ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Μεταπτυχιακό Πρόγραμμα Σπουδών

Θεωρία και Σχεδιασμός πρωτοκόλλων

Αθανασία Αλωνιστιώτη
nancy@di.uoa.gr

Based on Lectures notes from: * Oleg Chistokhvalov, LUT, IT Dep. ,* Dr. Junzhao Sun U. OULU Dept. Electrical & Information Engineering, Computer Engineering Lab

Outline

- Introduction to SDL
  - Language
  - Purpose & Application
  - Key SDL features
- Static SDL Components
  - Description of the System Structure
  - Concepts of System, Block and Process
  - Communication Paths: Channels, Signals
- Dynamic SDL Component
  - State, Input, Output, Process, Task, Decision, Procedure …
  - Data in SDL
  - Inheritance
  - Block and Process Sets
- Examples

Language

- Γλώσσα περιγραφής συστημάτων και πρωτοκόλλων.
- Προτάθηκε από τον οργανισμό ITU-T (Recommendation Z.100.)
- Καλύπτει ποικίλους τομείς διαδραστικών και κατανεμημένων συστημάτων (π.χ., δικτυακά πρωτόκολλα, hardware).
- Graphic Representation (SDL/GR) & textual Phrase Representation (SDL/PR).
- Ένα σύστημα προδιαγράφεται ως ένα σύνολο διασυνδεδεμένων «μηχανών» που είναι επέκταση των μηχανών πεπερασμένων καταστάσεων.
- Ο φορμαλισμός της γλώσσας επιτρέπει την προσομοίωση αλλά και την αυτόματη παραγωγή κώδικα.

Language

- Πρώτη έκδοση 1976.
- Το 1988 -> SDL-88 περιέχει object oriented concepts (π.χ., inheritance, abstract generic types etc).
- Το 1992 βελτιωμένη υποστήριξη για υλοποιήσεις.
- Η SDL-2000 είναι η τελευταία έκδοση η οποία βασίζεται αποκλειστικά σε object-orientation.
- Κατά την προσομοίωση (simulation) ενός συστήματος παρέχεται η γραφική αναπαράσταση της διαδοχής των σημάτων με τα MSCs (Message Sequence Charts).

Introduction to SDL (Purpose & Application)

Why SDL exists ?

- The purpose of SDL is to be a language for unambiguous specification and description of the structure, behaviour and data of telecommunications systems.

- The terms specification and description are used with the following meaning:

  - a specification of a system is the description of its required behaviour
  - a description of a system is the description of its actual behaviour, that is its implementation.

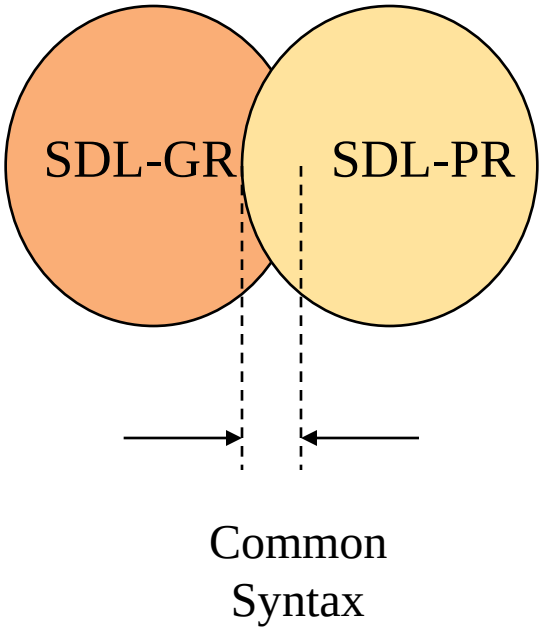
Introduction to SDL (Purpose & Application)

Why SDL exists ?

- SDL may be used for producing
  - Specification and Design of diverse applications: aerospace, automotive control, electronics, medical systems,
  - Telecommunications Standards and Design for (examples):
    - Call & Connection Processing,
    - Used Worldwide for all kinds of complex, communication systems
    - 3G, Cellular Phones, switches , WLANS, Bluetooth device etc,
    - Maintenance and fault treatment (for example alarms, automatic fault clearance, routine tests) in general telecommunications systems,
    - Intelligent Network (IN) products,
    - Mobile handsets and base stations,
    - Satellite protocols,
- Increasingly used to generate product code directly with help of tools like ObjectGeode, Tau/SDT, Cinderella

SDL Representations

- SDL has two representation forms:
  - SDL-GR - graphical representation
  - SDL-PR - textual, phrase representation



Επαυξημένες μηχανές πεπερασμένων καταστάσεων

- Μπορούμε με FSM να περιγράψουμε μεταβλητές
- Πόσες καταστάσεις χρειαζόμαστε;
- Εισαγωγή μεταβλητών, πράξεις (π.χ., εκχωρήσεις), συνθήκες.
- Η κατάσταση μιας μηχανής είναι τώρα η κατάσταση και η τρέχουσα τιμή των μεταβλητών.
- Τις μεταβλητές μιας μηχανής μπορούν να τις βλέπουν άλλες, αλλά αλλαγές γίνονται από τις άλλες μόνο έμμεσα (π.χ., με κατάλληλο σήμα).

Ιεραρχική Δομή

- Σε επίπεδο *συστήματος* συναντάμε οντότητες τύπου block, οι οποίες επικοινωνούν μεταξύ τους και με το περιβάλλον μέσω διαδρομής καναλιών (channel routes),
  - Κάθε τύπος block μπορεί να έχει ένα ή περισσότερα στιγμιότυπα.
  - Κάθε στιγμιότυπο καθορίζεται από μια δεδομένη χρονική στιγμή, όπου οι διεργασίες που περιέχονται στο block βρίσκονται σε συγκεκριμένες καταστάσεις και οι μεταβλητές έχουν συγκεκριμένες τιμές.

- Σε επίπεδο **block** συναντάμε οντότητες τύπου διεργασίας (process), οι οποίες επικοινωνούν μεταξύ τους μέσω διαδρομών σημάτων (signal routes). Οι διαδρομές καναλιών του επιπέδου συστήματος καταλήγουν σε μία ή περισσότερες διαδρομές σημάτων.

- Σε επίπεδο **process** συναντάμε μηχανές πεπερασμένων καταστάσεων, οι οποίες βρίσκονται σε συγκεκριμένες καταστάσεις και μεταβαίνουν σε άλλες με την αποστολή και λήψη μηνυμάτων. Κατά τις μεταβάσεις αυτές μπορούν να μεταβάλλουν τις τιμές κάποιων μεταβλητών . Μια διεργασία είναι είτε αυθύπαρκτη είτε δημιουργείται κατά την εκτέλεση της εφαρμογής από μια άλλη διεργασία.

Static & Dynamic SDL

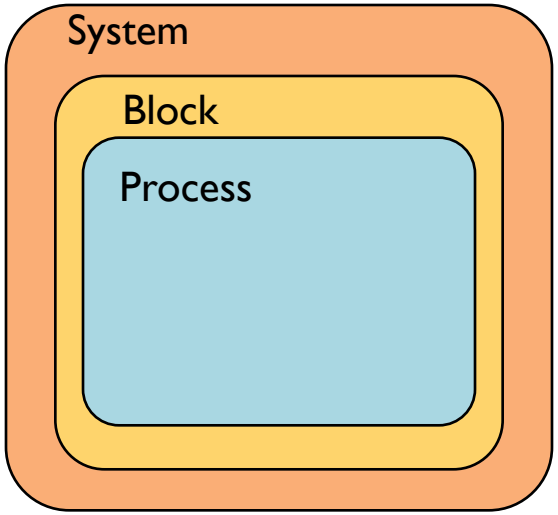
- SDL has a static component, and a dynamic component.

- The Static component describes/specifies system structure
  - Functional decomposition to sub-entities
  - How they are connected
  - What signals they use to communicate

- The Dynamic component describes/specifies system operation - behavior
  - SDL Transitions, Transitions Actions
  - Communications
  - Birth, Life and Death of Processes

Static SDL

- **System** is the highest level of abstraction
- A system can be composed of 1 or more **blocks**
- A block can be composed of processes and blocks
- **Processes** are finite state machines, and define dynamic behavior



Static SDL Terms

- agent

The term agent is used to denote a system, block or process that contains one or more extended finite state machines.

- system

A system is the outermost agent that communicates with the environment.

- block

A block is an agent that contains one or more concurrent blocks .

- process

A process is an agent that contains an extended finite state machine, and may contain other processes.

System,Block, and Process

❑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
•Communicate between each other and with environment using signals
•Processes have explicit and variable defined implicit state

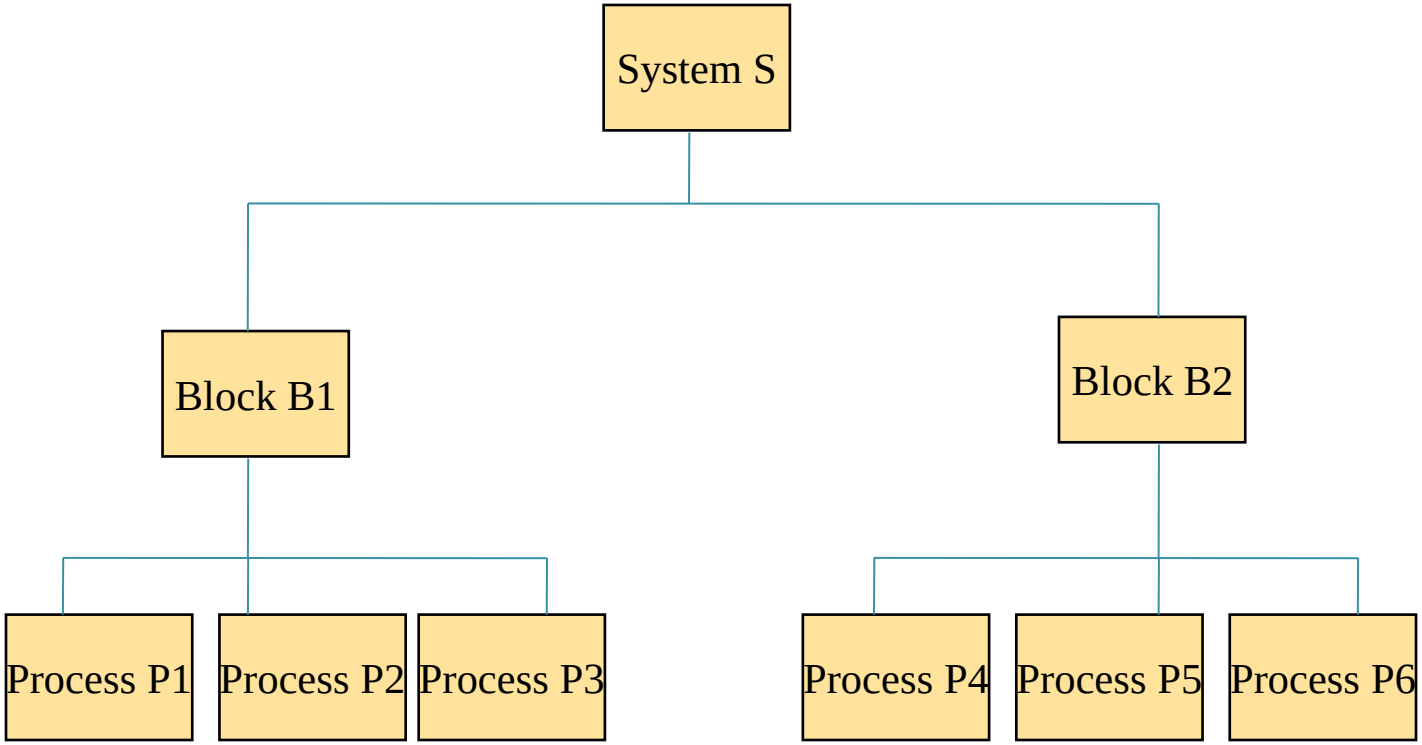
System Decomposition

- When dealing with large and complex systems it is best to decompose down to the manageable size functional components: BLOCKs ("Divide and Conquer").

