SDL

Prof. Stephen A. Edwards

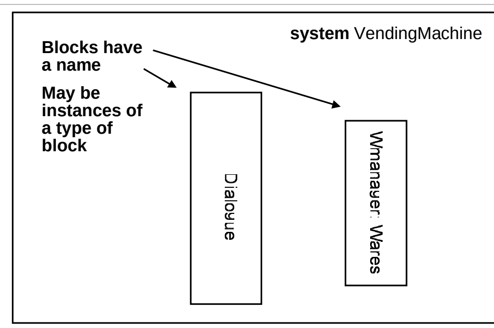
The Most Generic Name Yet

- SDL = Specification and Description Language
- Grew out of the European telecommunications world
- Good for describing protocols implemented on distributed systems
- Both textual and formal graphical syntax

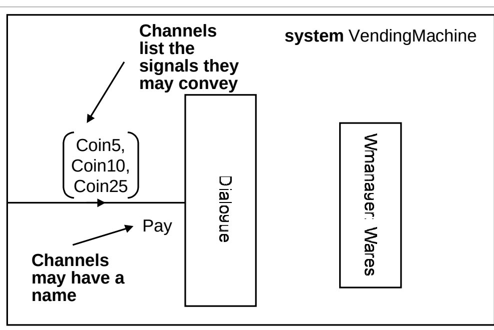
Three Components in SDL Systems

- **System**
  - Collection of concurrently-running blocks
  - Blocks communicate through explicit channels
  - Represents distributed, communicating computers
- **Block**
  - Collection of concurrently-running processes or collection of blocks
  - Blocks communicate through explicit channels
  - Represents a single processor
- **Process**
  - Extended finite-state machine

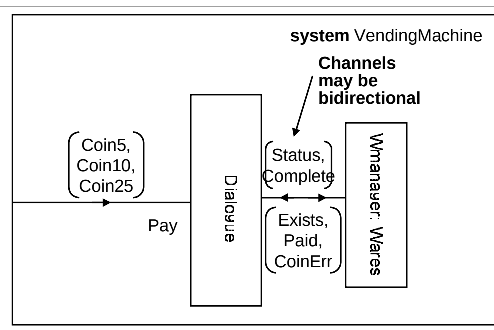
Vending Machine System



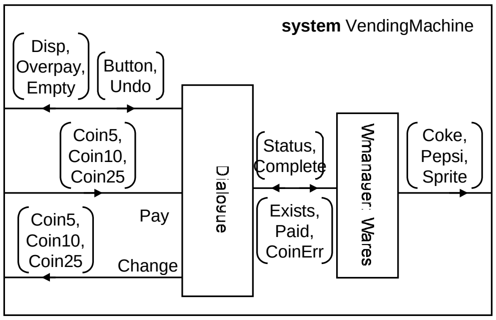
Vending Machine System



Vending Machine System



Vending Machine System



Communication in SDL

SDL Communication

- **Processes, blocks, and systems communicate through signals conveyed through channels**
- **Signal is a message corresponding to an event, e.g.,**
  - **Ring**
  - **HangUp**
  - **Dial**

SDL Signals

- **Pure signals have no value**
  - **Ring**
  - **Hangup**
- **Valued signals also convey data**
  - **dial(digit)**
- **SDL's type system for values fairly complex**

Signals Have Addresses

- **Signals may include the address of the process that sent them**
- **This is useful for distinguishing among multiple instances of a single process**
- **Each process may correspond to, say, a different call in progress**
  - **Which call just hung up?**

SDL Communication

- **Communication within a block (computer) is assumed instantaneous**
  - **Assumed quick because it's all on the same processor**

- **Communication between blocks has uncontrollable delays**
  - **Assumed slow because it is done across long distances**

SDL Channels

- Signals travel between blocks and processes through channels
- Channel: point-to-point connection that defines which signals may travel along it
- A signal may traverse many channels before reaching its destination

