlarged.)

Once the `SetIcon` event has occurred the `displayer` window will update its display accordingly. This `SetIcon` event corresponds to an icon being displayed which represents the `Units` object — the bitmap which looks like a zero digit in the on-screen clock display. (The `Units` icon has been moved from its default position to its specified initial position. At this stage the other object icons have yet to be moved from their default positions to their specified initial positions and bitmaps.)

8.4.4 Choosing automatic animation

An `animation option` allows the `animator` tool to run in automatic mode. In this mode events are automatically selected from the menu of possible next events — user prompted selection is not then required. The `animation option` summarizes the basis on which automatic next event selection takes place.

The tool is in automatic mode and all remaining events in the possible next event menu have been executed. These remaining events cause the rest of the object icons to be displayed at their specified initial positions and with their specified initial bitmaps.

8.4.5 Direct interaction with the specification

The `animator` user may act in the rôle of the VCR user by direct graphical manipulation of the object icons (e.g. by clicking on or dragging the object icons). This screen dump shows a situation where the user has just clicked upon the `GoLeft` icon (corresponding to the VCR user pushing the button to make the on-screen cursor move to the left).

This interaction has been captured by the `displayer` tool which then offers an `IconClicked` event to the VCR system.

The VCR system (simulated by `hippo`) is itself offering to synchronize on an `IconClicked` event.

The `animator` matches the event offers from the `displayer` and from `hippo`, resulting in the shown possible next event menu. Since the `animator` tool is in automatic mode, this event will automatically be chosen. The execution of this event will initiate a 'chain reaction' of events (all chosen automatically). This chain of LOTOS events will represent the sequence of SOLVe object message invocations which represent the activation of the `GoLeft` button and the subsequent movement of the `Cursor` to the left (in accordance with the requirements for the VCR, chapter 8.1, and the example message sequence explained in figure 2).

This chain of execution will result in the `Cursor` moving to a position under the `Tens` digit of the on-screen clock.

Input from the VCR user is queued. If the user, say, quickly clicks three times in succession on the `IncNum` icon, three `IconClicked` events are registered in the possible next event menu as shown. In automatic mode these will be selected on a FIFO basis. In user prompted mode these can be selected in any order by the user.
This shows the state of the VCR system resulting from the completion of the above three `IconClicked` events. The `Units` digit has become `3`.

Chapter 9

The SOLVe toolset

The SOLVe toolset consists of 6 main programs: `editor`, `parser`, `displayer`, `animator`, a modified version of `hippo` [Mar89], and `solve`. The tools are written in C, X [You89] (using XDesigner [Tec92]), YACC and UNIX shell.

9.1 The `editor`

The `editor` tool has not been written. (SOLVe is only a prototype system and the implementation of the `editor` tool was not necessary to assess the worth of the SOLVe idea.) The `editor` tool would appear as an X window with a number of pull-down menus. The `editor` tool would allow the user to:

- invoke a standard UNIX editor (e.g. `vi`) to edit the text of a SOLVe description
- browse a bitmap library of object icons
- browse a library for previously defined objects
- cut, copy and paste objects from the library to the description under construction, and vice versa
- invoke the `parser` tool to check the syntax of a SOLVe description.

The facility to cut, copy and paste objects would preserve the relative references of the manipulated objects. For example, say we want to copy a composite object which is composed of the simple objects x and y, where object x sends a message to object y. The copy facility will make copies, x' and y', of the defining texts of x and y. In the original text x makes a message call to y, but the copy facility will have replaced this text in the copy text with a message call from x' to y'.

The `editor` tool would create files containing SOLVe descriptions (`name.solve.oo` files) and files containing the object library (`name.solve.lib` files).

9.2 The parser

The `parser` tool has been written using YACC. The parser accepts a `name.solve.oo` file containing a SOLVe specification. If this specification is syntactically correct then the parser produces a `name.solve.lot` file (containing a LOTOS specification) and a `name.solve.anim` file (containing information needed for animation). However, if the description is syntactically incorrect then the `parser` produces a `name.solve.err` file (containing a list of error messages).

9.3 The displayer

The `displayer` tool appears as a X window. The `displayer` performs two main tasks. It displays object icons (bitmaps and identification text) in response to requests from the `animator` tool, and it passes requests to click or drag object icons (indicated by mouse pointer events) from the user to the `animator` tool.

9.4 The animator

The `animator` tool appears as a set of X windows with pull-down menus. Its function is to manage the interactive animation of a SOLVe specification. When invoked with a `name.solve.lot` file and a `name.solve.anim` file, the `animator` tool spawns the `displayer` tool and the `hippo` tool as child processes. The `hippo` tool is initialized with the file `name.solve.lisa` (produced by first running the `hippo` toolset's `sclotos`, `lastb` and `lisa` programs on the `name.solve.lot` file), and the `displayer` tool is initialized with the `name.solve.anim` file. The `animator` communicates via UNIX pipes to/from the `stdin` and `stdout` of `hippo` and `displayer` (see appendix A.1).

The `hippo` tool simulates the LOTOS specification that it is given. This, in effect, defines the behaviour offered by the specified system. The `displayer` tool turns user input into event offers. This, in effect, defines the behaviour offered by the environment of the specified system. To manage an interactive animation, the `animator` synchronizes the `hippo` events offers and the `displayer` event offers. This is possible because both the `parser` and `displayer` tools generate only (a small number of) predefined LOTOS event structures. (A diagram outlining the architecture and interworking of the `animator`, `displayer` and `hippo` tools is given in appendix A.2.)

At any one instant during an animation there may be a choice of possible events. A menu option can be set so that the either the `animator` resolves a choice between possible events itself, or it prompts the SOLVe user to resolve a choice when it occurs. If the first option is set, the `animator` resolves choices between possible events on the basis that internal events have highest priority, `SetIcon` observable events next, and then other observable events are chosen in the order in which they are offered. This basis makes sense for the object-oriented architecture of the SOLVe generated LOTOS specifications.

9.5 The `hippo` simulator

The SOLVe system uses various programs from the SEDOS toolset [Mar89] to generate a `<name>.solve.lisa` file from a `<name>.solve.lot` file, and uses a modified version of the SEDOS `hippo` simulator tool to simulate a `<name>.solve.lisa` file (under the management of SOLVe's `animator` tool). Minor modifications to `hippo` have been made to make it easier to parse `hippo` output.

9.6 The `solve` script

Given a set of `name.solve.*` files the `solve` shell/make script invokes, as appropriate, the programs `parser`, `lastb`, `lisa`, `sclotos`, `animator` (and `displayer` and `hippo`). The table below summaries the programs invoked by the solve script and the resulting files, etc.

Function	Programs	Input files	Output files
syntax and static semantics check SOLVe specification	(SOLVe) parser	name.solve.oo	name.solve.lot (if ok) name.solve.anim (if ok) name.solve.err (if error)
syntax check LOTOS specification	(SEDOS) sclotos	name.solve.lot	
static semantics check and flattening LOTOS specification	(SEDOS) lastb|lisa	name.solve.lot	name.solve.lisa (if ok)
interactive animation of specification	(SOLVe) animator (SOLVe) displayer (SEDOS) hippo	 name.solve.anim name.solve.lot	produces an X based graphical animation

Chapter 10

Translating SOLVe to LOTOS

This chapter describes how a SOLVe specification is represented in LOTOS. We describe how the SOLVe model is supported at the LOTOS language level. Then we outline, by example, the algorithm used to translate a SOLVe specification into a LOTOS specification.

10.1 The SOLVe model in LOTOS

Figure 5 illustrates the architecture of a SOLVe design at the LOTOS specification level.

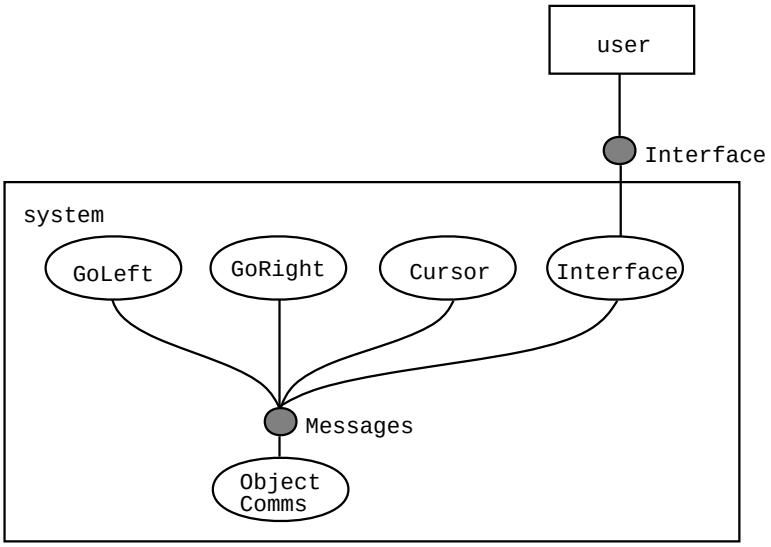


Figure 5: Architecture of a design at the LOTOS specification level

At the LOTOS specification level objects are realized as LOTOS processes. These object processes synchronize on a process called `ObjectComms` which acts as a communications medium. All inter-object (intra-system) communication occurs at the hidden LOTOS gate `Messages`. All system-user communication occurs at the observable LOTOS gate `Interface`.

For two resons we support inter-object communication using an `ObjectComms` process instead of supporting direct synchronization between communicating objects. Most importantly the `ObjectComms` process model supports non-blocking `Tell` method invocations. `Tell` calls are

effectively queued within the `ObjectComms` process until the targeted process (server object) is ready to receive them. The `ObjectComms` process is always willing to accept `Tell` calls, thus the sending process (client object) never has to wait to synchronize on a `Tell Send` event (see figure 6). The second reason for using the `ObjectComms` process model is its flexibility in routing message communications. This communications model supports an object model in which the communication connections between objects can be dynamically modified. This is a feature of most object-oriented models, although the features of dynamic object creation and dynamic modification of communication connections are not supported in the current version of the SOLVe language.

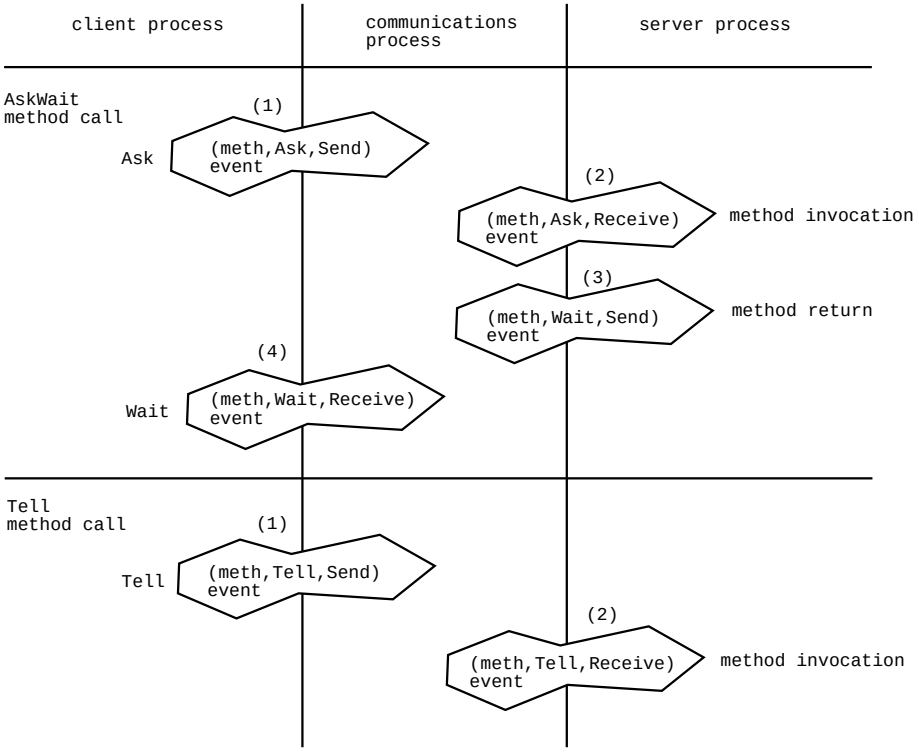


Figure 6: Realization of `AskWaitCall`s and `TellCall`s as LOTOS event sequences

Figure 6 provides two event sequence diagrams which show how `AskWait` and `Tell` method invocations are conceptualized at the LOTOS specification level. An `AskWait` method invocation consists of an `Ask` message communication followed by a complementary `Wait` message communication. A `Tell` method invocation consists solely of a `Tell` message communication. Each `Ask`, `Wait` or `Tell` message communication is delimited by two LOTOS events carrying the parameters `Send` and `Receive`. For example, an `Ask` message communication consists of an event with parameters `(meth,Ask,Send)` followed by an event with parameters `(meth,Ask,Receive)`, where the `meth` parameter is any legitimate method name.

