S-R Machines: A Visual Formalism for Reactive and Interactive Systems

George W.Cherry
Thought**Tools, Inc.
5151 Emerson Road
Canandaigua, NY 14425
1-716-396-2233

Introduction. Engineers face many problems when developing realtime (reactive) systems. Some of these problems are caused by flawed notations. One example of a problematical (but widely used) notation is State Transition Diagrams. Most realtime methods, whether object-oriented or not, use State Transition Diagrams to describe realtime behavior. Cherry [1] and Harel [2] have described the shortcomings of State Transition Diagrams. Harel's Statecharts and our Stimulus-Response (S-R) Machines are new visual formalisms claiming to solve the problems of State Transition Diagrams.

This note describes S-R machines, the newest notation. The semantics of S-R Machines are derived strictly from Concurrent C++ and Ada. By means of a case study, this note describes some of the advantages of S-R Machines over State Transition Diagrams.

A Toy Vending Machine (State Transition Diagram). Figure 1 is the State Transition Diagram for a toy vending machine. It's adapted from a respected Computer Science textbook [4]. Its octagonal blobs represent states; its arrows represent state transitions. (Some authors use other shapes for states.) The blobs (states) give the machine sufficient memory to emit the right responses. The state transition labels have the format "Stimulus | Response" ("gum_request | dispense_gum" is an example) to represent stimuli to and responses from the machine. In general, a stimulus on an State Transition Diagram causes a change of state and a response. However, the response may be absent; for example, a nickel stimulus in the 10¢ state causes a transition to the 15¢ state, but no overt response.

There's much to criticize about the ergonomics, realism, and maintainability of this machine: there's no change return request nor state (sum) display; the price of the confections is absurdly low (and difficult to modify); the machine doesn't accept quarters; and the diagram doesn't explain what happens if a customer deposits a nickel or dime in the 20¢ state. Why do the authors of [4] "toy" with the vending machine problem? (I believe the answer is: an ergonomic, realistic vending machine would require over 100 blobs and 500 arrows! See below.)

A Realistic Vending Machine (S-R Machine). S-R Machines may be viewed as "black boxes" (their expected external behavior); "state machines" (their internal partitions that map stimulus/state pairs into responses and new states); and "clear boxes" (their state machines' internal mechanisms). The quoted terminology is due to Mills et al [3]. Figures 2-6 are an S-R Vending Machine broken down along these views. Figure 2 is the black-box view; we call it the interface. Figure 3 contains the state machines view; it decomposes the S-R Machine into three state machines: one for "any state"; one for {sum >= gum_price}; and another for {sum >= candy_price}. Figures 4-6 are the clear boxes for the three state machines, i.e., their bodies or internal logic. We call Figures 3-6 the S-R Machine body. It is box structured and (in CASE tools) hypergraphically linked.

For our syntax in this case study, we have chosen Ada. (We could have chosen Concurrent C.)

Figure 2 gives the syntactic interface and the expected external behavior of the Vending Machine. We've implemented it as an Ada task with six interrupt invoked entries. The stimulus-response patterns (traces) describe the expected external behavior (semantics) of the machine.

Each one of Figures 3-6 is the inside of an abstraction. Start reading any of these figures at its root node, the cross-hatched body icon.

The 1st act of the Vending Machine (Figure 3) is to elaborate the declarations of its two constants and its state variable, sum. Its second act is to execute its loop (named "events"). The loop repeatedly (1) executes its selective wait statement and (2) updates its display of sum. (Because each select alternative must conclude by updating sum, we have "bottom-factored" this operation from the three select alternatives.)

We have used an elegant principle for decomposing the Vending Machine into three subordinate state machines, a particular equivalence relation on its set of stimuli, {nickel, dime, quarter, change request, gum request, candy request}: "is selected in the same set of states as". An equivalence relation partitions a set into a set of subsets with the characteristics that the subsets are disjoint, no subset is empty, and their union is the whole set (so that every stimulus is a member of one and only one subset). The interesting fact about this kind of partition is that I programmed–for clarity–dozens of S-R Machines in several problem domains–and in every case I unwittingly used this equivalence relation to partition the S-R Machines into maintainable lower-level state machines. Now that I've perceived the mathematical pattern of these many decompositions, I proceed to good decompositions faster.