SDL Processes

SDL Processes

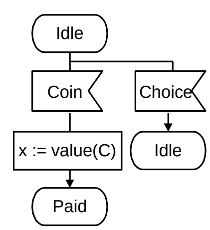
- Each process is a finite-state machine
- Each process has a single input signal queue
- Execution: remove next signal from queue and react
  - Make decisions
  - Emit more signals
  - Compute the next state
- Processes may be created and terminate while system is running

SDL Processes

Textual form

```
state Idle;
  input Coin(C);
    task x := value(C);
    nextstate Paid;
  input Choice;
    nextstate Idle;
endstate Idle;
```

Graphical form

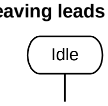


SDL Process States

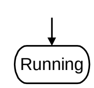
- At a particular state,
- A signal is removed from the queue
- If a transition defined for the signal in current state,
  - Run the transition
  - Transmit signals
  - Update internal variables
  - Choose a next state
- If no transition defined for the signal in current state,
  - Discard the signal
  - Leave the state unchanged

The State Symbol

- Can denote both a current and a next state
- Line leaving leads to rules for a current state

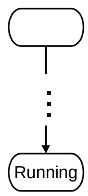


- Arrow entering means a next state



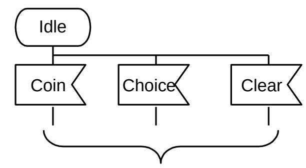
The Start Symbol

- Denotes where the execution of a process begins
- Nameless state



The Receive Symbol

- Appears immediately after a state
- Indicates which signal triggers each transition



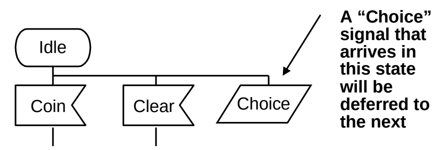
Lead to diagrams for each transition

Received Signals

- Complete Valid Input Signal Set
  - Set of all signals that the process will ever accept
  - An error occurs if a signal outside this set is received
- In any state, only certain signals may have a transition
  - A valid signal that has no transition is simply discarded without changing the state
  - The "implicit transition"

The Save Symbol

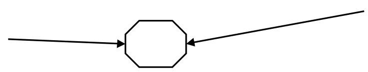
- Like receive, but instead pushes the signal back in the queue



- Designed for handling signals that arrive out of order

The Save Symbol

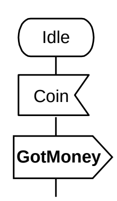
- Single process input queue totally orders the sequence of events that arrive at a process
- What if two events arrive from different processes at more-or-less the same time?



- The save symbol can be used to dictate the order in which signals that arrive out of order are processed

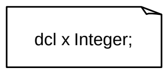
The Output Symbol

- Send a signal to another process
- Which channel to send it on usually follows from its type



Local Variables

- An SDL process has local variables it can manipulate
- Partially shared variables
  - Only the owning process may write a variable
  - Other processes may be allowed to read a variable
- Variables are declared in a text annotation

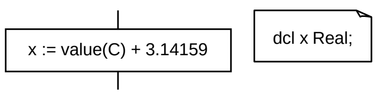


SDL Sorts

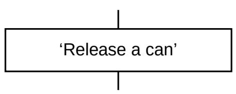
- Each variable is of a particular “sort” (type)
  - Possible values (e.g., integer numbers)
  - Operators on those values (e.g., +, *)
  - Literals (e.g., “zero”, “1”, “2”)
- Built-in sorts: integer, Boolean, real, character, and string
- Can be combined in structures, arrays, enumerations, and sets

Task Symbol

- Assignment of variable to value of expression

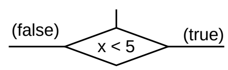


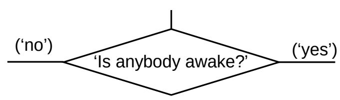
- Informal text
  - Produces an incomplete specification
  - Intended to be later refined