The `ObjectComms` process is always willing to synchronize on any message event, thus it never delays the progress of object processes. The `ObjectComms` process queues `Receive` event offers, thus the responsibility for the acceptance of events representing method invocations and returns rests purely with the targeted object processes.

10.2 An outline of the translation algorithm

Here we briefly outline the translation of a SOLVe language description into a LOTOS specification. We describe, for example cases, the results produced by parts of the translation function. `_TRANS` is used to denote the translation function, and `< >` brackets mark the position of detailed text which has been suppressed for brevity. See appendix D for an example of a complete SOLVe specification and appendix E for the corresponding LOTOS specification automatically generated by the `parser` tool.

10.2.1 Translating the description shell and object declarations

Consider the translation of the following SOLVe text:

```
_TRANS(
   System VcrControl Is

    PicDecls
        digitZero, digitOne, <more PicDecls>
    EndPicDecls

    ObjectDeclarations
        Object Cursor() Is Left()() Right()() QueryXPos()(Int) EndObject
        <declaration of other objects>
    EndObjectDeclarations

    ObjectDefinitions
        <definition of Cursor object>
        <definitions of other objects>
    EndObjectDefinitions

   EndSystem
)
```

This is expanded to the following LOTOS text:

```
(* System *)
SPECIFICATION VcrControl [Interface] :NOEXIT

   LIBRARY Boolean, Integer ENDLIB

   <TYPE definitions for
      MessagePrimitiveType (ie. Send, Receive),
      MessageDescriptorType (ie. Ask, Wait, Tell),
      SystemConstantsType (ie. XMIN, YMIN, XMAX, YMAX),
      PicConstantsType (ie. the user declared icon picture identifiers)>

   (* ObjInstDecls *)
   BEHAVIOUR
      HIDE Messages IN
      (    ObjectComms[Messages]
         |[Messages]|
            (    Interface[Interface,Messages]
            ||| initCursor[Messages](0,0,0)
            ||| <instantiations of processes representing other objects>
            )
      )
   WHERE
   <TYPE definition for objNamesType>
   <TYPE definition for NamesType>
   (* EndObjInstDecls *)
```

```
(* ObjInstDefns *)
_TRANS( <definition of Cursor object> )
_TRANS( <definitions of other objects> )

(* process *)
PROCESS Interface[Interface,Messages] :NOEXIT :=
      (* method   SetIcon *)
      <accept a (SetIcon,Ask/Tell,Receive) event from an object>
      <redisplay the appropriate object icon on the display>
      <if an Ask event was accepted then
          send a confirming (Send,SetIcon,Wait) event back to the object>
      <recurse>
   []
      (* call an IconClicked method *)
      <accept an IconClicked event from the user>
      <send a (IconClicked,Tell,Send) event to the appropriate object>
      <recurse>
   []
      (* call an IconMoveRequest method *)
      <accept an IconMoveRequest event from the user>
      <send a (IconMoveRequest,Tell,Send) event to the appropriate object>
      <recurse>
ENDPROC

PROCESS ObjectComms[Messages] :NOEXIT :=
   (* carrier text for messages of method Interface.SetIcon *)
      <accept a (Interface.SetIcon,Ask/Tell,Send) event>
      (
         <offer a corresponding (Interface.SetIcon,Ask/Tell,Receive) event>
         STOP
      |||
         ObjectComms[Messages]
      )
   []
      <accept a (Interface.SetIcon,Wait,Send) event>
      (
         <offer a corresponding (Interface.SetIcon,Wait,Receive) event>
         STOP
      |||
         ObjectComms[Messages]
      )
   (* end carrier text for messages of method Interface.SetIcon *)
[]
   <carrier text for messages of method <anyObject>.IconClicked>
[]
   <carrier text for messages of method <anyObject>.IconMoveRequest>
[]
   <carrier text for messages of each of the user declared methods>
ENDPROC
(* EndObjInstDefns *)

ENDSPEC
(* EndSystem *)
```

- Notice that the top-level LOTOS behaviour expression reflects the object-oriented architecture depicted in figure 5. The `Interface`, `Cursor` and other objects inter-communicate, asynchronously, via the hidden gate `Messages` of the `ObjectComms` process. Only the `Interface` object communicates with the system user via the external gate `Interface`.

- The `Interface` object offers three functions: displaying an object icon in response to a `SetIcon` message from an object; sending an `IconClicked` message to an object in response to a user initiated event; or sending an `IconMoveRequest` message to an object

in response to a user initiated event.

- The `ObjectComms` consists of a choice of behaviours expressions. Each of these behaviour expressions carries messages for one method type. The method types are the implicitly declared methods `SetIcon`, `IconClicked` and `IconMoveRequest`, and each of the user-declared methods.

  For each of these method types, the `ObjectComms` process offers to accept any `Send` event then (concurrently) both recurse and offer to provide a complementary `Receive` event. In this way the `ObjectComms` process offers to queue an unlimited number of inter-object communication messages. This scheme supports non-blocking `Tell` method invocations (see figure 3).

10.2.2 Translating an object definition

Consider the translation of the following SOLVe text:

```
_TRANS(
   Object Cursor() Is
      <definition for the method Initialize> )
      <definition for the method IconClicked> )
      <definition for the method IconMoveRequest> )
      <definition for the method Left> )
      <definition for the method Right> )
      <definition for the method QueryXPos> )
   EndObject
)
```

This is expanded to the following LOTOS text:

```
(* Object *)
(* object Cursor *)
PROCESS initCursor[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE
   PROCESS Cursor[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
   []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
   []
      Left[Messages](xPos,yPos,iconPic)
   []
      Right[Messages](xPos,yPos,iconPic)
   []
      QueryXPos[Messages](xPos,yPos,iconPic)
   ENDPROC

   _TRANS( <definition for the method Initialize> )
   _TRANS( <definition for the method IconClicked> )
   _TRANS( <definition for the method IconMoveRequest> )
   _TRANS( <definition for the method Left> )
   _TRANS( <definition for the method Right > )
   _TRANS( <definition for the method QueryXPos> )
ENDPROC
(* EndObject *)
```

- The process `initCursor` calls the process `Initialize` in order to perform any user-defined initialization of the object.

- Once `Initialize` has completed its behaviour it calls the process `Cursor` which infinitely often offers the remaining methods of the object.
- Object methods are offered as alternate threads of behaviour because of the use of the LOTOS choice operator within the `Cursor` process.

10.2.3 Translation of a method definition

Consider the translation of the following SOLVe text:

```
_TRANS(
   Method QueryXPos() Is
      Return(xPos)
   EndMethod
)
```

This is expanded to the following LOTOS text:

```
(* Method *)
(* method QueryXPos *)
PROCESS QueryXPos[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort
               !QueryXPos[(msgDescr eq Ask) or (msgDescr eq Tell)];
   (LET dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Cursor!clientObj!Wait!QueryXPos!xPos;
            Cursor[Messages](xPos,yPos,iconPic)
         )
      []
         ([msgDescr eq Tell]->
            Cursor[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)
```

- The first event offer corresponds to an `(QueryXpos,mesgDescr,Receive)` message communication event (see figure 6).
- The `LET` statement is where local variable declarations (declared using `Variables ... EndVariables`, see chapter 7) would be placed if there where any for this method definition.
- If the method is invoked using an `Ask` message (i.e. `msgDescr eq Ask`) then the method finishes by returning a complementary `Wait` message, before re-instantiating the `Cursor` object process (the process from which this method process was instantiated). If the method is invoked using a `Tell` message (i.e. `msgDescr eq Tell`) then no return message is expected so the method simply re-instantiates the `Cursor` object process.
- Note that `Initialize` method definitions are translated in a way different to other method definitions (such as this `QueryXPos` method). `Initialize` translations contain no initial `Receive` or final `Send` events.

10.2.4 Translation of an `Assign` statement

Consider the translation of the following SOLVe text:

```
_TRANS(
   Assign(xPos,xPos Plus 1)
   <more SOLVe text>
)
```

This is expanded to the following LOTOS text:

```
(* Assignment *)
(LET xPos:Nat = xPos + succ(0) in
   _TRANS( <more SOLVe text> )
)
```

10.2.5 Translation of an `AskWait` call

Consider the translation of the following SOLVe text:

```
_TRANS(
   AskWaitCall DataBase.QueryValue()(reply)
   <more SOLVe text>
)
```

This is expanded to the following LOTOS text:

```
(* AskWaitCall *)
Messages!Send!User1!DataBase!Ask!QueryValue;
Messages!Receive!DataBase!User1!Wait!QueryValue?reply:Int;
(* EndAskWaitCall *)
_TRANS( <more SOLVe text> )
```

- The method invoker is an object called `User1` (a data type constant defined through the translation process). The `User1` object calls the `QueryValue` method of an object called `DataBase`.
- From the perspective of the `User1` object, the `AskWait` method call consists of two consecutive events which correspond to the the events labelled `(1)` and `(4)` in the `AskWait` call depicted in figure 6.
- Events representing message communications have the structure:

```
Messages!<Send/Receive>!<ID of sender object>!<ID of receiver object>
           !<Ask/Wait/Tell>!<method name>!<data parameters, if any>
```

- The method returns with an `Int` data value which is stored in the variable `reply`.

10.2.6 Translation of a `Tell` call

Consider the translation of the following SOLVe text:

```
_TRANS(
   TellCall Robot.SetLocation(2,1)
   <more SOLVe text>
)
```

This is expanded to the following LOTOS text:

```
(* TellCall *)
Messages!Send!User1!Robot!Tell!SetLocation!Succ(Succ(0))!Succ(0);
(* EndTellCall *)
_TRANS( <more SOLVe text> )
```

- The method invoker is an object called `User1`. The `User1` object calls the `SetLocation` method of an object called `Robot`, with the data parameters `2,1`. (Also see the `Tell` call depicted in figure 6.)

10.2.7 Translation of an `If` statement

Consider the translation of the following SOLVe text:

```
_TRANS(
   If ((xPos Plus 1) Gt   XMAX)
      Then
         <more SOLVe text (1)>
      Else
         <more SOLVe text (2)>
   EndIf
   <more SOLVe text (3)>
)
```

This is expanded to the following LOTOS text:

```
(* If *)
(
   (* Then *)
   ([(xPos + succ(0)) Gt XMAX]->
      _TRANS( <more SOLVe text (1)> )
   )
   []
   (* Else *)
   ([Not((xPos + succ(0)) Gt XMAX)]->
      _TRANS( <more SOLVe text (2)> )
   )
   []
   (* dummy EXIT list *)
   ([True eq False]->
      EXIT(cursorXPos,iconPic,yPos,xPos)
   )
)
(* EndIf *)
>> ACCEPT cursorXPos:Int,iconPic:Int,yPos:Int,xPos:Int IN
_TRANS( <more SOLVe text (3)> )
```

- The `Then` and `Else` branches of a SOLVe `If` statement are represented in LOTOS as guarded behaviours within a choice expression.
- The third alternative behaviour can never occur (`True eq False` is never true). This third alternative behaviour is introduced to ensure that the `ACCEPT` statement, which always follows `If` statement behaviour, is satisfactorily matched with `EXIT` behaviour (even when the `Then` and `Else` branches do not `EXIT`) — a LOTOS static semantics requirement.
- Each parameter list contains the variables and implicitly declared instance parameters which are in scope at that point.