The semantics of the select statement are strictly Concurrent C and Ada. When control reaches the select statement, it evaluates all its guards. If a guard is true, than the select alternative is open. Next, the select statement checks whether a stimulus is pending for an open select alterative. If so, the select statement accepts the stimulus and performs the associated actions; if not, it nonbusily waits for a stimulus to any open select alternative. Note that the select statement waits nondeterministically for a stimulus; this is an especially useful mechanism for event-driven (reactive) systems.

Note the guards on Figure 3. They are computed sets of states (for example: {sum >= gum_price}). Notice the textboxes on Figures 4-6. The state variable transitions are effected by computations and assignment statements (for example: sum := sum – gum_price). (Since Computer Science is the Science of Computing, wouldn't it be better to teach students a computational mechanism like S-R Machines–rather than the unsystematic, noncomputational mechanism of STDs, which, like the abacus, should have only historical interest?)

How to avoid blob and arrow explosion. To even attempt drawing a realistic State Transition Diagram, we must limit its number of states (values of sum), and then provide some mechanism to block coin inputs which could make sum larger than the biggest and last state. We'll assume some mechanism that prevents the customer driving sum above $5.00. Still, the news is bad:

The State Transition Diagram must have 101 blobs for the states: 0¢, 5¢, 10¢, 15¢, ..., 495¢, 500¢.

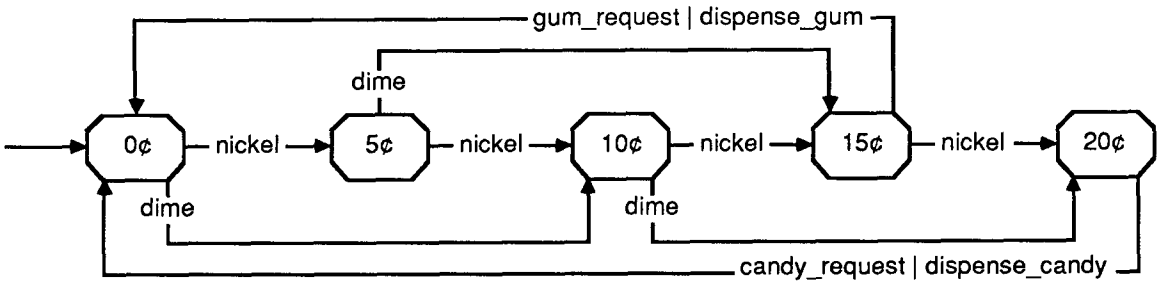
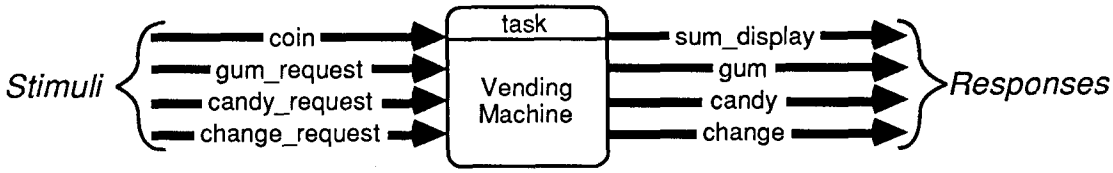


Figure 1. Finite State Toy Vending Machine (State Transition Diagram)



task declaration

```
task Vending_Machine is
   -- The following six entries are called by interrupts.
   entry nickel; entry dime; entry quarter; -- coin entries
   entry gum_request;
   entry candy_request;
   entry change_request;
end;
```

sample stimulus-response patterns for gum_price = 25¢, candy_price = 50¢

```
<0.0, quarter, 0.25, quarter, 0.50, quarter, 0.75, quarter, 1.00, quarter, 1.25, gum_request, gum, 1.00>
<0.0, dime, 0.10, nickel, 0.15, nickel, 0.20, gum_request, null, coin_return_request, change(0.20), 0.0>
<0.0, nickel, 0.5, dime, 0.15, quarter, 0.40, candy_request, null, dime, 0.50, candy_request, candy, 0.0>
```