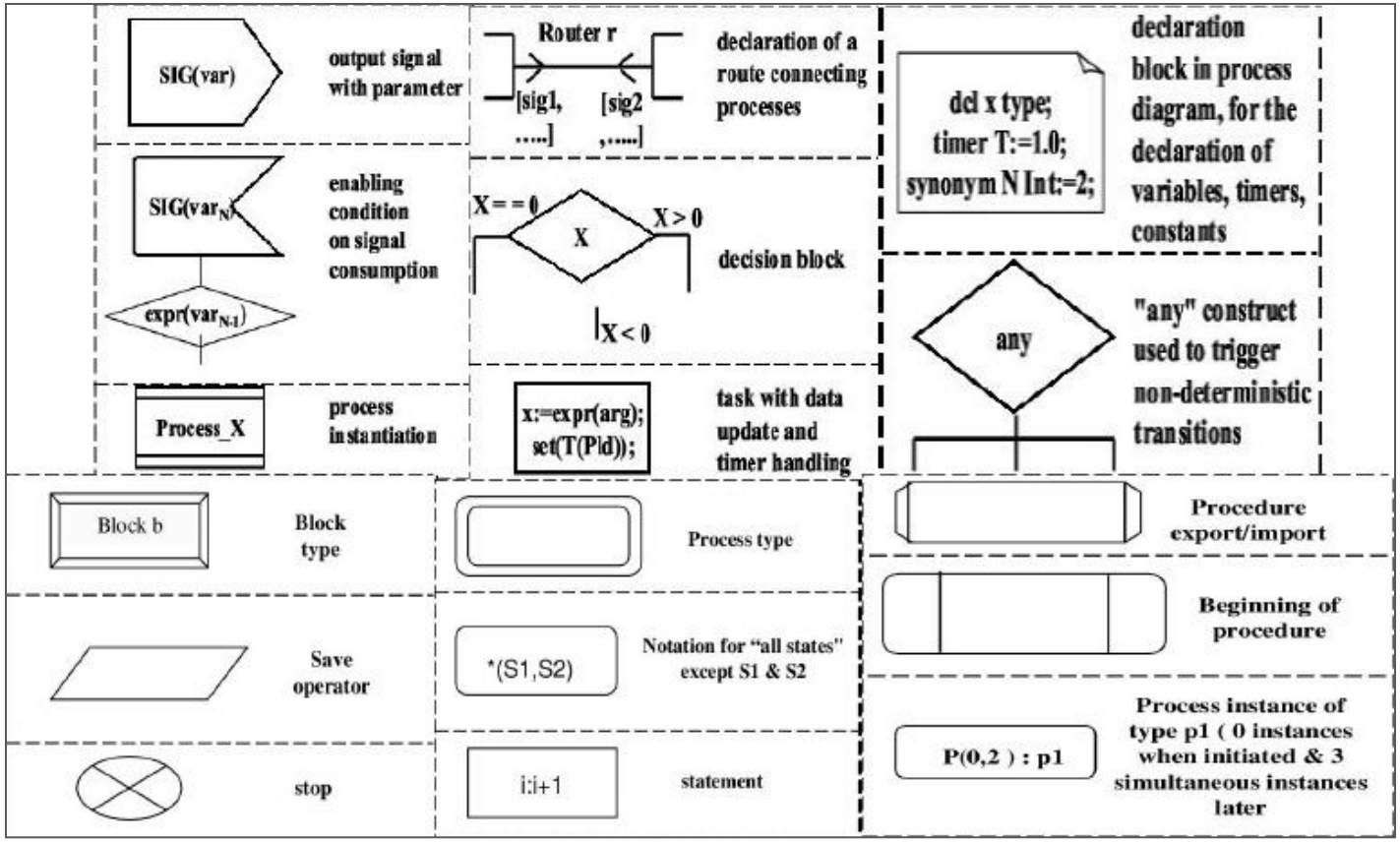
- Follow natural subdivisions: BLOCKs may correspond to actual software/hardware modules.

- Minimise interfaces between BLOCKs in terms of the number and volume of signals being exchanged.

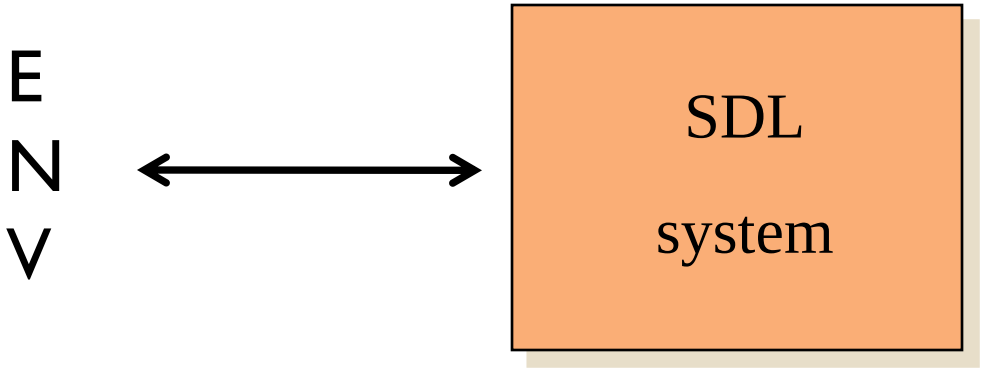
Structuring of the System Description



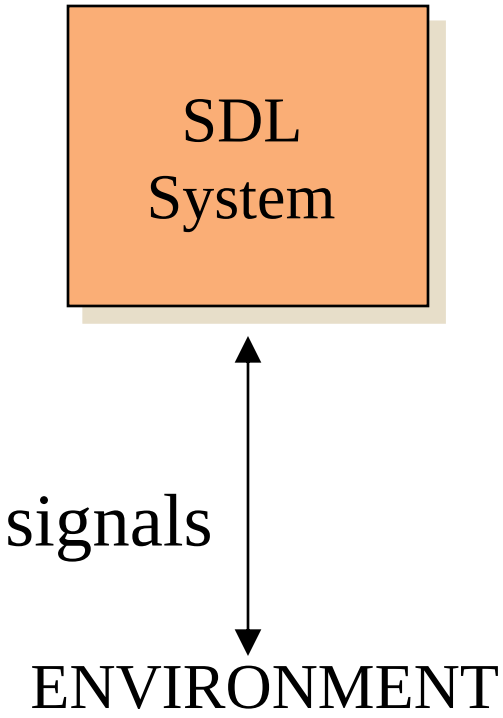
SDL Syntax


SIG(var)
output signal with parameter
Router r
[sig1, ...]
[sig2 ,.....]
declaration of a route connecting processes
dcl x type;
timer T:=1.0;
synonym N Int:=2;
declaration block in process diagram, for the declaration of variables, timers, constants
SIG(var_N)
expr(var_N-1)
enabling condition on signal consumption
X = = 0
X
X > 0
X < 0
decision block
any
"any" construct used to trigger non-deterministic transitions
Process_X
process instantiation
x:=expr(arg);
set(T(Pid));
task with data update and timer handling
Block b
Block type
Process type
Procedure export/import
Save operator
*(S1,S2)
Notation for "all states" except S1 & S2
Beginning of procedure
stop
i:i+1
statement
P(0,2) : p1
Process instance of type p1 (0 instances when initiated & 3 simultaneous instances later