10.2.8 Translation of a `While` statement

Consider the translation of the following SOLVe text:

```
_TRANS(
   While (line Eqi Busy) Do
      <more SOLVe text (1)>
   EndWhile
   <more SOLVe text (2)>
)
```

This is expanded to the following LOTOS text:

```
   WhileLoop0[Messages](arg2,arg1,iconPic,yPos,xPos)
   >> ACCEPT arg2:Int,arg1:Bool,iconPic:Int,yPos:Int,xPos:Int IN
   _TRANS( <more SOLVe text (2)> )

<the WhileLoop0 process is defined as a local process definition within the
 process definition of the object in which the While statement occurred>
WHERE
    <...>

    (* a process embodying a while-loop *)
    PROCESS WhileLoop0[Messages](arg2:Int,arg1:Bool,iconPic:Int,yPos:Int,xPos:Int)
                                            :EXIT (Int,Bool,Int,Int,Int):=
       (* While *)
       (
          (* Do loop *)
          ([line eq Busy]->
             _TRANS( <more SOLVe text (1)> )
             >> ACCEPT arg2:Int,arg1:Bool,iconPic:Int,yPos:Int,xPos:Int IN
             WhileLoop0[Messages](arg2,arg1,iconPic,yPos,xPos)
          )
          []
          (* exit from loop *)
          ([Not(line eq Busy)]->
             EXIT(arg2,arg1,iconPic,yPos,xPos)
          )
       )
       (* EndWhile *)
    ENDPROC
```

- In LOTOS the `While` loop is defined within a process definition (`WhileLoop0`) local to the process definition of the object.
- Each instantiation of `WhileLoop0` corresponds to one iteration of the SOLVe `While` loop.

- On each instantiation, the process `WhileLoop0` recurses if the `While` condition is not satisfied. If the `While` condition is satisfied then the process `WhileLoop0` `EXIT`s.
- The parameters `iconPic`, `yPos` and `xPos` are implicit object instance parameters. The parameters `arg2` and `arg1` are either local method variables declared earlier using the `Variable` statement, or explicitly declared object instance parameters.

Chapter 11

Further work and conclusions

The following list indicates some of the areas in which further work is needed on SOLVe.

- The SOLVe `editor` tool has not been implemented.
- Enhancements to the SOLVe object-oriented language, for example:
  - Introduce complex data types into the language — presently the language only handles simple values of the sort `Int` or the sort `Bool`.
  - Provide inheritance or some other form of *reference and description*. Presently the user must provide explicit definitions of all the declared objects and methods. With an inheritance mechanism the definitions of objects or methods could be simple references to suitable definitions of previously defined objects or methods. Also, if a definition of a declared object or method was not given, then the definition would default to a reference to the definition of some other appropriate object or method.
  - Allow objects to be dynamically created and destroyed. Section 10.1 describes how the LOTOS `ObjectComms` process model supports the dynamic routing of inter-object message communication which is, in general, needed to support dynamic object creation and destruction.
  - Presently objects are represented by bitmap icons. This idea could be greatly elaborated to have objects represented by sound, text or vector graphics, etc. ("multi-media objects").

  A point to remember is that such extensions to the SOLVe language have drawbacks. At the moment, a LOTOS specification can be generated during a single pass through the SOLVe source description. This might no longer be the case if dynamic object creation or inheritance were introduced. Another possible disadvantage of enhancing the SOLVe language would be the additional complexity of the generated LOTOS specifications. At present, the generated LOTOS specification do not contain an excess of 'object-oriented support mechanism' — the structure of the generated LOTOS specification is very readable and closely reflects the structure of the SOLVe source descriptions.
- Modify the `displayer` tool to provide immediate shadow feedback when the user clicks or drags an icon. At present, if the user drags an icon nothing immediately happens until the triggered sequence of events has been executed. This execution via `hippo` often takes

some time. An immediate shadow feedback feature would provide an immediate moving outline of the icon.

- There is a problem with the amount of memory which the `hippo` program `mallocs`. `Hippo` monotonically increases its memory requirements per event occurrence. (On my workstation `hippo` it exhausts its 24Mbyte virtual data segment after about 100 events, and comes to a halt.)

  The `animator` forks `hippo` as a subprocess and communicates with `hippo`'s `stdin` and `stdout` via UNIX pipes. As far as the author is aware, it is impossible to communicate with any of the other currently available LOTOS engines in such a straightforward way. However the future release 4.0 of SMILE [vE88] may provide a solution to this problem. `SMILE`'s memory demands are not as great as `hippo`'s, and the new version of `SMILE` will provide a Tool Control Language interface which may allow the `animator` to interact with `SMILE` in a fashion similar to present way in which the `animator` interacts with `hippo`.

- It might be useful to directly define semantics for the SOLVe language. At present the SOLVe semantics are defined indirectly by the SOLVe to LOTOS translation function. A stand alone definition of the SOLVe semantics would allow us to reason more directly about a SOLVe description, would make SOLVe LOTOS independent, and might uncover ways in which to improve the language.

- If the SOLVe approach (of coding in a programming like language and automatically generating LOTOS) proves useful then it might be better to drop the SOLVe object-oriented language in favour of a well known object-oriented language such as CLOS, but still use the SOLVe approach.

- This document has focussed on the SOLVe language and supporting software tools but has not discussed a supporting methodology. It may be useful to explore how the so-called classic object-oriented analysis and design techniques can be used with SOLVe.

In conclusion, we hope that it will prove easier to code requirements using the program-like SOLVe language than using the LOTOS language, for anyone who is unfamiliar with formal languages. A SOLVe `editor` would allow the user to quickly create requirements descriptions by copying and customizing predefined objects from construction kit libraries for the problem domain at hand. The SOLVe `parser` automatically generates LOTOS specifications from SOLVe descriptions. The SOLVe `animator` supports the visualization of SOLVe generated LOTOS specifications. The `animator` makes it easy to prototype requirements specifications using interactive animation.

The basic concepts of SOLVe are quite general. LOTOS processes (etc.) represent objects. Objects represent interactive graphical entities. These graphical entities could be customized for the visualization of the behaviour of, say, a communication service, an electronic circuit, an interface design or a space invaders game.

The SOLVe toolset should be regarded as experimental software developed to demonstrate the SOLVe approach. We believe that SOLVe is a useful prototype in the path towards the development of more mature systems for the object-oriented, visual, formal specification of interactive-systems.

References

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[DDN92] Sarah Douglas, Eckehard Doerry, and David Novick. QUICK: a tool for graphical user-interface construction by non-programmers. *The Visual Computer*, 8(2):117–133, 1992.

[Gib93a] J. Paul Gibson. *Formal Object Oriented Development of Software Systems using LOTOS*. PhD thesis, Uni. of Stirling, Scotland, 1993.

[Gib93b] J. Paul Gibson. A LOTOS-based approach to neural network specification. Technical Report TR112, Uni. of Stirling, May 1993.

[HdR91] C. Huizing and W. P. de Roever. Introduction to design choices in the semantics of statecharts. *Information Processing Letters*, 37:205–213, February 1991.

[HHR92] Jurgen Herczeg, Hubertus Hohl, and Matthias Ressel. Progress in building user interface toolkits: The world according to XIT. In *Proc. of the ACM Symp. on User Interface Software and Technology*, November 1992.

[HHR93] Jurgen Herczeg, Hubertus Hohl, and Matthias Ressel. A new approach to visual programming in user interface design. In *Proc. of HCI Int. 1993, 5th Int. Conf. on Human-Computer Interaction jointly with 9th Symp. on Human Interface (Japan), Orlando, Florida*, August 1993.

[HLN+90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMENT: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[ISO89] ISO. Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour. International Organization for Standardization, 1989. 8807.

[Mar89] A. K. Marshall. Introduction to LOTOS tools. In *[vEVD89]*. North-Holland, 1989.

[MC93] Ana M. D. Moreira and Robert G. Clark. Using rigorous object-oriented analysis. Technical Report TR111, Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland., August 1993.

[McC93] Ashley McClenaghan. ReCap-IS: a tool for capturing the requirements of interactive-systems, using LOTOS. Technical Report SPLICE/9, University of Stirling, May 1993. Rejected FORTE'93 submission.

[McC94] Ashley McClenaghan. XDILL: an X-based simulator tool for DILL. Technical Report to be released, University of Stirling, January 1994.

[Naf91] Maurice Naftalin. A visual refinement system. Technical Report TR73, Uni. of Stirling, August 1991.

[PR91] Ken Parker and Gordon Rose, editors. *FORTE'91, Fourth International Conference on: Formal Description Techniques*, Sydney, Australia, 1991. Elsevier Science Publishers B.V.

[Rud92] Steve Rudkin. Inheritance in LOTOS. In *[PR91]*, pages 409–424, 1992.

[Tec92] Imperial Software Technology. *X Designer*. Imperial Software Technology Limited, 95 London St., Reading, Berkshire, U.K., 1992.

[Thi90] Harold W. Thimbleby. *User Interface Design*. Addison-Wesley, 1990.

[Thi93a] Harold W. Thimbleby. *The Frustrations of a Push-Button World*, pages 202–219. Encyclopaedia Britannica yearbook of Science and the Future. Encyclopaedia Britannica, 1993.

[Thi93b] Harold W. Thimbleby. Hyperdoc: a system combining systems and their manuals in the same hypertext. Technical report, University of Stirling, Febrauary 1993.

[Tur89] Kenneth J. Turner. A LOTOS-based development strategy. In *[Vuo89]*, pages 157–174, November 1989.

[Tur92] Kenneth J. Turner. SPLICE I: Specification using LOTOS for an interactive customer environment – Phase I. Technical Report SPLICE/1, University of Stirling, February 1992.

[Tur93a] Kenneth J. Turner. An engineering approach to formal methods. In *13th IFIP Symp. on Protocol Specification, Testing and Verification, Liège, André Danthine, Guy Leduc and Pierre Wolper (eds)*, 1993.

[Tur93b] Kenneth J. Turner. *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*. John Wiley & Sons, Inc., first edition, 1993.

[vE88] Peter H. J. van Eijk. *Software tools for the Specification Language LOTOS*. PhD thesis, Twente University of technology, Enschede, Netherlands, 1988.

[vEVD89] Peter H. J. van Eijk, Chris A. Vissers, and Michel Diaz, editors. *The Formal Description Technique LOTOS*. North-Holland, 1989.

[vH89] Wilfried H. P. van Hulzen. Object-oriented specification style in LOTOS. Technical Report Lo/WP1/T1.1/RNL/N0002, European LOTOSPHERE Consortium, Esprit 2304, July 1989.

[Vuo89] Son T. Vuong, editor. *The IFIP TC/WG 6.1 Second International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols, FORTE '89*, Vancouver, Canada, 1989. North-Holland.

[You89] Douglas A. Young. *X Window Systems: Programming and Applications with Xt*. Prentice Hall, 1989.