Figure 2. Stimulus-Response Vending Machine Interface

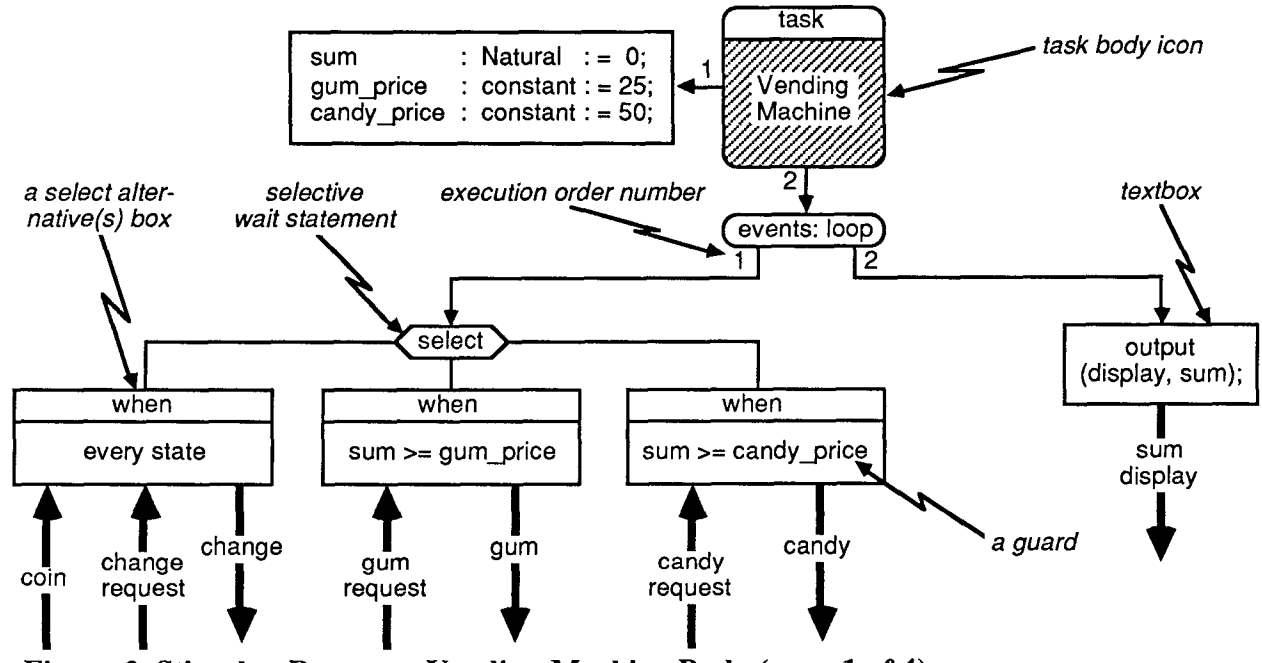


Figure 3. Stimulus-Response Vending Machine Body (page 1 of 4)