Σύστημα και Περιβάλλον



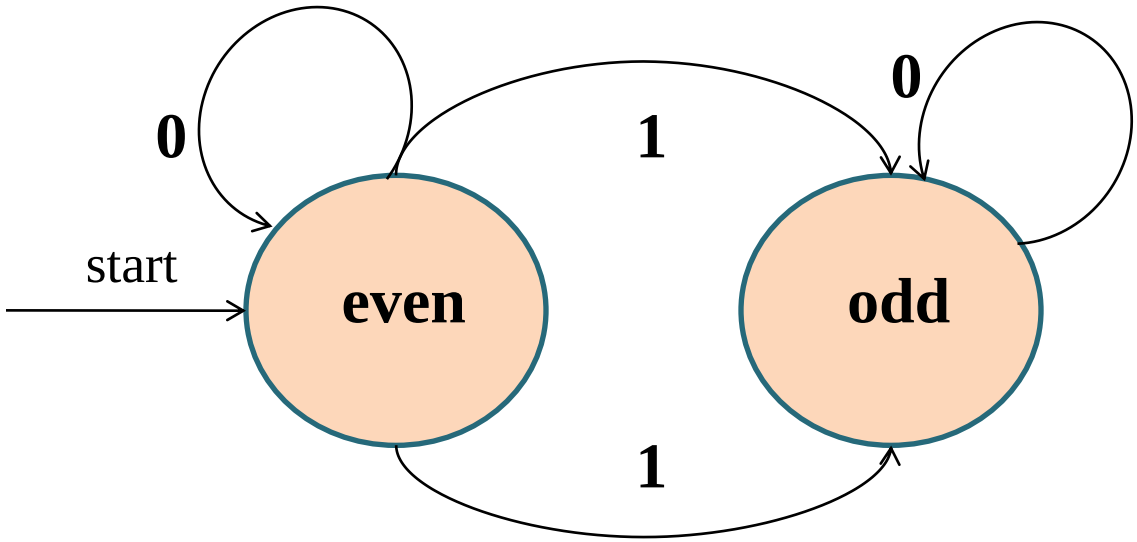
System and Environment



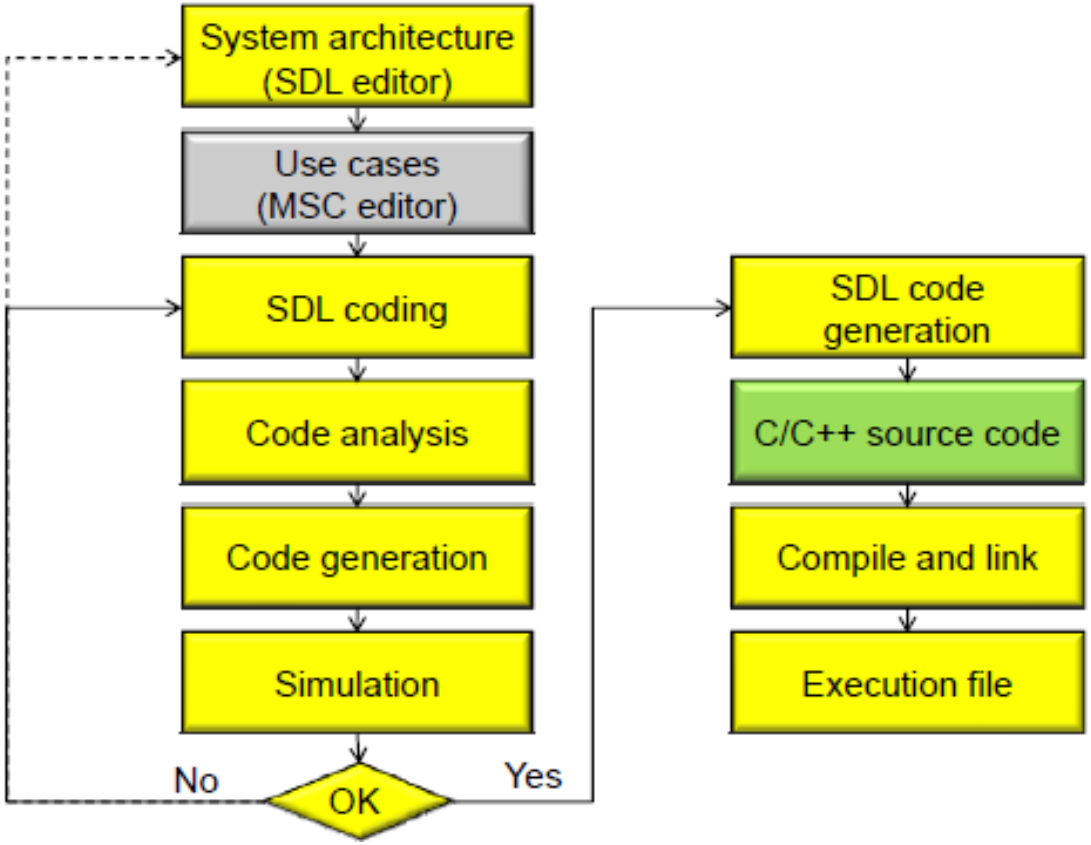
- The SDL specification defines how Systems reacts to events in the Environment which are communicated by Signals sent to the System

- The only form of communication of an SDL system to environment is via Signals

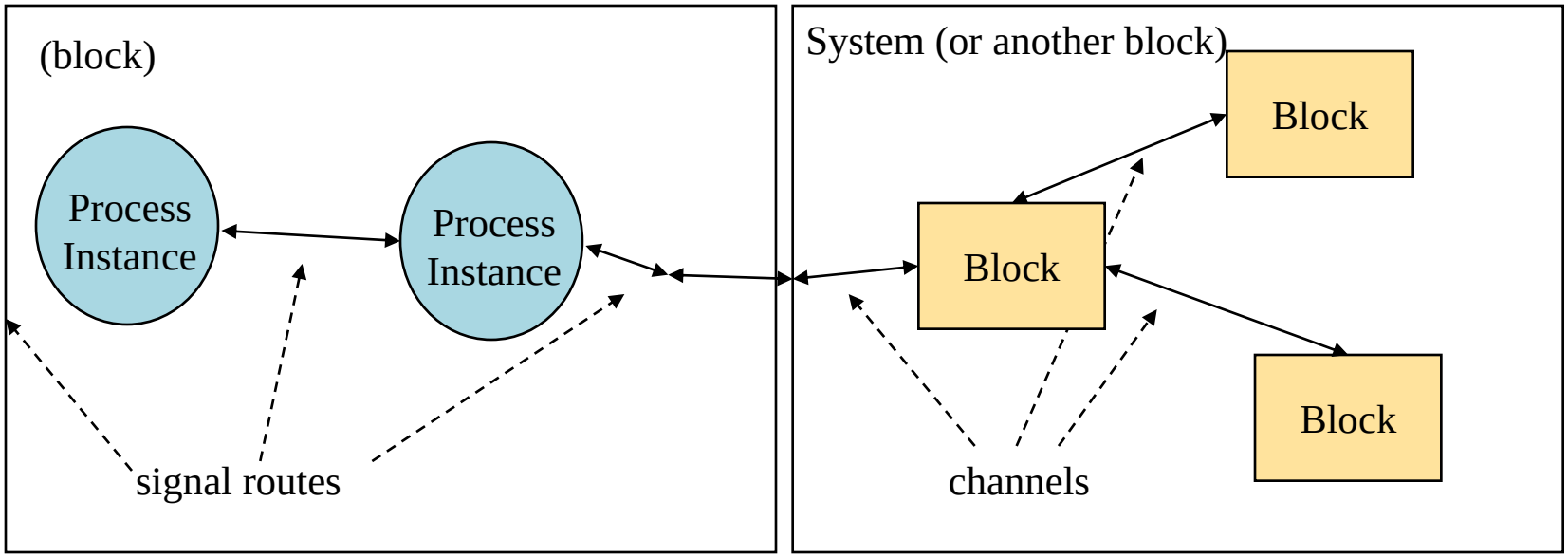
Μηχανές Πεπερασμένων Καταστάσεων



Basic Concept of Design Flow

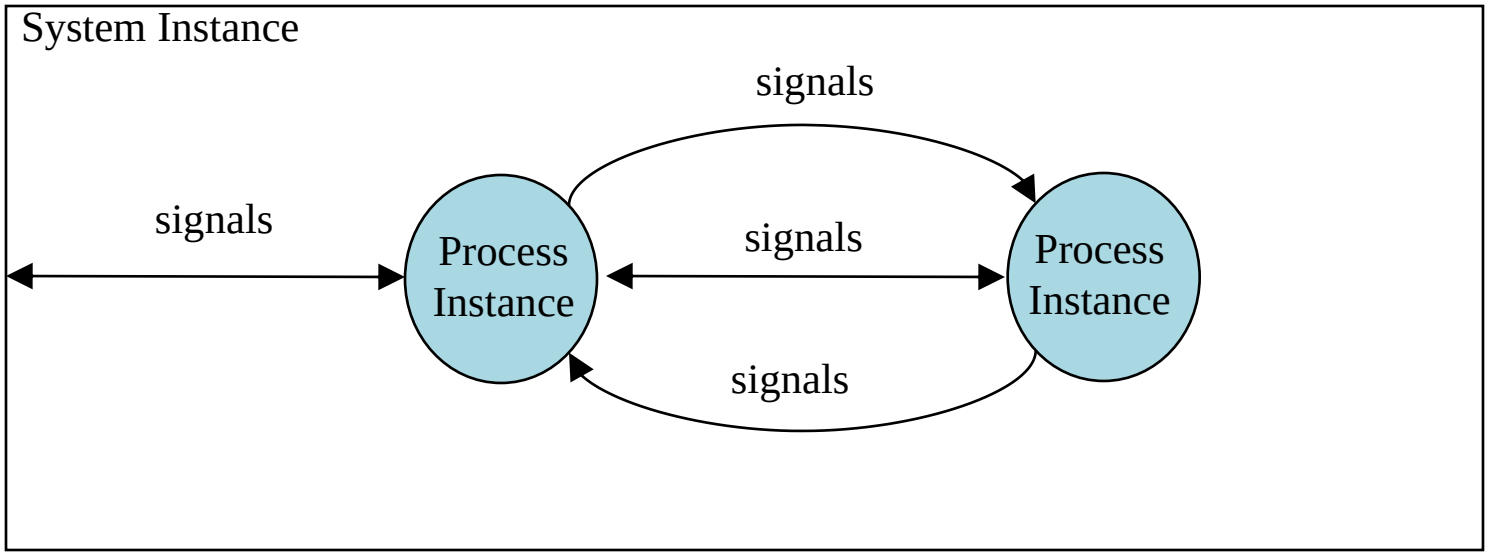


SDL Overview Blocks



- Large number of process without structure leads to loss of overview
- Blocks are used to define a system structure
- Signal routes transfer signal immediately while channels may be delaying