Appendix A

Architecture of the animator, displayer **and** hippo **tools**

A.1 Communications

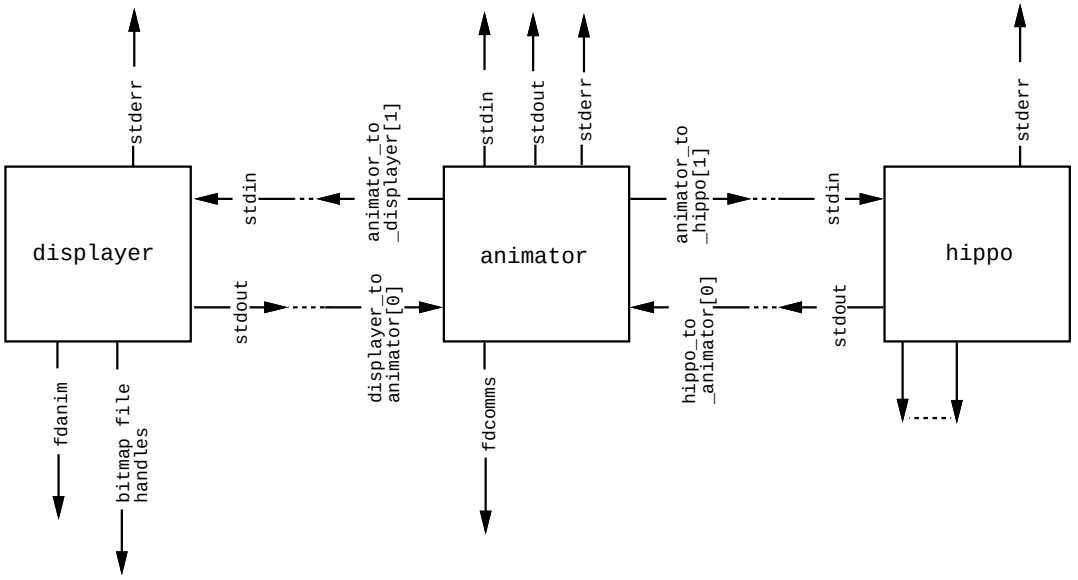


Figure 7: Communication via UNIX file descriptors

A.2 Interworking architecture

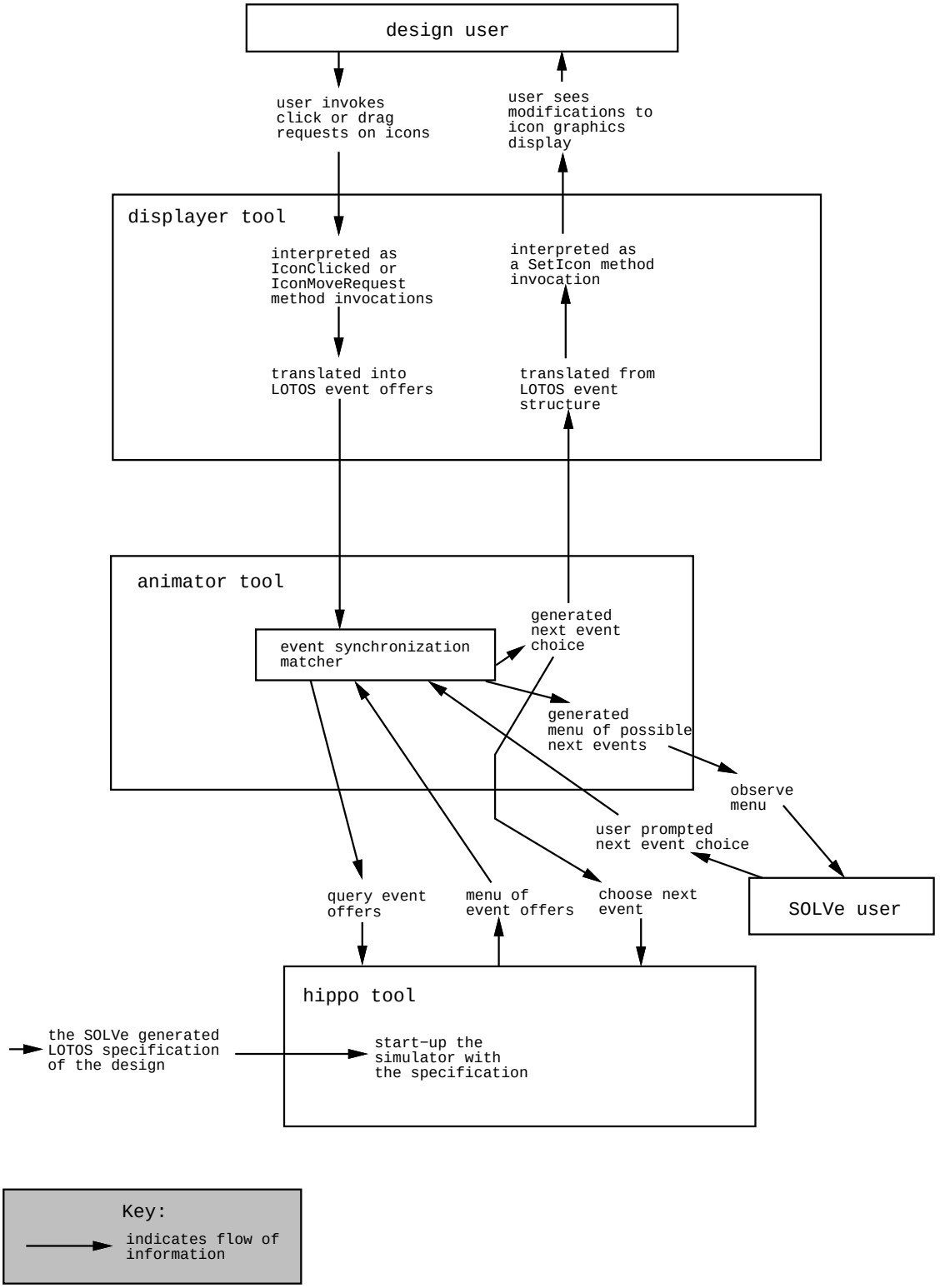


Figure 8: Interworking architecture of `animator`, `displayer` and `hippo`

Appendix B

File structure of SOLVe toolset

This appendix describes the structure of the SOLVe toolset directory tree, provides information necessary for `making` the SOLVe toolset, and points out a few implementation details.

The SOLVe programs `solve` and `animator` interact with programs from the SEDOS toolset. To support this interaction minor modifications have been made to a few of the SEDOS programs. These modifications are described in appendix B.2, but for now we concentrate on describing the SOLVe programs.

B.1 Structure of the SOLVe toolset

The following tree includes the important source code and executable SOLVe files.

```
SOLVe
    bin
        solve
        solve.animator
        solve.displayer
        solve.parser
    lib
        solve.make
        bitmaps
            cursorPtr.btm
            leftArrow.btm
            rightArrow.btm
            digitZero.btm
            ...
        specs
            vcr.solve.oo
            ls.solve.oo
    src
        makefile
        solve.typedefs.h
        animator
            makefile
            animator.xmain.xd
            animator.xmain.c
            animator.ADD_TO_MAIN.c
            animator.typedefs.h
            animator.init.c
            animator.init.h
            animator.xincludes.h
```

animator.utilities.h
animator.utilities.c
animator.callbacks.h
animator.callbacks.c
animator.eventmenu.h
animator.eventmenu.c
animator.parsehippo.h
animator.parsehippo.c
animator.parsedisplayer.h
animator.parsedisplayer.c

displayer
makefile
displayer.xmain.xd
displayer.xmain.c
displayer.ADD_TO_MAIN.c
displayer.typedefs.h
displayer.init.c
displayer.init.h
displayer.xincludes.h
displayer.utilities.h
displayer.utilities.c
displayer.callbacks.h
displayer.callbacks.c
displayer.eventhandlers.h
displayer.eventhandlers.c

parser
makefile
parser.main.c
parser.typedefs.h
parser.lotosstrs.h
parser.yacc.h
parser.yacc.y

Environment variables: The UNIX environment variable `SOLVEDIR` should contain the path of the root of the SOLVe directory tree (e.g. `/usr/local/lib/SOLVe`). The variable `LOTOSDIR` should contain the path of the root of SEDOS root directory.

Path: The `PATH` variable should be set to include the paths `($SOLVEDIR)/bin` and `($LOTOSDIR)/bin`.

Makefiles: The top-level makefile `($SOLVEDIR)/src/make` invokes three makefiles to make the three binaries in the `bin` directory.

Executables: The `bin` directory contains the shell script `solve` (described in chapter 9.6) and the binaries created my the SOLVe `make` files.

The orderly building of a SOLVe specification: The `make` script `solve.make` is invoked by the `solve` script to orderly parse, translate and animate a SOLVe specification as outlined in chapter 9.6.

Bitmaps: The `displayer` program expects to find bitmap files for the object icons in the `bitmaps` directory. To change where the `displayer` looks for its bitmap files, modify the constant `iconPicNamesPath` in the `($SOLVEDIR)/src/displayer/displayer.typedefs.h` file.

Bitmap filenames have the format `name.btm` where `name` can be used as an identifier within a `PicDecls` statement in a SOLVe specification (see chapter 7). Bitmap files can be created

using the X `bitmap` tool. Bitmaps are expected to be 48×48 (this can be changes by altering the constants `PIXMAP_WIDTH` and `PIXMAP_HEIGHT` in the `($SOLVEDIR)/src/displayer/displayer.typedefs.h` file).

Example specifications: The `lib/specs` directory contains two example SOLVe specifications: `vcr.solve.oo` a simple VCR on-screen control system (the example used in chapter 8; and `ls.solve.oo` a simple light switch system.

XDesigner code: The X utility XDesigner has been used to generate code for the `animator` and `displayer` programs. The XDesigner readable files `animator.xmain.xd` and `displayer.xmain.xd` contain data which can be used by the XDesigner program to re-generate the C files `animator.xmain.c` and `displayer.xmain.c`. For example, the user may modify the `animator` X interface by loading the `animator.xmain.xd` file into XDesigner, modifying the interface design, and then instructing XDesigner to generate the `animator.xmain.c` file (not forgetting to save the now modified version of the `animator.xmain.xd` file). Before remaking the `animator` executable, the user must edit the `animator.xmain.c` file to copy the `animator.ADD_TO_MAIN.c` file into the `animator.xmain.c` at the position described in the `animator.ADD_TO_MAIN.c` file. The `animator.ADD_TO_MAIN.c` contains necessary code not generated from the `animator.xmain.xd` file by XDesigner. The user may now re-make the `animator`.

Source code size: The parser source consists of approximately 3000 lines of C and YACC code. The animator source consists of approximately 2000 lines of handwritten C code and 500 lines of XDesigner generated C code. The displayer source consists of approximately 1000 lines of handwritten C code and 300 lines of XDesigner generated C code.

B.2 Modifications to SEDOS toolset

We have made the following minor alterations to the SEDOS toolset:

- In order for the SOLVe animator to read `hippo`'s `stdout` via a UNIX pipe we have:
  - set `hippo`'s `stdout` to be line buffered
  - replaced the use of the `forked` programs `more` and `cat` by direct writes to `hippo`'s `stdout`
  - carried out these modifications to the `($LOTOSDIR)src/hippo/simkernel` files `io.ask.c`, `io.ask.yacc.y` and `io.disp.c`.
- To allow the SEDOS programs to handle the sometimes long LOTOS text lines generated by the SOLVe `parser` we have increased the `-w -p` arguments of `($LOTOSDIR)src/hippo/simkernel/procsim.c`'s invocation of `pplotos`.
- Appropriate modifications to path names in the various `makefiles`.

Appendix C

Syntax of the SOLVe language

The BNF rules in appendixes C.1-C.4 define the syntax of the SOLVe language. ({} indicates zero or more occurrences. [] indicates zero or one occurrence. "" indicates a terminal symbol. <> indicates a comment.)

Appendix C.5 lists the static semantic errors checked for by the SOLVe `parser`.

C.1 SOLVe expressions

```
system = SYS_SYMBOL IDENT  IS_SYMBOL
        pic_decls obj_inst_decls obj_inst_defns
        SYS_END_SYMBOL .

pic_decls = PIC_DECLS_SYMBOL pic_decl_list PIC_DECLS_END_SYMBOL .

pic_decl_list = pic_decl {pic_decl} .

pic_decl = IDENT .

obj_inst_decls = OBJ_INST_DECLS_SYMBOL
                 obj_inst_decl_list
                 OBJ_INST_DECLS_END_SYMBOL .

obj_inst_decl_list = [obj_inst_decl_list2] .

obj_inst_decl_list2 = obj_inst_decl {obj_inst_decl} .

obj_inst_decl = OBJECT_SYMBOL
                IDENT "(" sort_param_list ")" IS_SYMBOL
                meth_inst_decl_list
                OBJECT_END_SYMBOL .

meth_inst_decl_list = {meth_inst_decl} .

meth_inst_decl = IDENT "(" sort_param_list ")" "(" sort_param_list ")" .

obj_inst_defns = OBJ_INST_DEFNS_SYMBOL
                 obj_inst_defn_list
```

```
                OBJ_INST_DEFNS_END_SYMBOL .

obj_inst_defn_list = {obj_inst_defn} .

obj_inst_defn = OBJECT_SYMBOL
                IDENT "(" ident_decls ")" IS_SYMBOL
                methods
                OBJECT_END_SYMBOL .

methods = method_list.

method_list = {method} .

method = METHOD_SYMBOL
         IDENT "(" ident_decls ")" IS_SYMBOL
         vars_decls
         a2meth_body
         METHOD_END_SYMBOL .

vars_decls = {VARS_SYMBOL ident_decls VARS_END_SYMBOL} .

a2meth_body = method_body

method_body = method_finish
            | askwait_call method_body
            | tell_call method_body
            | assign2
            | while_statement method_body
            | if_statement method_body .

enclosed_meth_body = <NULL>
                   | method_finish
                   | askwait_call enclosed_meth_body
                   | tell_call enclosed_meth_body
                   | assign3
                   | while_statement enclosed_meth_body
                   | if_statement enclosed_meth_body .

assign2 = ASSIGN_SYMBOL "(" IDENT "," value_expr ")" method_body .

assign3 = ASSIGN_SYMBOL "(" IDENT "," value_expr ")" enclosed_meth_body .

while_statement = WHILE_SYMBOL "("  condition ")" DO_SYMBOL
                  enclosed_meth_body
                  WHILE_END_SYMBOL .

if_statement = IF_SYMBOL "("  condition ")" THEN_SYMBOL
               enclosed_meth_body
               ELSE_SYMBOL
               enclosed_meth_body
               IF_END_SYMBOL .

method_finish = METHOD_FINISH_SYMBOL value_expr_list .

askwait_call = ASKWAITCALL_SYMBOL IDENT "." IDENT
               value_expr_list "(" var_param_list ")" .
```

```
tell_call = TELLCALL_SYMBOL IDENT "." IDENT
            value_expr_list .

sort_param_list = [sort_param_list2] .

sort_param_list2 = a_sort_param {a_sort_param} .

a_sort_param = IDENT .

var_param_list = [var_param_list2] .

var_param_list2 = a_var_param {"," a_var_param} .

a_var_param = IDENT .

ident_decls = [ident_decl_list] .

ident_decl_list = ident_decl {"," ident_decl} .

ident_decl = SORT ":" IDENT .

condition = value_expr .

value_expr_list = "(" [value_expr_list2] ")" .

value_expr_list2 = value_expr {"," value_expr} .

value_expr = term
             | unary_op term
             | value_expr binary_op term .

unary_op = BOOL_UNARY_OP
         | INT_UNARY_OP .

binary_op = BOOL_BINARY_OP
          | INT_BINARY_OP .

term = IDENT
     | INTEG
     | "(" value_expr ")"
     | SYS_LABEL_CONSTANT .
```