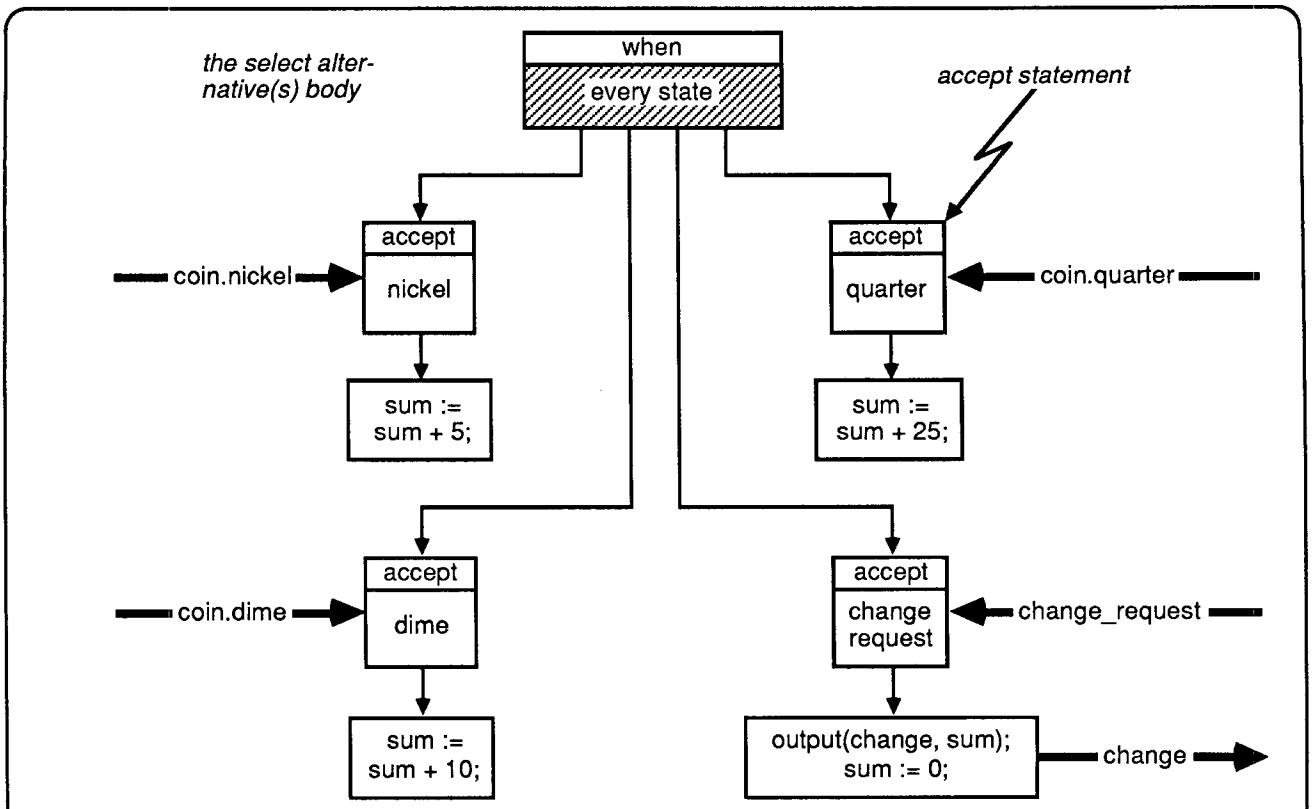


Figure 4. Stimulus-Response Vending Machine Body (page 2 of 4)

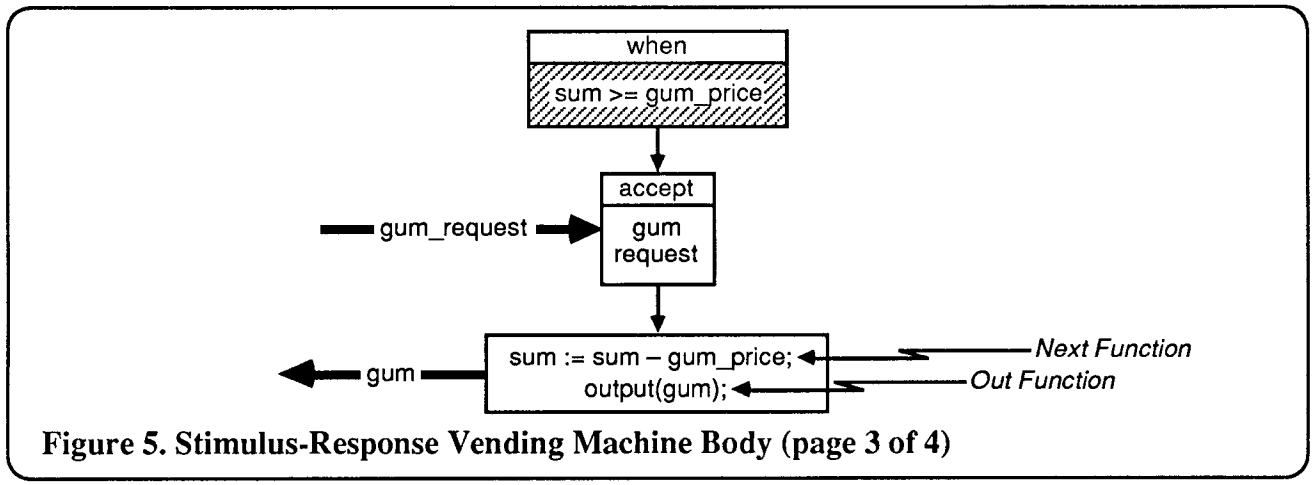


Figure 5. Stimulus-Response Vending Machine Body (page 3 of 4)