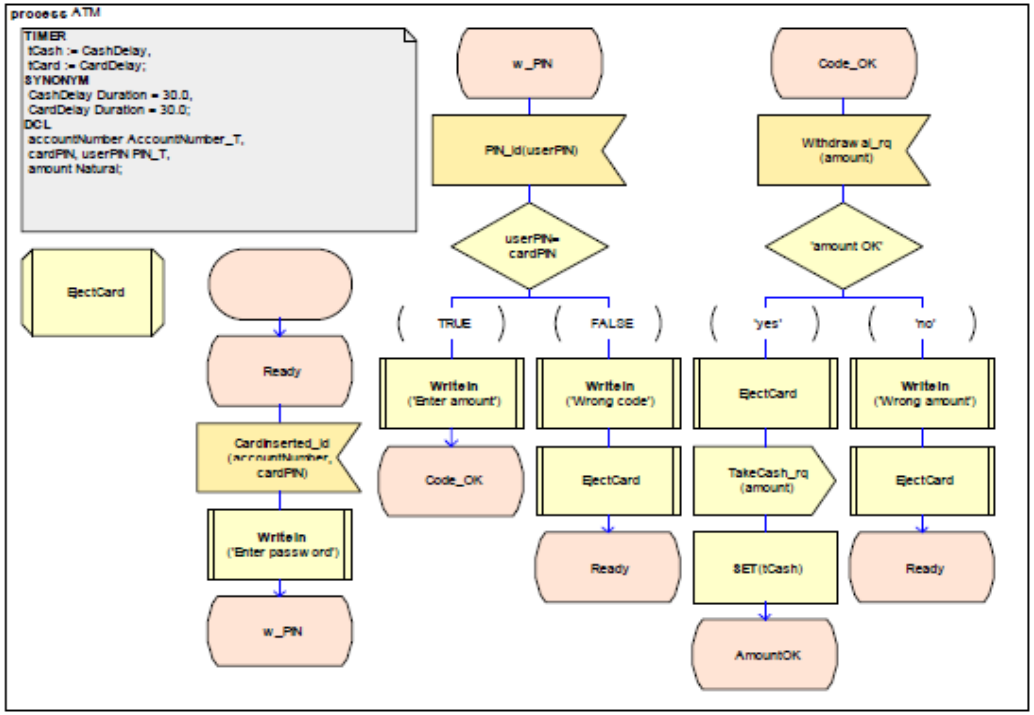
SDL Overview - Process



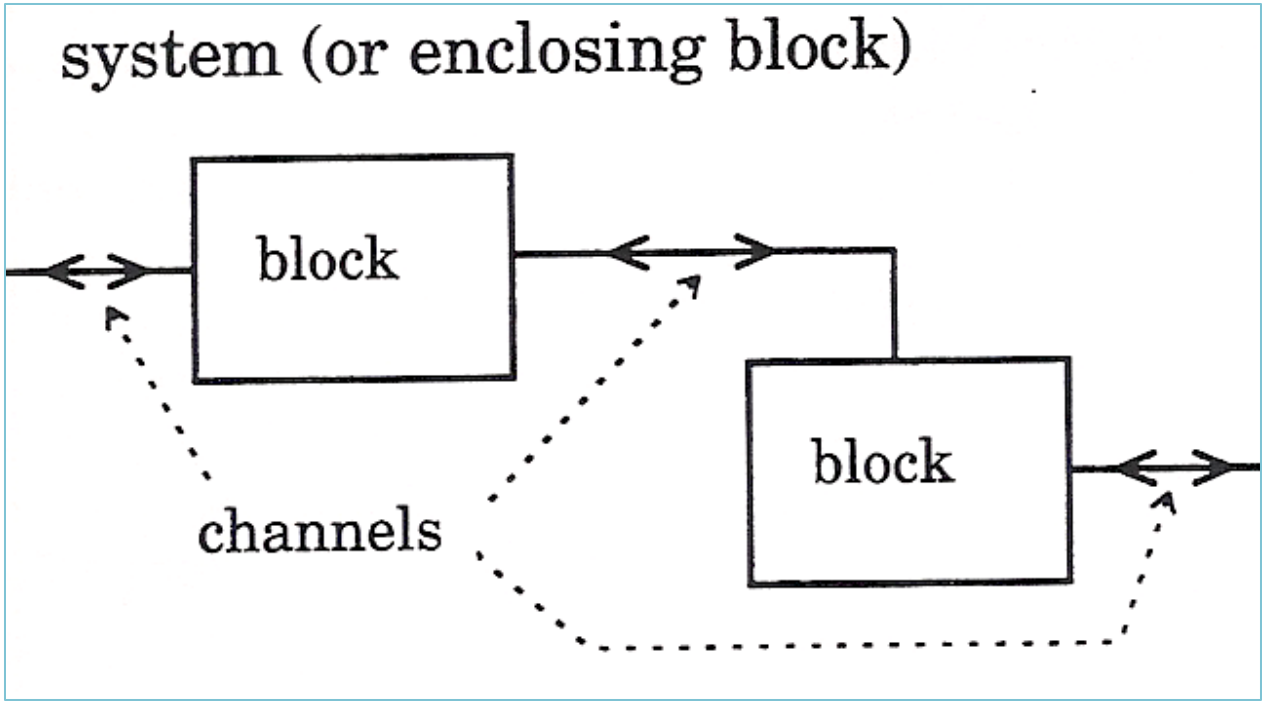
- A process is an agent that contains an extended finite state machine, and may contain other processes.