C.2 Strings

```
IDENT = <string of A-Z,a-z letters> .

INTEG = <string of 0-9 digits> .
```

C.3 Reserved words

```
SYS_SYMBOL = "System" .

IS_SYMBOL = "Is" .

SYS_END_SYMBOL = "EndSystem" .

OBJ_INST_DECLS_SYMBOL = "ObjectDeclarations" .

OBJ_INST_DECLS_END_SYMBOL = "EndObjectDeclarations" .

OBJ_INST_DEFNS_SYMBOL = "ObjectDefinitions" .

OBJ_INST_DEFNS_END_SYMBOL = "EndObjectDefinitions" .

OBJ_INST_DEFN_SYMBOL = "Object" .

OBJ_INST_DEFN_END_SYMBOL = "EndObject" .

METHOD_SYMBOL = "Method" .

METHOD_END_SYMBOL = "EndMethod" .

IF_SYMBOL = "If" .

IF_END_SYMBOL = "EndIf" .

THEN_SYMBOL = "Then" .

ELSE_SYMBOL = "Else" .

ASKWAITCALL_SYMBOL = "AskWaitCall" .

TELLCALL_SYMBOL = "TellCall" .

METHOD_FINISH_SYMBOL = "Return" .

VARS_SYMBOL = "Variables" .

VARS_END_SYMBOL = "EndVariables" .

ASSIGN_SYMBOL = "Assign" .

WHILE_SYMBOL = "While" .

WHILE_END_SYMBOL = "EndWhile" .

DO_SYMBOL = "Do" .

PIC_DECLS_SYMBOL = "PicDecls" .

PIC_DECLS_END_SYMBOL = "EndPicDecls" .
```

C.4 Other reserved identifiers

```
SORT = "Bool" | "Int" .

BOOL_UNARY_OP = "Not" .

BOOL_BINARY_OP = "Eqb" | "Neb" | "And" | "Or" .

BOOL_VALUE = "True" | "False" .

INT_UNARY_OP = "Succ" | "Pred" .

INT_BINARY_OP = "Plus" | "Minus" | "Eqi" | "Nei" | "Le" | "Lt" | "Gt" | "Ge" .

SYS_LABEL_CONSTANT = "XMIN" | "XMAX" | "YMIN" | "YMAX" .
```

C.5 Static semantic errors

```
-- redeclaration of identifier
-- method definitions missing for this object
-- ObjIdent identifier has not been declared
-- MethIdent identifier not known for this object
-- Sort identifier not known
-- UnaryOp identifier not known
-- BinaryOp identifier not known
-- identifier is not a Var or a SortConst or a SysConst
-- If condition is not of Sort Bool
-- Sort of operand not compatible with unary operator
-- Sort of first operand not compatible with binary operator
-- Sort of second operand not compatible with binary operator
-- attempted redefinition of object
-- actual parameter Sort mismatch
-- too few actual parameters
-- redefinition of method within this object
-- too few formal parameters
-- formal parameter Sort mismatch
-- called object does not serve the specified method
-- all declared objects have not been defined
-- too many parameters
-- self referencing call
-- nested Ifs or Whiles too deep
-- too few variable parameters
-- variable parameter Sort mismatch
-- variable, in variable paramter list, undeclared
-- attempted assignment to non-variable
-- Sort of assigned value not compatible with variable Sort
-- identifier in parameter declaration list is not a Sort
-- the first three parameters in a declaration of object instance
      variables must be of Sort Int
-- no in parameters are expected for the IconClicked method
-- only two in parameters expected for the IconMoveRequest method
-- the two in parameters are expected to be of Sort Int for the
      IconMoveRequest method
-- no out parameters are expected for the IconClicked method
-- too many parameters (maximum is %d)
-- no out parameters are expected for the IconMoveRequest method
-- must have two in parameters for an IconMoveRequest method
```

```
-- method already declared for this object (Note Initialize, IconClicked
      and IconMoveRequest are implicitly declared)
```

Appendix D

SOLVe specification of a VCR on-screen control system

The following listing is a complete SOLVe specification of a VCR on-screen control system.

```
--------------------- VCR on-screen control system ----------------------
--
-- File:    vcr.solve.oo
-- Author: Ashley McClenaghan
-- Date:    19/11/93
--
-- This is a specification of an extremely simplified an on-screen control
-- system for a VCR. The specified system allows the VCR user to set
-- an on-screen 24 hour clock using an on-screen cursor manipulated
-- via cursor control buttons.
--
System VcrControl Is

 ---------------------- picIcon declarations ---------------------------
 PicDecls
    digitZero, digitOne,       -- declare picIcons to represent digits
    digitTwo, digitThree,      -- (exploit the fact that the integer
    digitFour, digitFive,      -- values which are assigned to this
    digitSix, digitSeven,      -- enumeration of picIcons will correspond to
    digitEight, digitNine,    -- the number values represented by the picIcons)
    cursorPtr, leftArrow,      -- declare picIcons to represent the cursor and
    rightArrow, plusSign,      -- cursor controls
    minusSign
 EndPicDecls


 ---------------------- object declarations ----------------------------
 ObjectDeclarations
    Object Tens() Is Inc()() Dec()() QueryValue()(Int) EndObject
    Object Units() Is Inc()() Dec()() EndObject
    Object Cursor() Is Left()() Right()() QueryXPos()(Int) EndObject
    Object GoLeft() Is EndObject
    Object GoRight() Is EndObject
    Object IncNum() Is EndObject
    Object DecNum() Is EndObject
 EndObjectDeclarations


 ObjectDefinitions

  ------------------ definition of object Tens ----------------------
```

```
Object Tens() Is

   Method Initialize() Is
      Assign(xPos,1)
      Assign(yPos,1)
      Assign(iconPic,digitZero)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is Return() EndMethod

   Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

   Method Inc() Is
      If (iconPic Eqi 2)                    -- maximum value for tens for hours?
         Then
            -- do nothing
         Else
            Assign(iconPic, iconPic Plus 1)                              -- inc value
            TellCall Interface.SetIcon(xPos,yPos,iconPic)         -- redisplay
      EndIf
      Return()
   EndMethod

   Method Dec() Is
      If (iconPic Eqi 0)                    -- minimum value for tens for hours?
         Then
            -- do nothing
         Else
            Assign(iconPic, iconPic Minus 1)                             -- dec value
            TellCall Interface.SetIcon(xPos,yPos,iconPic)         -- redisplay
      EndIf
      Return()
   EndMethod

   Method QueryValue() Is
      Return(iconPic)
   EndMethod

EndObject


------------------- definition of object Units -----------------------
Object Units() Is

   Method Initialize() Is
      Assign(xPos,2)
      Assign(yPos,1)
      Assign(iconPic,digitZero)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is Return() EndMethod

   Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

   Method Inc() Is
      Variables Int:hourTensValue EndVariables
      AskWaitCall Tens.QueryValue()(hourTensValue)
      If ((hourTensValue Eqi 2) And (iconPic Eqi 4))-- maximum value for clock?
         Then
            -- do nothing
         Else
            If (iconPic Eqi 9)                                    -- need to inc tens?
```

```
            Then
               TellCall Tens.Inc()                          -- inc tens
               Assign(iconPic, digitZero)                -- set units to zero
            Else
               Assign(iconPic, iconPic Plus 1)          -- inc units
         EndIf
         TellCall Interface.SetIcon(xPos,yPos,iconPic)   -- redisplay
      EndIf
      Return()
   EndMethod

   Method Dec() Is
      Variables Int:hourTensValue EndVariables
      AskWaitCall Tens.QueryValue()(hourTensValue)
      If ((hourTensValue Eqi 0) And (iconPic Eqi 0))-- minimum value for clock?
         Then
            -- do nothing
         Else
            If (iconPic Eqi 0)                                  -- need to dec tens?
               Then
                  TellCall Tens.Dec()                          -- dec tens
                  Assign(iconPic, digitNine)                -- set units to nine
               Else
                  Assign(iconPic, iconPic Minus 1)         -- dec units
            EndIf
            TellCall Interface.SetIcon(xPos,yPos,iconPic)   -- redisplay
      EndIf
      Return()
   EndMethod

EndObject


------------------- definition of object Cursor -----------------------
Object Cursor() Is

   Method Initialize() Is
      Assign(xPos,2)
      Assign(yPos,2)
      Assign(iconPic,cursorPtr)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is Return() EndMethod

   Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

   Method Left() Is
      If (xPos Eqi 1)                          -- already at leftmost cursor position?
         Then
            -- do nothing
         Else
            Assign(xPos, xPos Minus 1)                                  -- dec cursor pos
            TellCall Interface.SetIcon(xPos,yPos,iconPic) -- redisplay
      EndIf
      Return()
   EndMethod

   Method Right() Is
      If (xPos Eqi 2)                          -- already at rightmost cursor position?
         Then
            -- do nothing
         Else
            Assign(xPos, xPos Plus 1)                                   -- inc cursor pos
            TellCall Interface.SetIcon(xPos,yPos,iconPic) -- redisplay
```

```
      EndIf
      Return()
   EndMethod

   Method QueryXPos() Is
      Return(xPos)
   EndMethod

EndObject


---------------- definition of object GoLeft ------------------
Object GoLeft() Is

   Method Initialize() Is
      Assign(xPos,6)
      Assign(yPos,5)
      Assign(iconPic,leftArrow)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is
      TellCall Cursor.Left()
      Return()
   EndMethod

   Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

EndObject


---------------- definition of object GoRight ------------------
Object GoRight() Is

   Method Initialize() Is
      Assign(xPos,7)
      Assign(yPos,5)
      Assign(iconPic,rightArrow)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is
      TellCall Cursor.Right()
      Return()
   EndMethod

   Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

EndObject


------------------ definition of object IncNum --------------------
Object IncNum() Is

   Method Initialize() Is
      Assign(xPos,8)
      Assign(yPos,4)
      Assign(iconPic,plusSign)
      TellCall Interface.SetIcon(xPos,yPos,iconPic)
      Return()
   EndMethod

   Method IconClicked() Is
      Variables Int:cursorXPos EndVariables
```

```
        AskWaitCall Cursor.QueryXPos()(cursorXPos) -- find out where the cursor is
        If (cursorXPos Eqi 4)
             Then -- cursor under tens of hours digit
                TellCall Tens.Inc()   -- request tens of hours to inc
             Else -- cursor under units of hours digit
                TellCall Units.Inc() -- request units of hours to inc
        EndIf
        Return()
     EndMethod

     Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

  EndObject


  ------------------- definition of object DecNum --------------------
  Object DecNum() Is

     Method Initialize() Is
        Assign(xPos,8)
        Assign(yPos,5)
        Assign(iconPic,minusSign)
        TellCall Interface.SetIcon(xPos,yPos,iconPic)
        Return()
     EndMethod

     Method IconClicked() Is
        Variables Int:cursorXPos EndVariables
        AskWaitCall Cursor.QueryXPos()(cursorXPos) -- find out where the cursor is
        If (cursorXPos Eqi 4)
             Then -- cursor under tens of hours digit
                TellCall Tens.Dec()   -- request tens of hours to dec
             Else -- cursor under units of hours digit
                TellCall Units.Dec() -- request units of hours to dec
        EndIf
        Return()
     EndMethod

     Method IconMoveRequest(Int:a,Int:b) Is Return() EndMethod

  EndObject

 EndObjectDefinitions

EndSystem
```