The Decision Symbol

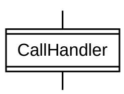
- A two-way branch that can check a condition
- Can be an expression or informal



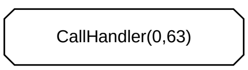


Process Creation Symbol

- A transition can cause another process to start



- Communication channels stay fixed
- Processes marked with initial and maximum number of copies that can be running



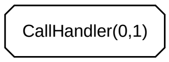
Process Creation

- Intended use is in a “server” style
- A new connection (call, interaction, etc.) appears
- A new server is created to handle this particular interaction
- It terminates when it has completed the task (e.g., the user hangs up the phone
- Maximum number of processes usually for resource constraints
  - Can’t handle more than 64 simultaneous calls without exhausting processor resources

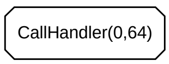
Process Creation

- Process is always running

- Process starts dormant. At most one instance of the process ever runs

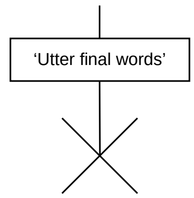


- As many as 64 copies of the process can be running

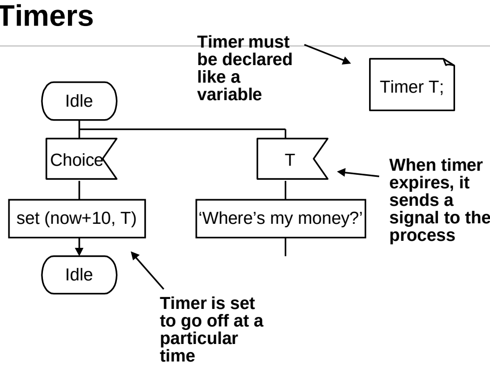


Process Termination

- A process can only terminate itself



Timers



Implementing an SDL system

Implementation

- Event-driven programming
- Each process is an infinite loop

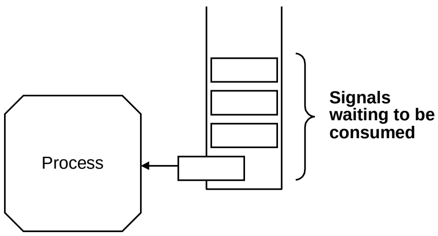
```
for (;;) {
  event = get_next_event();
  dispatch_handler(event, current_state);
}
```


Implementation

- Typical implementation:
- Code for each signal/current state pair becomes a separate function
- Pointers to all of these functions placed in a big table and called by main dispatcher
- No handler for a signal in a particular state: signal discarded and machine remains in the same state

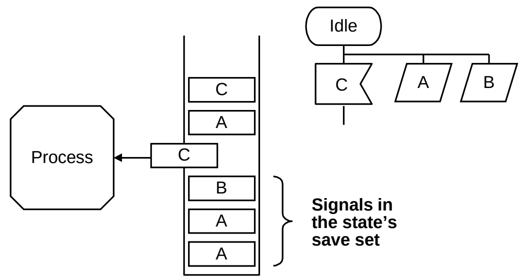
Implementing Input Queues

- Each process has a single input queue from which it consumes signals



Implementing the Save Operator

- Signals at the beginning of the queue in the current state's save set are ignored



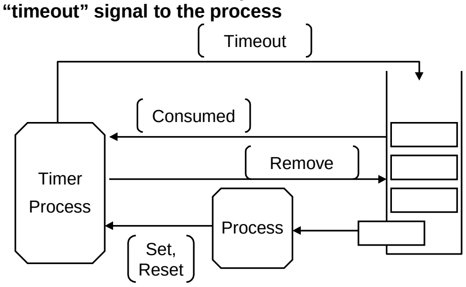
Implementing the Save Operator

- Search though signals in the queue starting at the head
- Consume the first one not in the save set