- A System is composed of a number of communicating process instances

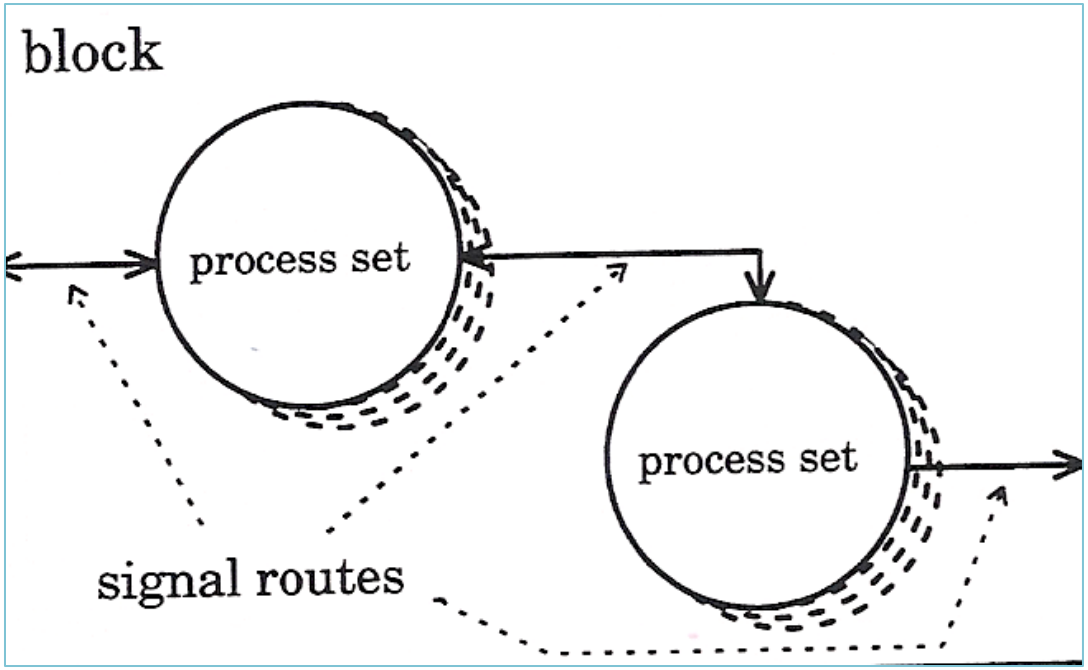
SDL Overview - Process Diagrams



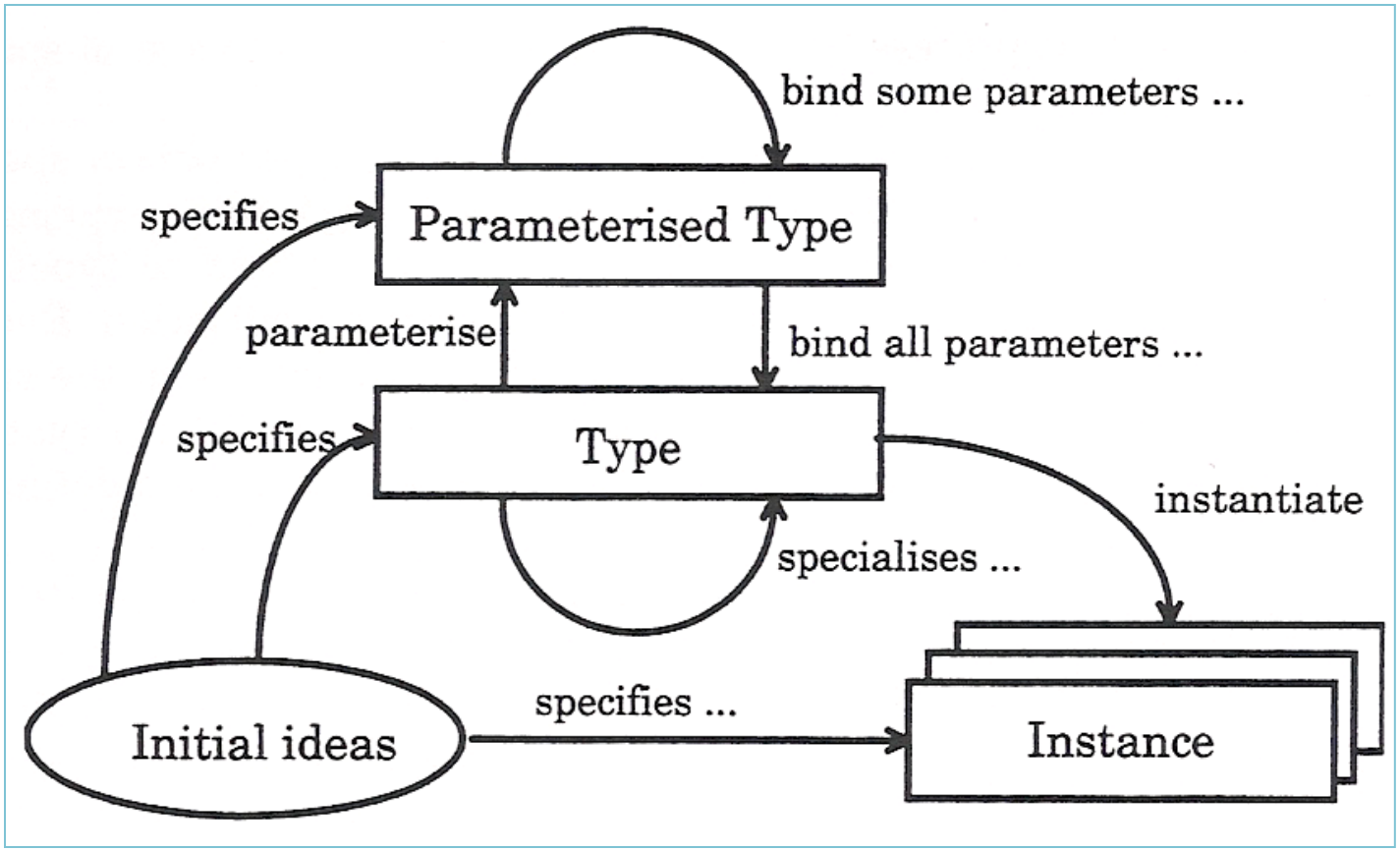
Χωρίζοντας ένα σύστημα σε blocks



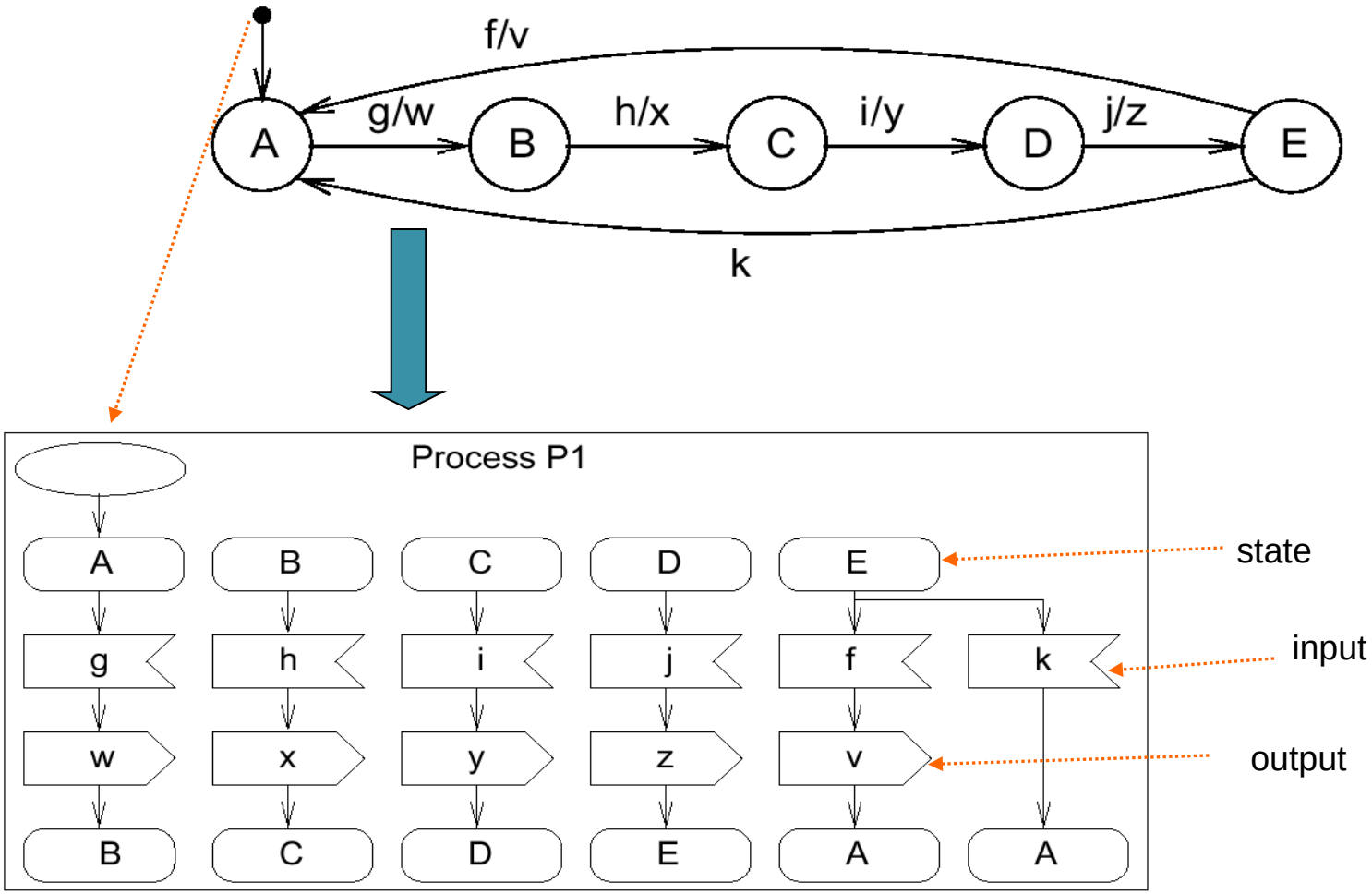
Χωρίζοντας blocks σύνολα διεργασιών



Η έννοια του τύπου στην SDL



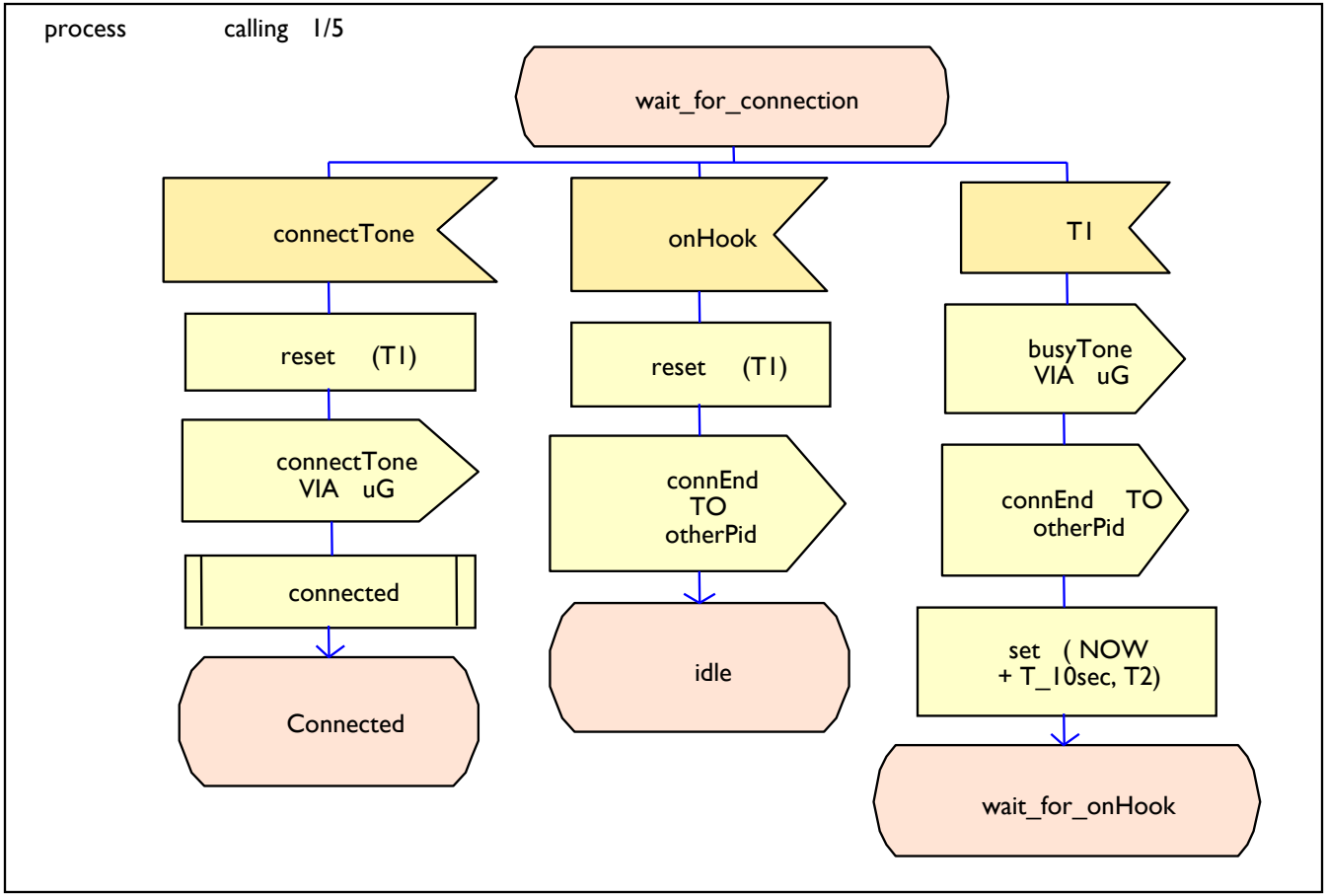
SDL representation of FSMs/processes



Dynamic Behavior

- A PROCESS exists in a state, waiting for an input (event).
- When an input occurs, the logic beneath the current state, and the current input executes.
- Any tasks in the path are executed.
- Any outputs listed are sent.
- The state machine will end up in either a new state, or return to the same state.
- The process then waits for next input (event)

Process Diagram Example

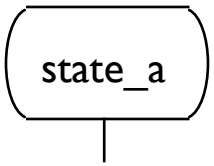


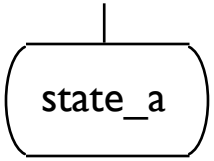
Process

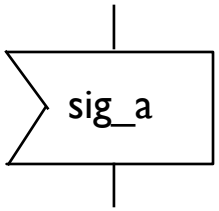
- PROCESS specifies dynamic behaviour
  - Process represents a communicating extended finite state machine.
  - each have a queue for input SIGNALs
  - may output SIGNALs
  - may be created with Formal PARameters and valid input SIGNALSET
  - it reacts to stimuli, represented in SDL by signal inputs.
  - stimulus normally triggers a series of actions such as data handling, signal sending, etc. A sequence of actions is described in a transition.

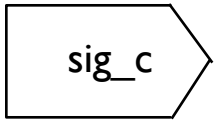
- PROCESS diagram is a Finite State Machine (FSM) description

Process







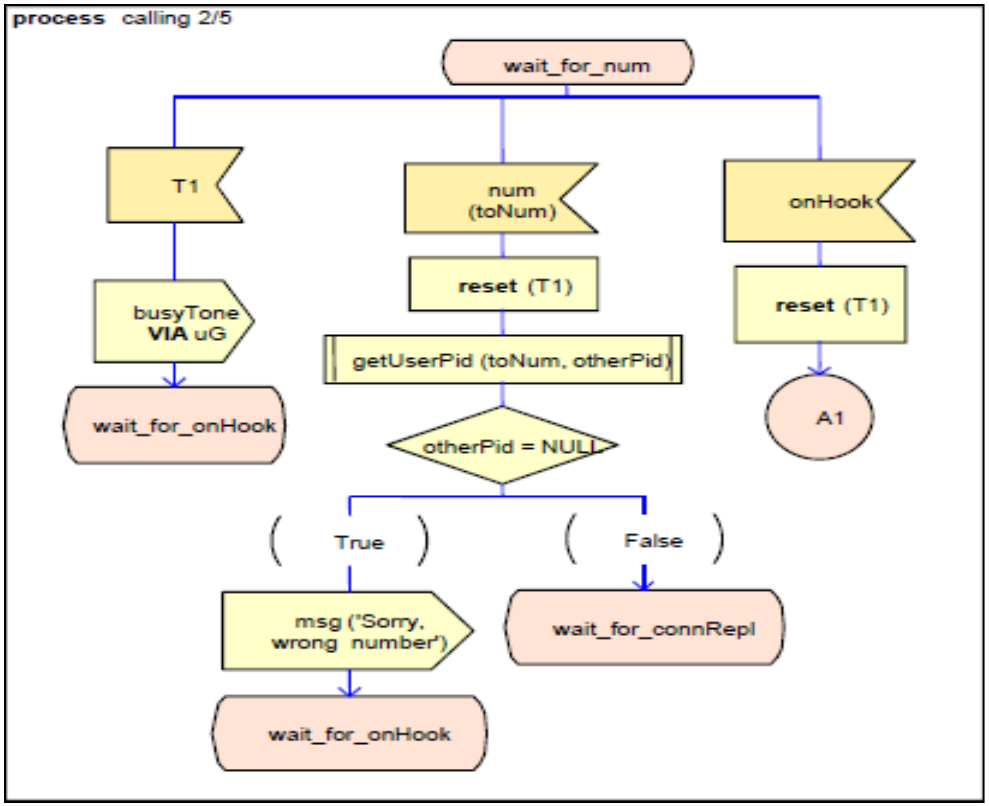


- STATEs: point in PROCESS where input queue is being monitored for arrived SIGNALs
  - subsequent state transition may or may not have a NEXTSTATE

- INPUT: indicates that the subsequent state transition should be executed if the SIGNAL matching the INPUT arrives
  - INPUTs may specify SIGNALs and values within those SIGNALs
  - Inputs can also specify timer expiry

- OUTPUT: specifies the sending of a SIGNAL to another PROCESS

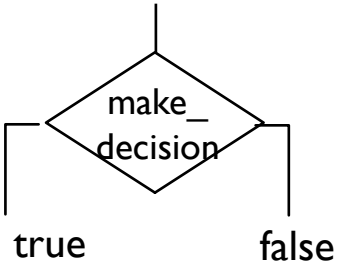
Process Example



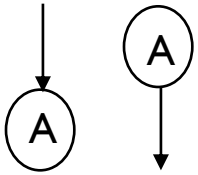
Process Diagram Components

- TASK: description of operations on variables or special operations

- The text within the TASK body can contain assign statements.

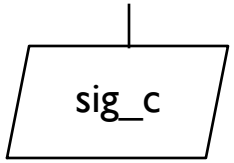


- DECISION: tests a condition to determine subsequent PROCESS flow

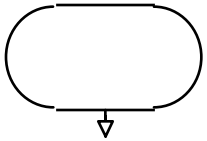


- JOIN: equivalent to GOTO.