Appendix E

LOTOS specification of a VCR on-screen control system

The following listing is a complete LOTOS specification of a VCR on-screen control system. It has been automatically generated by the `parser` tool from the SOLVe specification listed in appendix D.

```
(* System *)

SPECIFICATION VcrControl [Interface] :NOEXIT


 LIBRARY
    Boolean, Integer
 ENDLIB


 TYPE MessagePrimitiveType IS Integer
    SORTS
       MessagePrimitiveSort
    OPNS
       Send:->MessagePrimitiveSort
       Receive:->MessagePrimitiveSort
       _ eq _,_ ne _:MessagePrimitiveSort,MessagePrimitiveSort->Bool
       Ord:MessagePrimitiveSort->Int
    EQNS
       FORALL x,y:MessagePrimitiveSort
       OFSORT Int
          Ord(Send) = 0;
          Ord(Receive) = Succ(0);
       OFSORT Bool
          x eq y = Ord(x) eq Ord(y);
          x ne y = Ord(x) ne Ord(y);
 ENDTYPE


 TYPE MessageDescriptorType IS Integer
    SORTS
       MessageDescriptorSort
    OPNS
       Ask:->MessageDescriptorSort
       Wait:->MessageDescriptorSort
       Tell:->MessageDescriptorSort
       _ eq _,_ ne _:MessageDescriptorSort,MessageDescriptorSort->Bool
       Ord:MessageDescriptorSort->Int
```

```
   EQNS
      FORALL x,y:MessageDescriptorSort
      OFSORT Int
         Ord(Ask) = 0;
         Ord(Wait) = Succ(0);
         Ord(Tell) = Succ(Succ(0));
      OFSORT Bool
         x eq y = Ord(x) eq Ord(y);
         x ne y = Ord(x) ne Ord(y);
ENDTYPE


 TYPE SystemConstantsType IS Integer
    OPNS
       XMIN:->Int
       YMIN:->Int
       XMAX:->Int
       YMAX:->Int
    EQNS
       OFSORT Int
       XMIN = Succ(0);
       YMIN = Succ(0);
       XMAX = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))));
       YMAX = Succ(Succ(Succ(Succ(Succ(0)))));
 ENDTYPE


(* PicDecls *)
 TYPE PicConstantsType IS Integer
    OPNS
       digitZero:->Int
       digitOne:->Int
       digitTwo:->Int
       digitThree:->Int
       digitFour:->Int
       digitFive:->Int
       digitSix:->Int
       digitSeven:->Int
       digitEight:->Int
       digitNine:->Int
       cursorPtr:->Int
       leftArrow:->Int
       rightArrow:->Int
       plusSign:->Int
       minusSign:->Int
    EQNS
       OFSORT Int
       digitZero = 0;
       digitOne = Succ(0);
       digitTwo = Succ(Succ(0));
       digitThree = Succ(Succ(Succ(0)));
       digitFour = Succ(Succ(Succ(Succ(0))));
       digitFive = Succ(Succ(Succ(Succ(Succ(0)))));
       digitSix = Succ(Succ(Succ(Succ(Succ(Succ(0))))));
       digitSeven = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))));
       digitEight = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))));
       digitNine = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))));
       cursorPtr = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))));
       leftArrow = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))))));
       rightArrow = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))))));
       plusSign = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))))))));
       minusSign = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))))))));
 ENDTYPE
(* EndPicDecls *)
```

```
(* ObjInstDecls *)

BEHAVIOUR
   HIDE Messages IN
   (
         ObjectComms[Messages]
      |[Messages]|
         (
               Interface[Interface,Messages]
            |||
               initTens[Messages](0,0,0)
            |||
               initUnits[Messages](0,0,0)
            |||
               initCursor[Messages](0,0,0)
            |||
               initGoLeft[Messages](0,0,0)
            |||
               initGoRight[Messages](0,0,0)
            |||
               initIncNum[Messages](0,0,0)
            |||
               initDecNum[Messages](0,0,0)
         )
   )
WHERE

TYPE ObjNamesType IS Integer
   SORTS
      ObjNamesSort
   OPNS
      Interface:->ObjNamesSort
      Tens:->ObjNamesSort
      Units:->ObjNamesSort
      Cursor:->ObjNamesSort
      GoLeft:->ObjNamesSort
      GoRight:->ObjNamesSort
      IncNum:->ObjNamesSort
      DecNum:->ObjNamesSort
      _ eq _,_ ne _:ObjNamesSort,ObjNamesSort->Bool
      Ord:ObjNamesSort->Int
   EQNS
      FORALL x,y:ObjNamesSort
      OFSORT Int
         Ord(Interface) = Succ(0);
         Ord(Tens) = Succ(Succ(0));
         Ord(Units) = Succ(Succ(Succ(0)));
         Ord(Cursor) = Succ(Succ(Succ(Succ(0))));
         Ord(GoLeft) = Succ(Succ(Succ(Succ(Succ(0)))));
         Ord(GoRight) = Succ(Succ(Succ(Succ(Succ(Succ(0))))));
         Ord(IncNum) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))));
         Ord(DecNum) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))));
      OFSORT Bool
         x eq y = Ord(x) eq Ord(y);
         x ne y = Ord(x) ne Ord(y);
ENDTYPE

TYPE MethNamesType IS Integer
   SORTS
      MethNamesSort
   OPNS
      SetIcon:->MethNamesSort
      IconClicked:->MethNamesSort
      IconMoveRequest:->MethNamesSort
      Initialize:->MethNamesSort
      Inc:->MethNamesSort
```

```
      Dec:->MethNamesSort
      QueryValue:->MethNamesSort
      Left:->MethNamesSort
      Right:->MethNamesSort
      QueryXPos:->MethNamesSort
      _ eq _,_ ne _:MethNamesSort,MethNamesSort->Bool
      Ord:MethNamesSort->Int
   EQNS
      FORALL x,y:MethNamesSort
      OFSORT Int
         Ord(SetIcon) = Succ(0);
         Ord(IconClicked) = Succ(Succ(0));
         Ord(IconMoveRequest) = Succ(Succ(Succ(0)));
         Ord(Initialize) = Succ(Succ(Succ(Succ(0))));
         Ord(Inc) = Succ(Succ(Succ(Succ(Succ(0)))));
         Ord(Dec) = Succ(Succ(Succ(Succ(Succ(Succ(0))))));
         Ord(QueryValue) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))));
         Ord(Left) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))));
         Ord(Right) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))));
         Ord(QueryXPos) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))))));
      OFSORT Bool
         x eq y = Ord(x) eq Ord(y);
         x ne y = Ord(x) ne Ord(y);
ENDTYPE

(* EndObjInstDecls *)


(* ObjInstDefns *)


(* Object *)
(* object Tens *)
PROCESS initTens[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS Tens[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
      []
      Inc[Messages](xPos,yPos,iconPic)
      []
      Dec[Messages](xPos,yPos,iconPic)
      []
      QueryValue[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
         (* Assignment *)
         (LET xPos:Int = Succ(0) IN
            (* Assignment *)
            (LET yPos:Int = Succ(0) IN
               (* Assignment *)
               (LET iconPic:Int = digitZero IN
                  (* TellCall *)
                  Messages!Send!Tens!Interface!Tell!SetIcon!xPos!yPos!iconPic;
                  (* EndTellCall *)
                  (* MethodFinish *)
                  Tens[Messages](xPos,yPos,iconPic)
```

```
            (* EndMethodFinish *)
         )
      )
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconClicked *)
PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Tens?msgDescr:MessageDescriptorSort!IconClicked
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Tens!clientObj!Wait!IconClicked;
            Tens[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Tens[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconMoveRequest *)
PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Tens?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Tens!clientObj!Wait!IconMoveRequest;
            Tens[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Tens[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method Inc *)
PROCESS Inc[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Tens?msgDescr:MessageDescriptorSort!Inc
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* If *)
      (
         (* Then *)
         ([iconPic eq Succ(Succ(0))]->
            EXIT(iconPic,yPos,xPos)
```

```
         )
         []
         (* Else *)
         ([Not(iconPic eq Succ(Succ(0)))]->
            (* Assignment *)
            (LET iconPic:Int = iconPic + Succ(0) IN
               (* TellCall *)
               Messages!Send!Tens!Interface!Tell!SetIcon!xPos!yPos!iconPic;
               (* EndTellCall *)
               EXIT(iconPic,yPos,xPos)
            )
         )
         []
         (* dummy EXIT list *)
         ([True eq False]->
            EXIT(iconPic,yPos,xPos)
         )
      )
      (* EndIf *)
      >> ACCEPT iconPic:Int,yPos:Int,xPos:Int IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Tens!clientObj!Wait!Inc;
            Tens[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Tens[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method Dec *)
PROCESS Dec[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Tens?msgDescr:MessageDescriptorSort!Dec
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* If *)
      (
         (* Then *)
         ([iconPic eq 0]->
            EXIT(iconPic,yPos,xPos)
         )
         []
         (* Else *)
         ([Not(iconPic eq 0)]->
            (* Assignment *)
            (LET iconPic:Int = iconPic - Succ(0) IN
               (* TellCall *)
               Messages!Send!Tens!Interface!Tell!SetIcon!xPos!yPos!iconPic;
               (* EndTellCall *)
               EXIT(iconPic,yPos,xPos)
            )
         )
         []
         (* dummy EXIT list *)
         ([True eq False]->
            EXIT(iconPic,yPos,xPos)
         )
      )
```

```
         (* EndIf *)
         >> ACCEPT iconPic:Int,yPos:Int,xPos:Int IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!Tens!clientObj!Wait!Dec;
               Tens[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               Tens[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method QueryValue *)
   PROCESS QueryValue[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!Tens?msgDescr:MessageDescriptorSort!QueryValue
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0 IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!Tens!clientObj!Wait!QueryValue!iconPic;
               Tens[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               Tens[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)


(* Object *)
(* object Units *)
PROCESS initUnits[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS Units[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
      []
      Inc[Messages](xPos,yPos,iconPic)
      []
      Dec[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
```

```
      (* Assignment *)
      (LET xPos:Int = Succ(Succ(0)) IN
         (* Assignment *)
         (LET yPos:Int = Succ(0) IN
            (* Assignment *)
            (LET iconPic:Int = digitZero IN
               (* TellCall *)
               Messages!Send!Units!Interface!Tell!SetIcon!xPos!yPos!iconPic;
               (* EndTellCall *)
               (* MethodFinish *)
               Units[Messages](xPos,yPos,iconPic)
               (* EndMethodFinish *)
            )
         )
      )
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconClicked *)
PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Units?msgDescr:MessageDescriptorSort!IconClicked
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Units!clientObj!Wait!IconClicked;
            Units[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Units[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconMoveRequest *)
PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Units?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Units!clientObj!Wait!IconMoveRequest;
            Units[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Units[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)
```

```
(* Method *)
(* method Inc *)
PROCESS Inc[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Units?msgDescr:MessageDescriptorSort!Inc
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0(* Vars *),hourTensValue:Int=0(* EndVars *) IN
      (* AskWaitCall *)
      Messages!Send!Units!Tens!Ask!QueryValue;
      Messages!Receive!Tens!Units!Wait!QueryValue?hourTensValue:Int;
      (* EndAskWaitCall *)
      (* If *)
      (
         (* Then *)
         ([(hourTensValue eq Succ(Succ(0))) And (iconPic eq Succ(Succ(Succ(Succ(0)))))]->
            EXIT(hourTensValue,iconPic,yPos,xPos)
         )
         []
         (* Else *)
         ([Not((hourTensValue eq Succ(Succ(0))) And (iconPic eq Succ(Succ(Succ(Succ(0))))))]->
            (* If *)
            (
               (* Then *)
               ([iconPic eq Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))))]->
                  (* TellCall *)
                  Messages!Send!Units!Tens!Tell!Inc;
                  (* EndTellCall *)
                  (* Assignment *)
                  (LET iconPic:Int = digitZero IN
                     EXIT(hourTensValue,iconPic,yPos,xPos)
                  )
               )
               []
               (* Else *)
               ([Not(iconPic eq Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))))]->
                  (* Assignment *)
                  (LET iconPic:Int = iconPic + Succ(0) IN
                     EXIT(hourTensValue,iconPic,yPos,xPos)
                  )
               )
               []
               (* dummy EXIT list *)
               ([True eq False]->
                  EXIT(hourTensValue,iconPic,yPos,xPos)
               )
            )
            (* EndIf *)
            >> ACCEPT hourTensValue:Int,iconPic:Int,yPos:Int,xPos:Int IN
            (* TellCall *)
            Messages!Send!Units!Interface!Tell!SetIcon!xPos!yPos!iconPic;
            (* EndTellCall *)