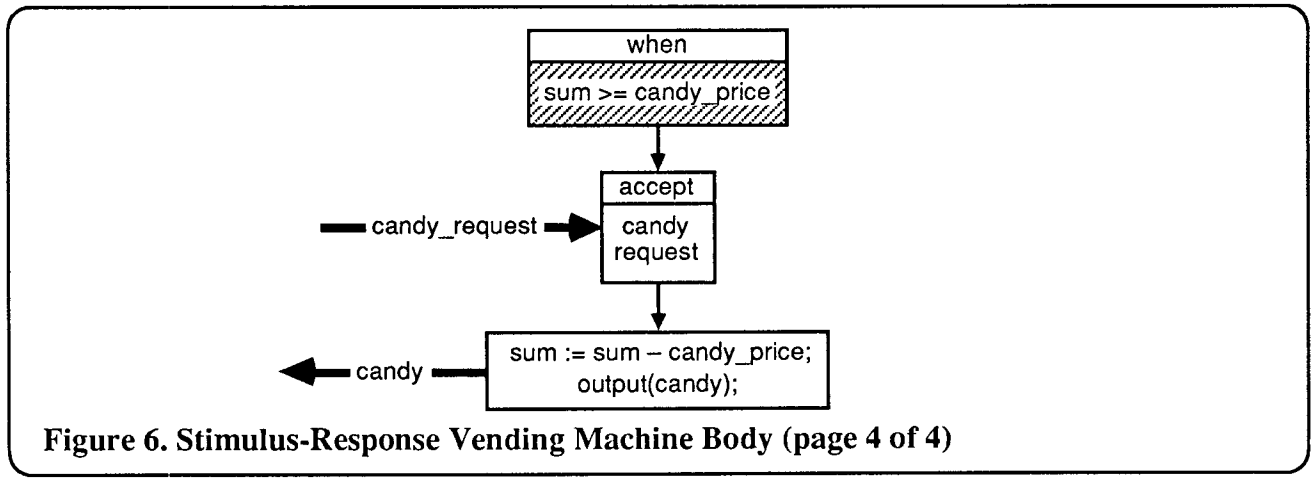


Figure 6. Stimulus-Response Vending Machine Body (page 4 of 4)

The State Transition Diagram must have (assuming gum is 25¢ and candy is 50¢):

100 arrows for state transitions caused by nickel inputs;
99 arrows for state transitions caused by dime inputs;
96 arrows for state transitions caused by quarter inputs;
101 arrows for state transitions caused by coin return requests;
96 arrows for state transitions caused by gum requests;
91 arrows for state transitions caused by candy requests;

for a total of 583 arrows! This STD cannot be drawn on a 8.5 by 11 inch page; nor can it be partitioned.

The Stimulus-Response (S-R) Machine has about 25 blobs and less than 50 arrows.

But the S-R Machine has (assuming Natural'last = 32765) 6,553 states!

To change the price of gum to 30¢ and the price of candy to 75¢ necessitates deleting many arrows and moving hundreds of arrows.

Of course, you can always avoid blob and arrow explosion by using S-R Machines. (And to make the same price-changes on the S-R Machine requires changing only 4 characters on page 1 of the Vending Machine's body.

References

1. Cherry, George. W. *Software Construction by Object-Oriented Pictures: Specifying Reactive and Interactive Systems.* Thought**Tools, Canadaigua, N.Y. 1990. [This book describes the *SCOOP-3™* method, including S-R Machines (also called Abstract State Machines). You may order it from Dorset House by calling 1-800-342-6657.]

   ™SCOOP-3 is a trademark of Thought**Tools, Inc.

2. Harel, D. "On Visual Formalisms", *Communications of the ACM*, May, 1988, 514-530. [Harel's criticisms of State Transition Diagrams led to his Statecharts and our S-R Machines.]

3. Mills, H.D., Linger, R.C., and Hevner, A.R. "Box-Structured information systems". In *Software State-of-the-Art: Selected Papers,* T. DeMarco, and T. Lister Eds. Dorset House Publishing, New York, N.Y. 1990, 322-339. [You may order this book from 1-800-342-6657]

4. Wulf, W. A., Shaw, M., and Hilfinger, P. N, and Flon , L. *Fundamental Structures of Computer Science.* Addison-Wesley, Reading, Massachusetts, 1981.

Appendix: Ada Source Code

```
task body Vending_Machine is
   sum            : Natural  :=  0;
   gum_price      : constant := 25;
   candy_price    : constant := 50;
begin
   events: loop
      select
         accept nickel do
            sum := sum + 5;
         end;
      or
         accept dime do
            sum := sum + 10;
         end;
      or
         accept quarter do
            sum := sum + 25;
         end;
      or
         accept change_request do
            output(change, sum);
            sum := 0;
         end;
      or
         when sum >= gum_price =>
         accept gum_request do
            sum := sum - gum_price;
            output(gum);
         end;
      or
         when sum >= candy_price =>
         accept candy_request do
            sum := sum - candy_price;
            output(candy);
         end;
      end select;
      output(display, sum)
   end loop;
end Vending_Machine;
```