Process Diagram Components



- SAVE: specifies that the consumption of a SIGNAL be delayed until subsequent SIGNALs have been consumed
  - the effect is that the SAVEd SIGNAL is not consumed until the next STATE
  - no transition follows a SAVE
  - the SAVEd SIGNAL is put at the end of the queue and is processed after other SIGNALs arrive



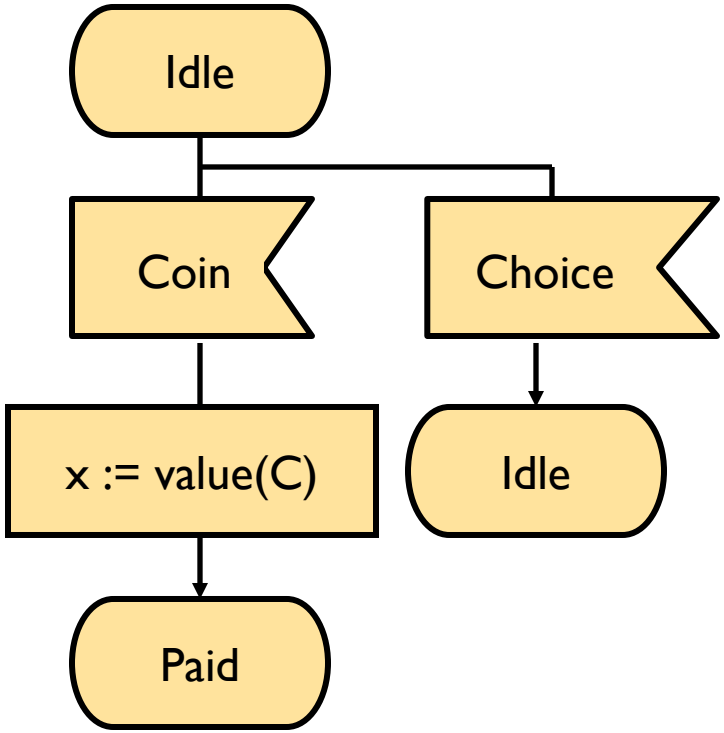
- START: used to describe behaviour on creation as well as indicating initial state
  - Similar shape to state only with semi-circular sides

SDL processes

Textual form

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

Graphical form

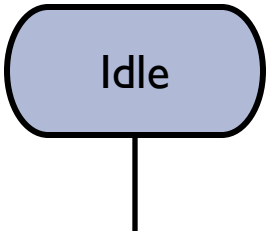


SDL process states

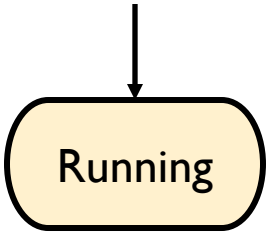
- At a particular state,
- A signal is removed from the queue
- If a transition defined for the signal in current state,
  - Run the transition
  - Transmit signals
  - Update internal variables
  - Choose a next state
- If no transition defined for the signal in current state,
  - Discard the signal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

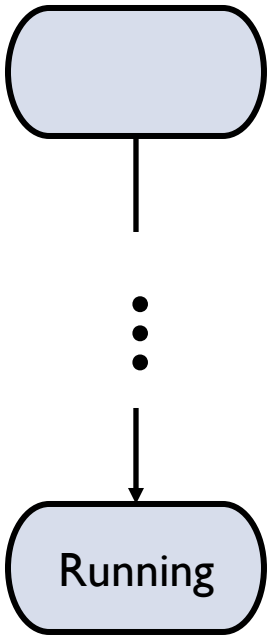


- Arrow entering means a next state



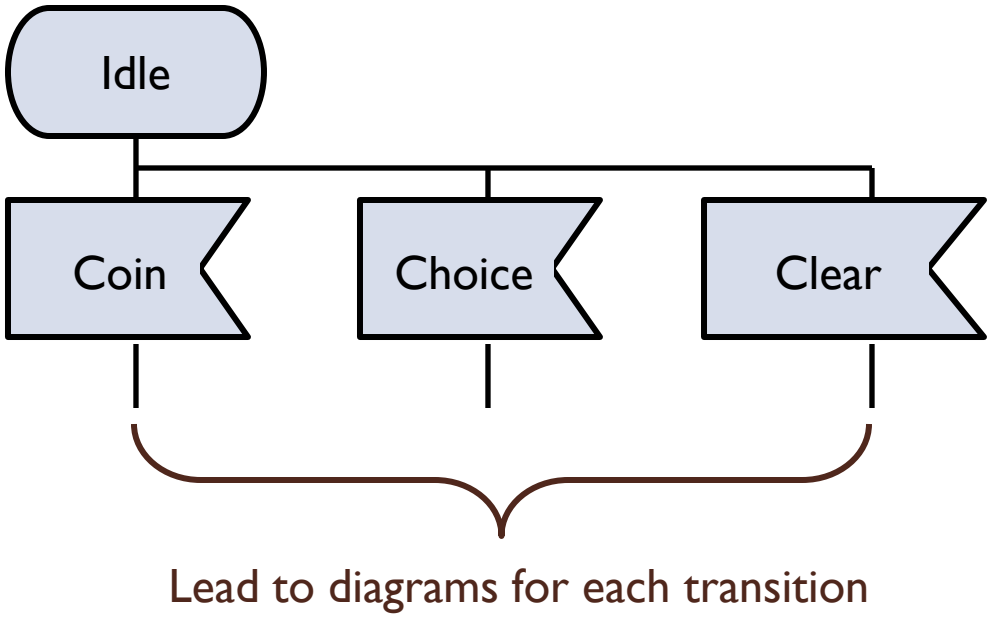
The start symbol

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



The receive symbol

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



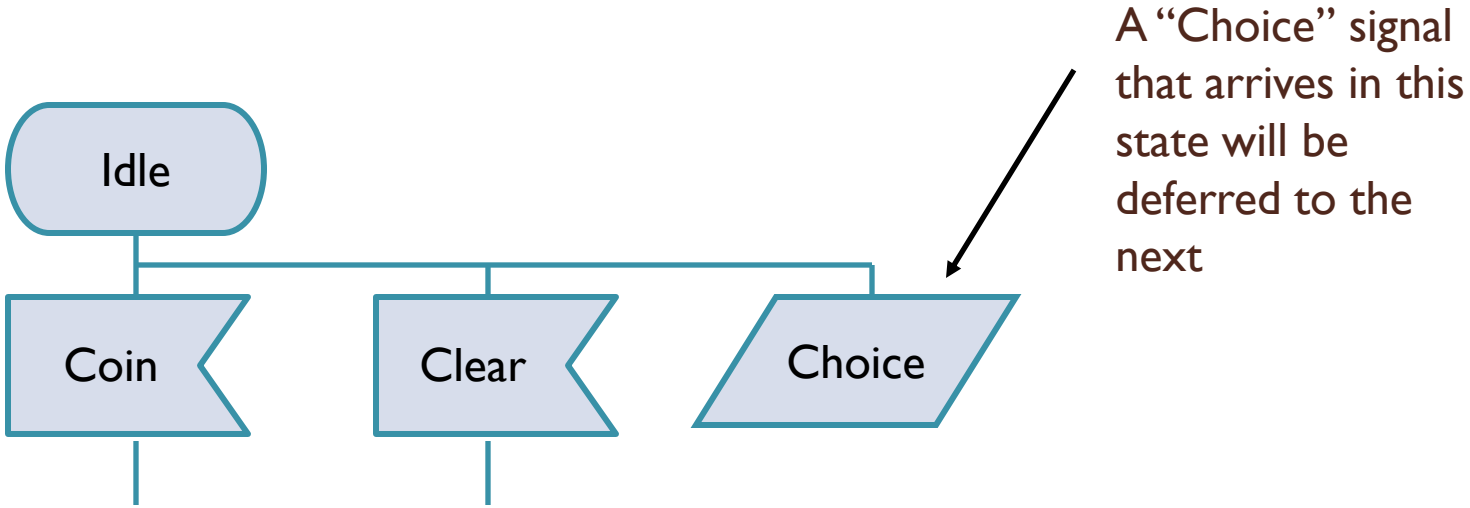
Received signals

- 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

- Designed for handling signals that arrive out of order

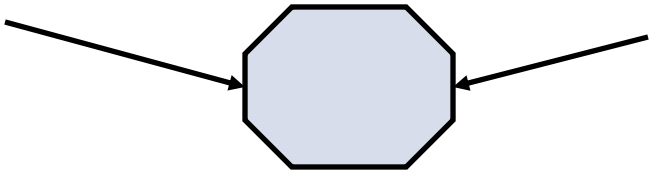


- Like receive, but instead pushes the signal back in the queue

The SAVE symbol

- Single process input queue totally orders the sequence of events that arrive at a process

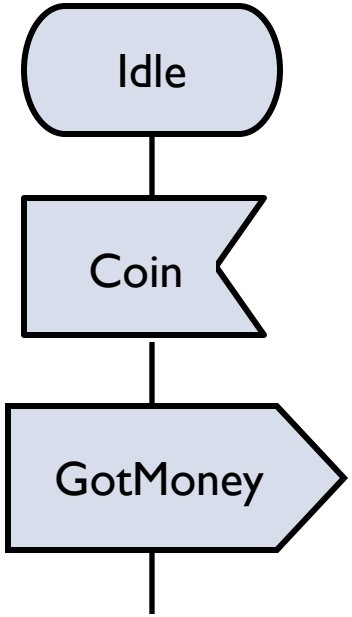
- What if two events arrive from different processes at more-or-less the same time?