            EXIT(hourTensValue,iconPic,yPos,xPos)
         )
         []
         (* dummy EXIT list *)
         ([True eq False]->
            EXIT(hourTensValue,iconPic,yPos,xPos)
         )
      )
      (* EndIf *)
      >> ACCEPT hourTensValue:Int,iconPic:Int,yPos:Int,xPos:Int IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Units!clientObj!Wait!Inc;
            Units[Messages](xPos,yPos,iconPic)
         )
```

```
         []
         ([msgDescr eq Tell]->
            Units[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method Dec *)
PROCESS Dec[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Units?msgDescr:MessageDescriptorSort!Dec
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0(* Vars *),hourTensValue:Int=0(* EndVars *) IN
      (* AskWaitCall *)
      Messages!Send!Units!Tens!Ask!QueryValue;
      Messages!Receive!Tens!Units!Wait!QueryValue?hourTensValue:Int;
      (* EndAskWaitCall *)
      (* If *)
      (
         (* Then *)
         ([(hourTensValue eq 0) And (iconPic eq 0)]->
            EXIT(hourTensValue,iconPic,yPos,xPos)
         )
         []
         (* Else *)
         ([Not((hourTensValue eq 0) And (iconPic eq 0))]->
            (* If *)
            (
               (* Then *)
               ([iconPic eq 0]->
                  (* TellCall *)
                  Messages!Send!Units!Tens!Tell!Dec;
                  (* EndTellCall *)
                  (* Assignment *)
                  (LET iconPic:Int = digitNine IN
                     EXIT(hourTensValue,iconPic,yPos,xPos)
                  )
               )
               []
               (* Else *)
               ([Not(iconPic eq 0)]->
                  (* Assignment *)
                  (LET iconPic:Int = iconPic - Succ(0) IN
                     EXIT(hourTensValue,iconPic,yPos,xPos)
                  )
               )
               []
               (* dummy EXIT list *)
               ([True eq False]->
                  EXIT(hourTensValue,iconPic,yPos,xPos)
               )
            )
            (* EndIf *)
            >> ACCEPT hourTensValue:Int,iconPic:Int,yPos:Int,xPos:Int IN
            (* TellCall *)
            Messages!Send!Units!Interface!Tell!SetIcon!xPos!yPos!iconPic;
            (* EndTellCall *)
            EXIT(hourTensValue,iconPic,yPos,xPos)
         )
         []
         (* dummy EXIT list *)
         ([True eq False]->
```

```
               EXIT(hourTensValue,iconPic,yPos,xPos)
            )
         )
         (* EndIf *)
         >> ACCEPT hourTensValue:Int,iconPic:Int,yPos:Int,xPos:Int IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!Units!clientObj!Wait!Dec;
               Units[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               Units[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)


(* Object *)
(* object Cursor *)
PROCESS initCursor[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS Cursor[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
      []
      Left[Messages](xPos,yPos,iconPic)
      []
      Right[Messages](xPos,yPos,iconPic)
      []
      QueryXPos[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
         (* Assignment *)
         (LET xPos:Int = Succ(Succ(0)) IN
            (* Assignment *)
            (LET yPos:Int = Succ(Succ(0)) IN
               (* Assignment *)
               (LET iconPic:Int = cursorPtr IN
                  (* TellCall *)
                  Messages!Send!Cursor!Interface!Tell!SetIcon!xPos!yPos!iconPic;
                  (* EndTellCall *)
                  (* MethodFinish *)
                  Cursor[Messages](xPos,yPos,iconPic)
                  (* EndMethodFinish *)
               )
            )
         )
      )
   ENDPROC
   (* EndMethod *)
```

```
(* Method *)
(* method IconClicked *)
PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort!IconClicked
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Cursor!clientObj!Wait!IconClicked;
            Cursor[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Cursor[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconMoveRequest *)
PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Cursor!clientObj!Wait!IconMoveRequest;
            Cursor[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            Cursor[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method Left *)
PROCESS Left[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort!Left
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* If *)
      (
         (* Then *)
         ([xPos eq Succ(0)]->
            EXIT(iconPic,yPos,xPos)
         )
         []
         (* Else *)
         ([Not(xPos eq Succ(0))]->
            (* Assignment *)
            (LET xPos:Int = xPos - Succ(0) IN
               (* TellCall *)
```

```
            Messages!Send!Cursor!Interface!Tell!SetIcon!xPos!yPos!iconPic;
            (* EndTellCall *)
            EXIT(iconPic,yPos,xPos)
         )
      )
      []
      (* dummy EXIT list *)
      ([True eq False]->
         EXIT(iconPic,yPos,xPos)
      )
   )
   (* EndIf *)
   >> ACCEPT iconPic:Int,yPos:Int,xPos:Int IN
   (* MethodFinish *)
   (
      ([msgDescr eq Ask]->
         Messages!Send!Cursor!clientObj!Wait!Left;
         Cursor[Messages](xPos,yPos,iconPic)
      )
      []
      ([msgDescr eq Tell]->
         Cursor[Messages](xPos,yPos,iconPic)
      )
   )
   (* EndMethodFinish *)
)
ENDPROC
(* EndMethod *)


(* Method *)
(* method Right *)
PROCESS Right[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort!Right
            [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* If *)
      (
         (* Then *)
         ([xPos eq Succ(Succ(0))]->
            EXIT(iconPic,yPos,xPos)
         )
         []
         (* Else *)
         ([Not(xPos eq Succ(Succ(0)))]->
            (* Assignment *)
            (LET xPos:Int = xPos + Succ(0) IN
               (* TellCall *)
               Messages!Send!Cursor!Interface!Tell!SetIcon!xPos!yPos!iconPic;
               (* EndTellCall *)
               EXIT(iconPic,yPos,xPos)
            )
         )
         []
         (* dummy EXIT list *)
         ([True eq False]->
            EXIT(iconPic,yPos,xPos)
         )
      )
      (* EndIf *)
      >> ACCEPT iconPic:Int,yPos:Int,xPos:Int IN
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!Cursor!clientObj!Wait!Right;
            Cursor[Messages](xPos,yPos,iconPic)
```

```
            )
            []
            ([msgDescr eq Tell]->
               Cursor[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method QueryXPos *)
   PROCESS QueryXPos[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!Cursor?msgDescr:MessageDescriptorSort!QueryXPos
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0 IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!Cursor!clientObj!Wait!QueryXPos!xPos;
               Cursor[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               Cursor[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)


(* Object *)
(* object GoLeft *)
PROCESS initGoLeft[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS GoLeft[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
         (* Assignment *)
         (LET xPos:Int = Succ(Succ(Succ(Succ(Succ(0))))) IN
            (* Assignment *)
            (LET yPos:Int = Succ(Succ(Succ(Succ(Succ(0))))) IN
               (* Assignment *)
               (LET iconPic:Int = leftArrow IN
                  (* TellCall *)
                  Messages!Send!GoLeft!Interface!Tell!SetIcon!xPos!yPos!iconPic;
                  (* EndTellCall *)
                  (* MethodFinish *)
                  GoLeft[Messages](xPos,yPos,iconPic)
```

```
                (* EndMethodFinish *)
             )
          )
       )
    )
 ENDPROC
 (* EndMethod *)


 (* Method *)
 (* method IconClicked *)
 PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
    Messages!Receive?clientObj:ObjNamesSort!GoLeft?msgDescr:MessageDescriptorSort!IconClicked
                [(msgDescr eq Ask) or (msgDescr eq Tell)];
    (let dummyPlaceHolder:Int = 0 IN
       (* TellCall *)
       Messages!Send!GoLeft!Cursor!Tell!Left;
       (* EndTellCall *)
       (* MethodFinish *)
       (
          ([msgDescr eq Ask]->
             Messages!Send!GoLeft!clientObj!Wait!IconClicked;
             GoLeft[Messages](xPos,yPos,iconPic)
          )
          []
          ([msgDescr eq Tell]->
             GoLeft[Messages](xPos,yPos,iconPic)
          )
       )
       (* EndMethodFinish *)
    )
 ENDPROC
 (* EndMethod *)


 (* Method *)
 (* method IconMoveRequest *)
 PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
    Messages!Receive?clientObj:ObjNamesSort!GoLeft?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
                [(msgDescr eq Ask) or (msgDescr eq Tell)];
    (let dummyPlaceHolder:Int = 0 IN
       (* MethodFinish *)
       (
          ([msgDescr eq Ask]->
             Messages!Send!GoLeft!clientObj!Wait!IconMoveRequest;
             GoLeft[Messages](xPos,yPos,iconPic)
          )
          []
          ([msgDescr eq Tell]->
             GoLeft[Messages](xPos,yPos,iconPic)
          )
       )
       (* EndMethodFinish *)
    )
 ENDPROC
 (* EndMethod *)
ENDPROC
(* EndObject *)


(* Object *)
(* object GoRight *)
PROCESS initGoRight[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE
```

```
PROCESS GoRight[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   IconClicked[Messages](xPos,yPos,iconPic)
   []
   IconMoveRequest[Messages](xPos,yPos,iconPic)
ENDPROC


(* Method *)
(* method Initialize *)
PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   (let dummyPlaceHolder:Int = 0 IN
      (* Assignment *)
      (LET xPos:Int = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))) IN
         (* Assignment *)
         (LET yPos:Int = Succ(Succ(Succ(Succ(Succ(0))))) IN
            (* Assignment *)
            (LET iconPic:Int = rightArrow IN
               (* TellCall *)
               Messages!Send!GoRight!Interface!Tell!SetIcon!xPos!yPos!iconPic;
               (* EndTellCall *)
               (* MethodFinish *)
               GoRight[Messages](xPos,yPos,iconPic)
               (* EndMethodFinish *)
            )
         )
      )
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconClicked *)
PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!GoRight?msgDescr:MessageDescriptorSort!IconClicked
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* TellCall *)
      Messages!Send!GoRight!Cursor!Tell!Right;
      (* EndTellCall *)
      (* MethodFinish *)
      (
         ([msgDescr eq Ask]->
            Messages!Send!GoRight!clientObj!Wait!IconClicked;
            GoRight[Messages](xPos,yPos,iconPic)
         )
         []
         ([msgDescr eq Tell]->
            GoRight[Messages](xPos,yPos,iconPic)
         )
      )
      (* EndMethodFinish *)
   )
ENDPROC
(* EndMethod *)


(* Method *)
(* method IconMoveRequest *)
PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Messages!Receive?clientObj:ObjNamesSort!GoRight?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
               [(msgDescr eq Ask) or (msgDescr eq Tell)];
   (let dummyPlaceHolder:Int = 0 IN
      (* MethodFinish *)
      (
```

```
            ([msgDescr eq Ask]->
               Messages!Send!GoRight!clientObj!Wait!IconMoveRequest;
               GoRight[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               GoRight[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)


(* Object *)
(* object IncNum *)
PROCESS initIncNum[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS IncNum[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
         (* Assignment *)
         (LET xPos:Int = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))) IN
            (* Assignment *)
            (LET yPos:Int = Succ(Succ(Succ(Succ(0)))) IN
               (* Assignment *)
               (LET iconPic:Int = plusSign IN
                  (* TellCall *)
                  Messages!Send!IncNum!Interface!Tell!SetIcon!xPos!yPos!iconPic;
                  (* EndTellCall *)
                  (* MethodFinish *)
                  IncNum[Messages](xPos,yPos,iconPic)
                  (* EndMethodFinish *)
               )
            )
         )
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method IconClicked *)
   PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!IncNum?msgDescr:MessageDescriptorSort!IconClicked
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0(* Vars *),cursorXPos:Int=0(* EndVars *) IN
         (* AskWaitCall *)
         Messages!Send!IncNum!Cursor!Ask!QueryXPos;
         Messages!Receive!Cursor!IncNum!Wait!QueryXPos?cursorXPos:Int;
         (* EndAskWaitCall *)
         (* If *)
```

```
         (
            (* Then *)
            ([cursorXPos eq Succ(Succ(Succ(Succ(0))))]->
               (* TellCall *)
               Messages!Send!IncNum!Tens!Tell!Inc;
               (* EndTellCall *)
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
            []
            (* Else *)
            ([Not(cursorXPos eq Succ(Succ(Succ(Succ(0)))))]->
               (* TellCall *)
               Messages!Send!IncNum!Units!Tell!Inc;
               (* EndTellCall *)
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
            []
            (* dummy EXIT list *)
            ([True eq False]->
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
         )
         (* EndIf *)