- Implications:
- Input queue is not a FIFO
- Need the ability to delete signals in the middle of the queue
- Suggests a linked-list implementation
- Fussy to make it work with a circular buffer

Implementing Timers

- In effect, a timer creates a process that feeds a "timeout" signal to the process



Implementing Timers

- Process starts a timer by signaling "set" to the timer
  - Timer signals queue to delete any unconsumed Timeout signals
- Process stops a timer by signaling "reset" to the timer
  - Timer signals queue to delete any pending Timeout signals
- When timer expires, it send a "Timeout" signal to the queue
  - Timeout behaves like a normal signal
  - When Timeout signal consumed, queue signals timer, which then shuts off.

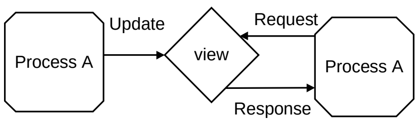
Implementing Communication

- Channels have FIFO behavior
  - A signal can't overtake another if they're traveling along the same channel

- Channels have nondeterministic delay
  - Signals sent along two parallel channels may arrive in any order

Implementing Viewed Variables

- If process A *reveals* its variable v, then process B may *view* the value of process A's variable v
- Conceptually, this is handled by a view process that maintains all viewed variables
- Revealers send updates to the view process
- Viewers send requests to view process

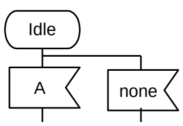


Nondeterminism

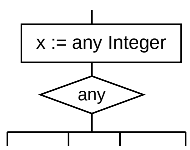
- Fundamentally nondeterministic because of implicit signal merge
- When two processes send signals to a third process at a single time, they arrive in some undefined order
- State machines usually sensitive to signal arrival order
- Save construct provides a way to handle some cases

Explicit Nondeterminism

- Spontaneous transition
  - Process may nondeterministically proceed down the "none" branch, even if a signal is waiting



- Nodeterminstic value:
- Nondeterminstic choice:



How SDL is used

- Originally intended as a system specification
- Meant to be interpreted by people, not automatically
- Sufficiently formal to enable mathematical reasoning about its behavior
- Intended to be more precise that English text or ad-hoc graphical specifications (flowcharts, etc.)
- Still its main use

How SDL is used

- Telelogic's Tau system
  - Graphical SDL system entry
  - Simulation of SDL systems
  - Automatic code generation
- Automatic code generation facilities not usually used for production
  - Code quality insufficient?
- Used mostly for system simulation
  - Much like Matlab is used for specifying and simulating signal processing algorithms

Summary

- SDL designed for specifying telecommunications protocols
- Not designed as a programming or modeling language per se
- Intended more as an improvement over English of specifying desired behavior
- System designers would devise specification, then hand it to implementers, who would perform their task manually

Summary

- Describes distributed systems composed of computers running concurrent processes
- Communication channels have FIFO behavior
- Each channel marked with the signals (messages) that may travel along it
- Processes are extended finite-state machines
- Each has a single input signal queue

Summary

- Graphical and textual syntax
  - Graphical syntax based on block diagrams and flowcharts
  - Textual syntax looks a little like Pascal
- Fundamentally nondeterministic
  - Nondeterminstic delays through communication channels
  - Implicit merge at the input to each process
  - Save construct give some ability to handle out-of-order arrivals due to nondeterminism
  - Some explicitly nondeterministic constructs

Summary

- Is this used?
- In telecom, fairly widely
- Outside, not as much
- A specification language
  - Not designed to be implemented automatically
  - At least one automatic system exists, mostly used for simulation
- Not a modeling langauge
  - Can't say anything about what actual delays are

Most Important Points

- Computational model:
  - Concurrent processes
  - Processes are finite-state machines described using flowcharts that may manipulate variables
  - Each process has a single input queue that collects signals from every process
- Explicit listing of what signals may travel through what channels