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

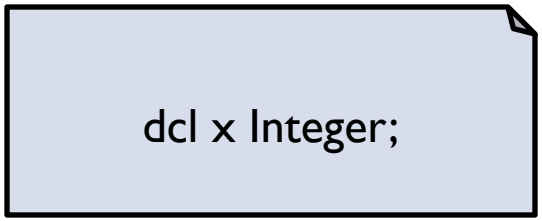
The Output symbol

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



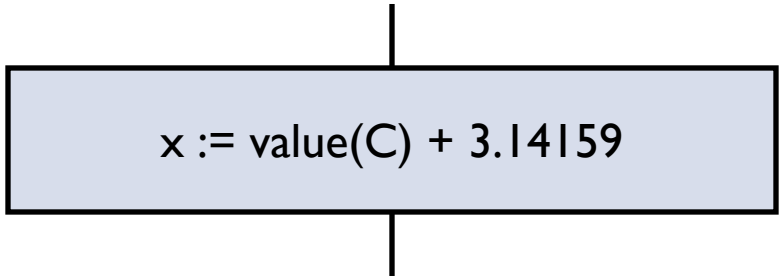
Vocal variables

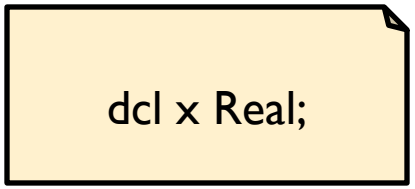
- An SDL process has local variables it can manipulate
- Partially shared variables
  - Only the owning process may write a variable
  - Other processes may be allowed to read a variable
- Variables are declared in a text annotation



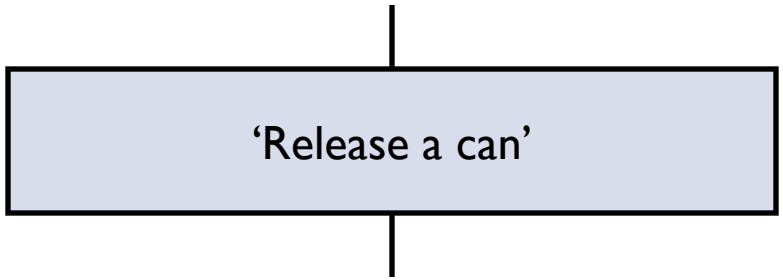
Task Symbol

- Assignment of variable to value of expression



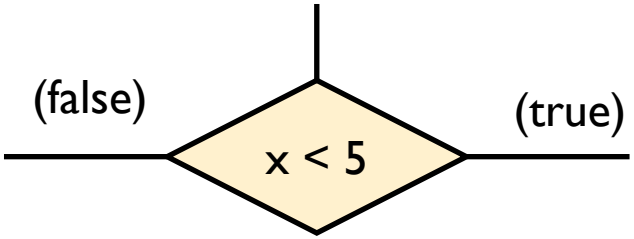


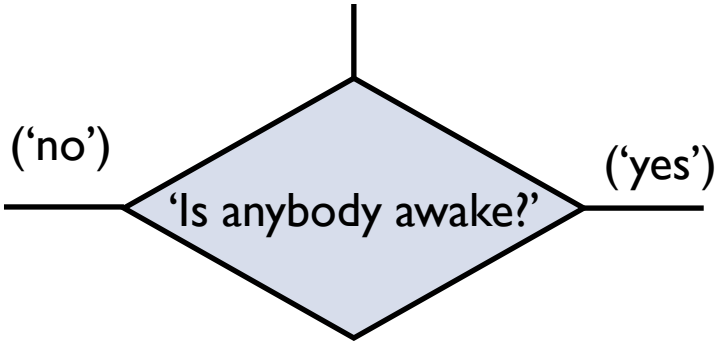
- Informal text
  - Produces an incomplete specification
  - Intended to be later refined



The Decision Symbol

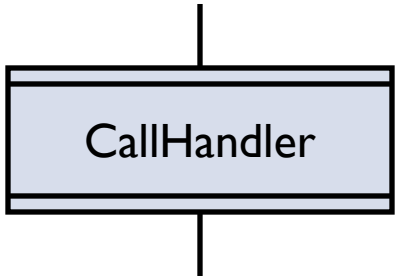
- A two-way branch that can check a condition
- Can be an expression or informal



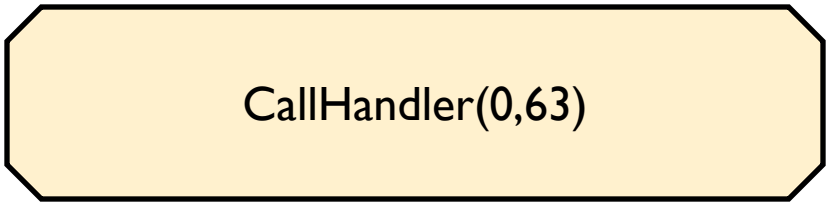


Process Creation Symbol

- 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

CallHandler(1,1)

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

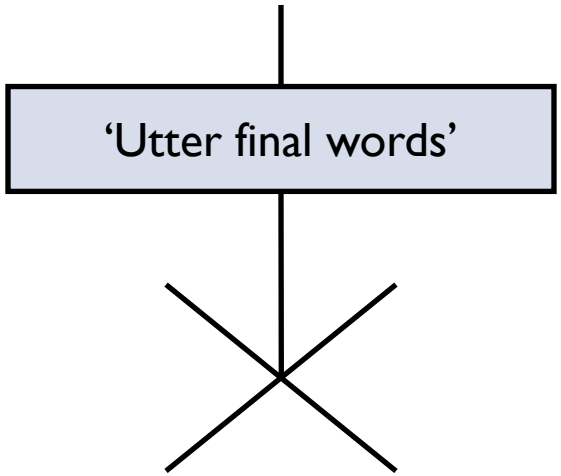
CallHandler(0,1)

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

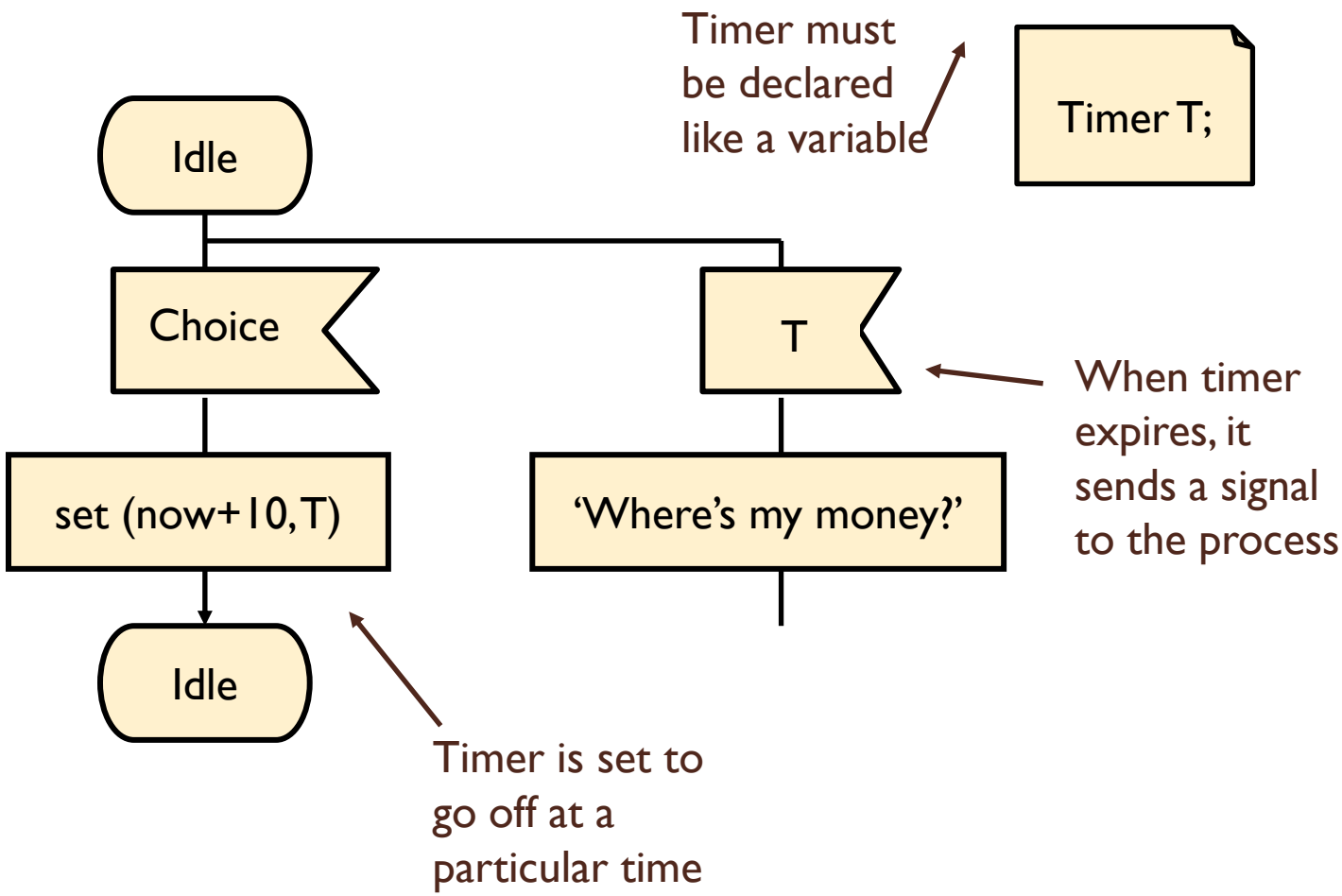
CallHandler(0,64)

Process Termination

- A process can only terminate itself



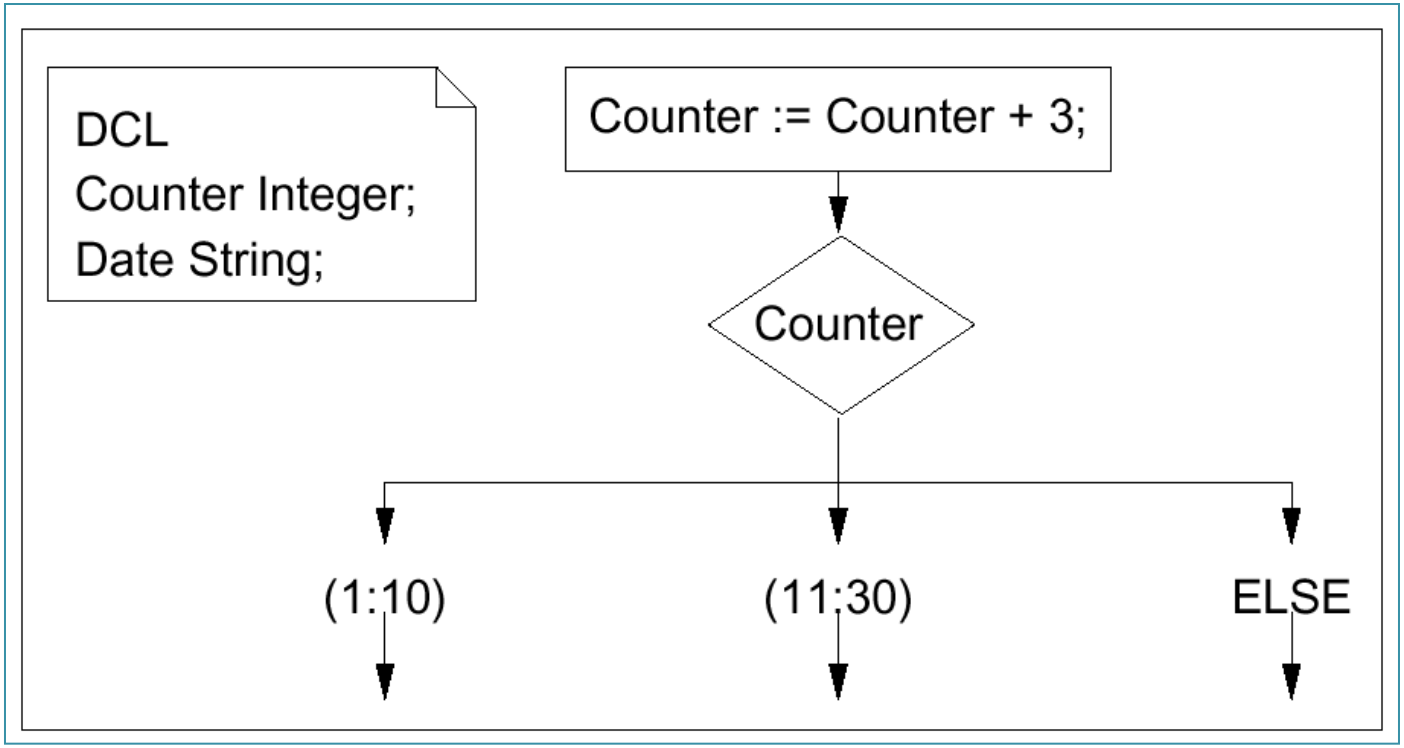
Timers


Timer must be declared like a variable
Timer T;
Idle
Choice
T
When timer expires, it sends a signal to the process
set (now+10,T)
'Where's my money?'
Idle
Timer is set to go off at a particular time