         >> ACCEPT cursorXPos:Int,iconPic:Int,yPos:Int,xPos:Int IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!IncNum!clientObj!Wait!IconClicked;
               IncNum[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               IncNum[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method IconMoveRequest *)
   PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!IncNum?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0 IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!IncNum!clientObj!Wait!IconMoveRequest;
               IncNum[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               IncNum[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)
```

```
(* Object *)
(* object DecNum *)
PROCESS initDecNum[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
   Initialize[Messages](xPos,yPos,iconPic)
WHERE


   PROCESS DecNum[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      IconClicked[Messages](xPos,yPos,iconPic)
      []
      IconMoveRequest[Messages](xPos,yPos,iconPic)
   ENDPROC


   (* Method *)
   (* method Initialize *)
   PROCESS Initialize[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      (let dummyPlaceHolder:Int = 0 IN
         (* Assignment *)
         (LET xPos:Int = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))))) IN
            (* Assignment *)
            (LET yPos:Int = Succ(Succ(Succ(Succ(Succ(0))))) IN
               (* Assignment *)
               (LET iconPic:Int = minusSign IN
                  (* TellCall *)
                  Messages!Send!DecNum!Interface!Tell!SetIcon!xPos!yPos!iconPic;
                  (* EndTellCall *)
                  (* MethodFinish *)
                  DecNum[Messages](xPos,yPos,iconPic)
                  (* EndMethodFinish *)
               )
            )
         )
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method IconClicked *)
   PROCESS IconClicked[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!DecNum?msgDescr:MessageDescriptorSort!IconClicked
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0(* Vars *),cursorXPos:Int=0(* EndVars *) IN
         (* AskWaitCall *)
         Messages!Send!DecNum!Cursor!Ask!QueryXPos;
         Messages!Receive!Cursor!DecNum!Wait!QueryXPos?cursorXPos:Int;
         (* EndAskWaitCall *)
         (* If *)
         (
            (* Then *)
            ([cursorXPos eq Succ(Succ(Succ(Succ(0))))]->
               (* TellCall *)
               Messages!Send!DecNum!Tens!Tell!Dec;
               (* EndTellCall *)
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
            []
            (* Else *)
            ([Not(cursorXPos eq Succ(Succ(Succ(Succ(0)))))]->
               (* TellCall *)
               Messages!Send!DecNum!Units!Tell!Dec;
               (* EndTellCall *)
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
            []
```

```
            (* dummy EXIT list *)
            ([True eq False]->
               EXIT(cursorXPos,iconPic,yPos,xPos)
            )
         )
         (* EndIf *)
         >> ACCEPT cursorXPos:Int,iconPic:Int,yPos:Int,xPos:Int IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!DecNum!clientObj!Wait!IconClicked;
               DecNum[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               DecNum[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)


   (* Method *)
   (* method IconMoveRequest *)
   PROCESS IconMoveRequest[Messages](xPos:Int,yPos:Int,iconPic:Int) :NOEXIT :=
      Messages!Receive?clientObj:ObjNamesSort!DecNum?msgDescr:MessageDescriptorSort!IconMoveRequest?a:Int?b:Int
                  [(msgDescr eq Ask) or (msgDescr eq Tell)];
      (let dummyPlaceHolder:Int = 0 IN
         (* MethodFinish *)
         (
            ([msgDescr eq Ask]->
               Messages!Send!DecNum!clientObj!Wait!IconMoveRequest;
               DecNum[Messages](xPos,yPos,iconPic)
            )
            []
            ([msgDescr eq Tell]->
               DecNum[Messages](xPos,yPos,iconPic)
            )
         )
         (* EndMethodFinish *)
      )
   ENDPROC
   (* EndMethod *)
ENDPROC
(* EndObject *)


(* process *)
PROCESS Interface[Interface,Messages] :NOEXIT :=
   (
      (* method SetIcon *)
      Messages!Receive?clientObj:ObjNamesSort!Interface?msgDescr:MessageDescriptorSort!SetIcon?xpos:Int?ypos:Int?pic:Int
                  [(msgDescr ne Wait)];
      Interface!SetIcon!clientObj!xpos!ypos!pic;
      (
         ([msgDescr eq Ask]->
            Messages!Send!Interface!clientObj!Wait!SetIcon;
            Interface[Interface,Messages]
         )
         []
         ([msgDescr eq Tell]->
            Interface[Interface,Messages]
         )
      )
```

```
   )
   []
   (
      (* call a IconClicked method *)
      Interface!IconClicked?serverObj:ObjNamesSort;
      Messages!Send!Interface!serverObj!Tell!IconClicked;
      Interface[Interface,Messages]
   )
   []
   (
      (* call a IconMoveRequest method *)
      Interface!IconMoveRequest?serverObj:ObjNamesSort?xpos:Int?ypos:Int;
      Messages!Send!Interface!serverObj!Tell!IconMoveRequest!xpos!ypos;
      Interface[Interface,Messages]
   )
ENDPROC


 PROCESS ObjectComms[Messages] :NOEXIT :=


   (* carrier text for messages of method Interface.SetIcon *)
      (* Ask or Tell message events *)
      Messages!Send
                  ?initObj:ObjNamesSort
                  !Interface
                  ?msg_descr:MessageDescriptorSort
                  !SetIcon
                  ?arg0:Int
                  ?arg1:Int
                  ?arg2:Int
                  [(msg_descr eq Ask) or (msg_descr eq Tell)];
      (
         Messages!Receive
                     !initObj
                     !Interface
                     !msg_descr
                     !SetIcon
                     !arg0
                     !arg1
                     !arg2;
         STOP
      |||
         ObjectComms[Messages]
      )
      (* end Ask or Tell message events *)
   []
      (* Wait message events *)
      Messages!Send
                  !Interface
                  ?targetObj:ObjNamesSort
                  ?msg_descr:MessageDescriptorSort
                  !SetIcon
                  [(msg_descr eq Wait)];
      (
         Messages!Receive
                     !Interface
                     !targetObj
                     !msg_descr
                     !SetIcon;
         STOP
      |||
         ObjectComms[Messages]
      )
      (* end Wait message events *)
   (* end carrier text for messages of method Interface.SetIcon *)
```

```
[]
(* carrier text for messages of method Tens.IconClicked *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !IconClicked
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !IconClicked
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !IconClicked
               [(msg_descr eq Wait)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !IconClicked
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Tens.IconClicked *)
[]
(* carrier text for messages of method Tens.IconMoveRequest *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !IconMoveRequest
               ?arg0:Int
               ?arg1:Int
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !IconMoveRequest
                  !arg0
                  !arg1
                  ;
         STOP
      |||
         ObjectComms[Messages]
```

```
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !IconMoveRequest
               [(msg_descr eq Wait)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !IconMoveRequest
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Tens.IconMoveRequest *)
[]
(* carrier text for messages of method Tens.Inc *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Inc
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Inc
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Inc
               [(msg_descr eq Wait)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Inc
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Tens.Inc *)
[]
```

```
(* carrier text for messages of method Tens.Dec *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Dec
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Dec
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Dec
               [(msg_descr eq Wait)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Dec
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Tens.Dec *)
[]
(* carrier text for messages of method Tens.QueryValue *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !QueryValue
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !QueryValue
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
```

```
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !QueryValue
                ?arg0:Int
                [(msg_descr eq Wait)];
   (
      Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !QueryValue
                   !arg0
                   ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Tens.QueryValue *)
[]
(* carrier text for messages of method Units.Inc *)
   (* Ask or Tell message events *)
   Messages!Send
                ?initObj:ObjNamesSort
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Inc
                [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !Inc
                   ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
                ?initObj:ObjNamesSort
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Inc
                [(msg_descr eq Wait)];
   (
      Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !Inc
                   ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Units.Inc *)
[]
(* carrier text for messages of method Units.Dec *)
   (* Ask or Tell message events *)
   Messages!Send
                ?initObj:ObjNamesSort
```

```
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Dec
                [(msg_descr eq Ask) or (msg_descr eq Tell)];
    (
       Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !Dec
                   ;
          STOP
       |||
          ObjectComms[Messages]
    )
    (* end Ask or Tell message events *)
    []
    (* Wait message events *)
    Messages!Send
                ?initObj:ObjNamesSort
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Dec
                [(msg_descr eq Wait)];
    (
       Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !Dec
                   ;
          STOP
       |||
          ObjectComms[Messages]
    )
    (* end Wait message events *)
(* end carrier text for messages of method Units.Dec *)
[]
(* carrier text for messages of method Cursor.Left *)
    (* Ask or Tell message events *)
    Messages!Send
                ?initObj:ObjNamesSort
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Left
                [(msg_descr eq Ask) or (msg_descr eq Tell)];
    (
       Messages!Receive
                   !initObj
                   !targetObj
                   !msg_descr
                   !Left
                   ;
          STOP
       |||
          ObjectComms[Messages]
    )
    (* end Ask or Tell message events *)
    []
    (* Wait message events *)
    Messages!Send
                ?initObj:ObjNamesSort
                ?targetObj:ObjNamesSort
                ?msg_descr:MessageDescriptorSort
                !Left
                [(msg_descr eq Wait)];
```

```
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Left
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Cursor.Left *)
[]
(* carrier text for messages of method Cursor.Right *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Right
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Right
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Ask or Tell message events *)
   []
   (* Wait message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !Right
               [(msg_descr eq Wait)];
   (
      Messages!Receive
                  !initObj
                  !targetObj
                  !msg_descr
                  !Right
                  ;
         STOP
      |||
         ObjectComms[Messages]
   )
   (* end Wait message events *)
(* end carrier text for messages of method Cursor.Right *)
[]
(* carrier text for messages of method Cursor.QueryXPos *)
   (* Ask or Tell message events *)
   Messages!Send
               ?initObj:ObjNamesSort
               ?targetObj:ObjNamesSort
               ?msg_descr:MessageDescriptorSort
               !QueryXPos
               [(msg_descr eq Ask) or (msg_descr eq Tell)];
   (
      Messages!Receive
```

```
                    !initObj
                    !targetObj
                    !msg_descr
                    !QueryXPos
                    ;
               STOP
            |||
               ObjectComms[Messages]
         )
         (* end Ask or Tell message events *)
         []
         (* Wait message events *)
         Messages!Send
                    ?initObj:ObjNamesSort
                    ?targetObj:ObjNamesSort
                    ?msg_descr:MessageDescriptorSort
                    !QueryXPos
                    ?arg0:Int
                    [(msg_descr eq Wait)];
         (
            Messages!Receive
                    !initObj
                    !targetObj
                    !msg_descr
                    !QueryXPos
                    !arg0
                    ;
               STOP
            |||
               ObjectComms[Messages]
         )
         (* end Wait message events *)
      (* end carrier text for messages of method Cursor.QueryXPos *)
ENDPROC
  (* EndObjInstDefns *)

ENDSPEC
(* EndSystem *)
```