Operations on Data

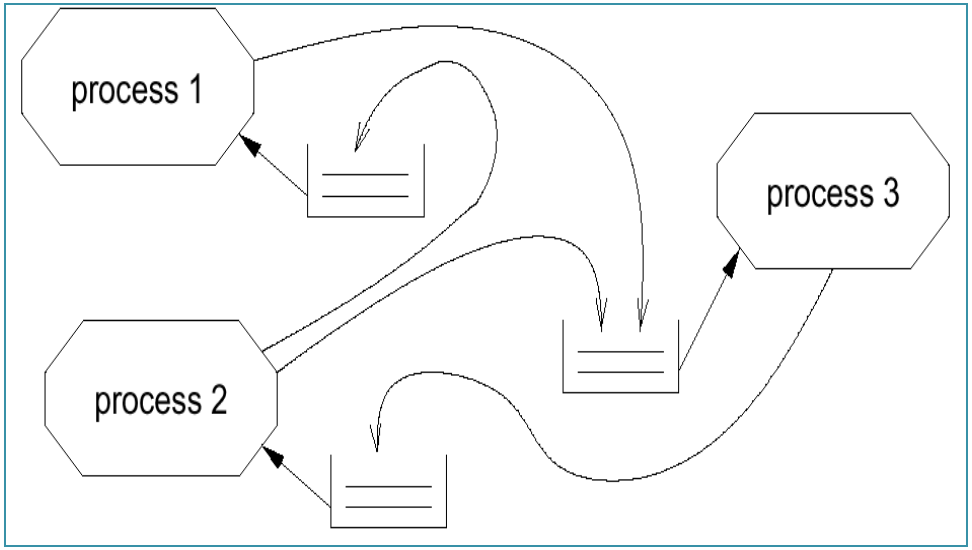
- Variables can be declared locally for processes.
- Their type can be predefined or defined in SDL itself.
- SDL supports abstract data types (ADTs).

Examples:



Επικοινωνία μεταξύ SDL-FSMs

- Communication between FSMs (or „processes“) is based on message-passing, assuming a potentially indefinitely large FIFO-queue.



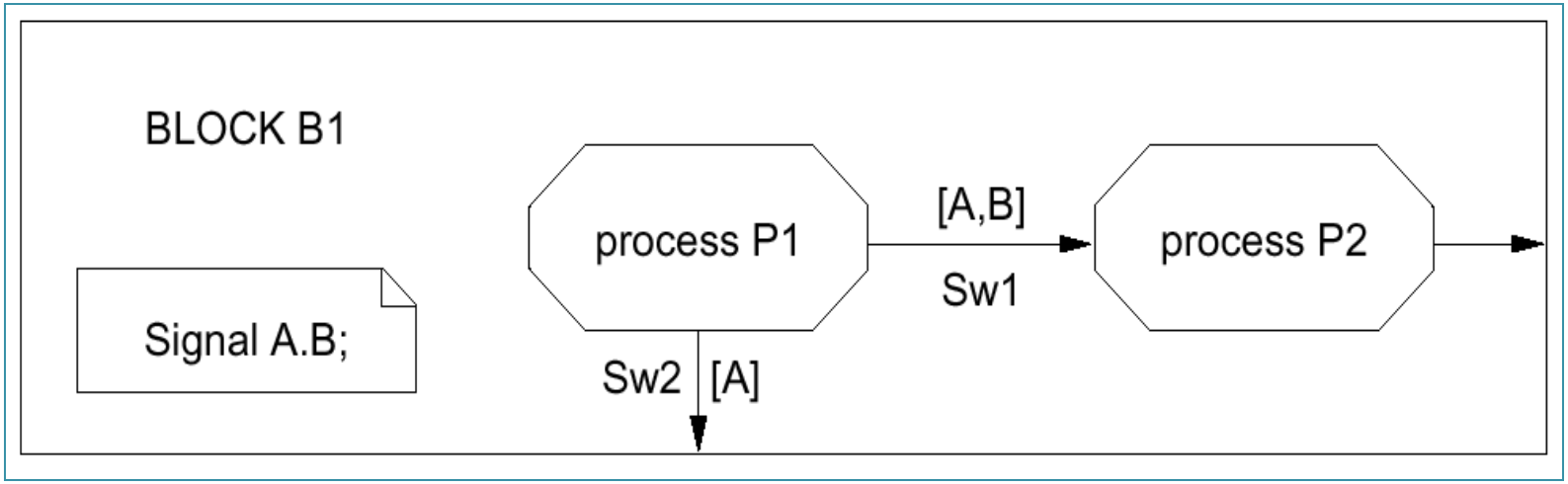
- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).

Επικοινωνία μεταξύ SDL-FSMs

- Interaction between processes can be described in process interaction diagrams (special case of block diagrams).

- In addition to processes, these diagrams contain channels and declarations of local signals.

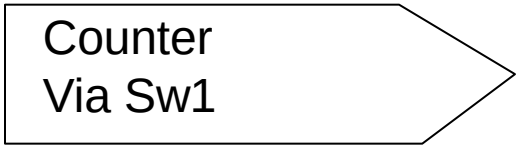
Example:



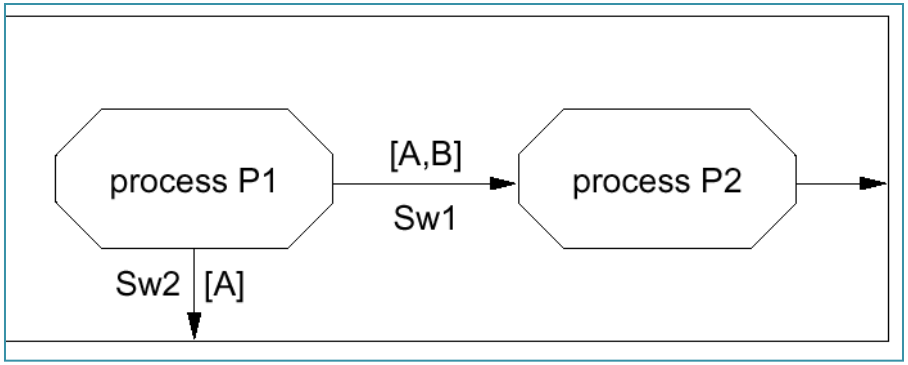
Designation of recipients

1. **Through process identifiers:**
   Example: OFFSPRING represents identifiers of processes generated dynamically.

2. **Explicitly:**
   By including the channel name.

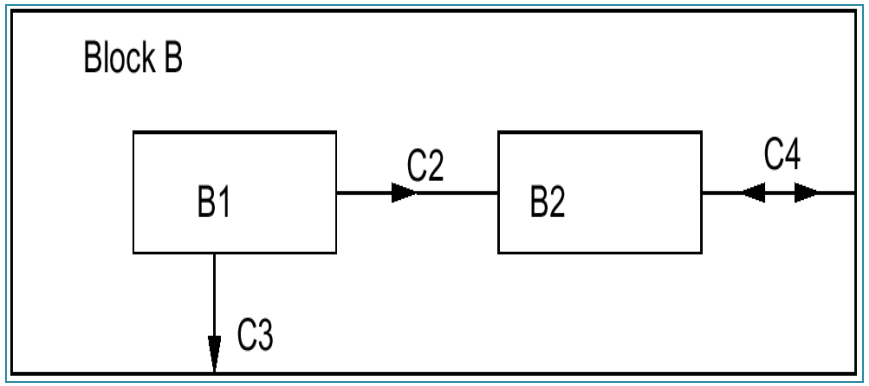


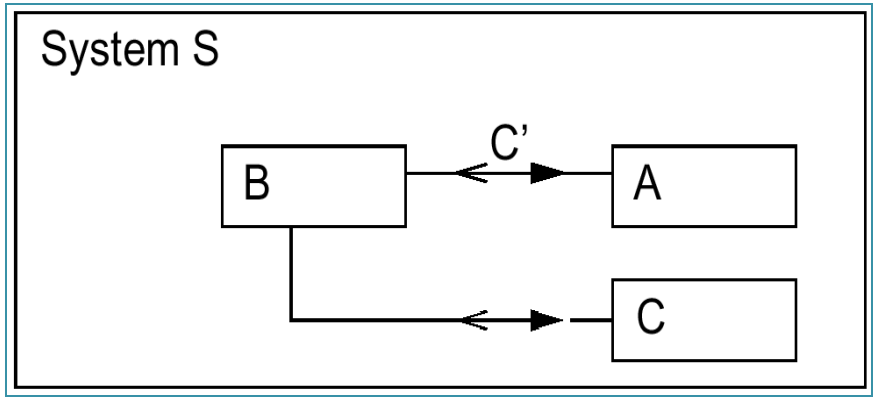
3. **Implicitly:**
   If signal names imply channel names (B → Sw1)

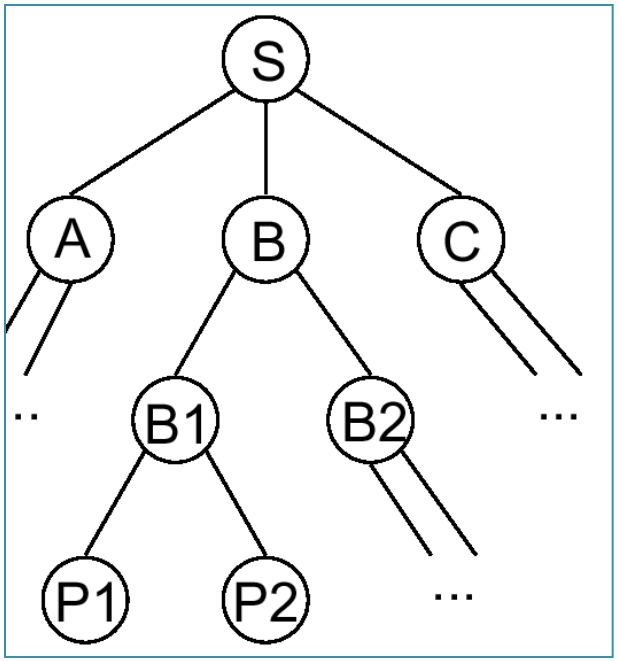


Ιεραρχία SDL

Process interaction diagrams can be included in **blocks.** The root block is called **system.**





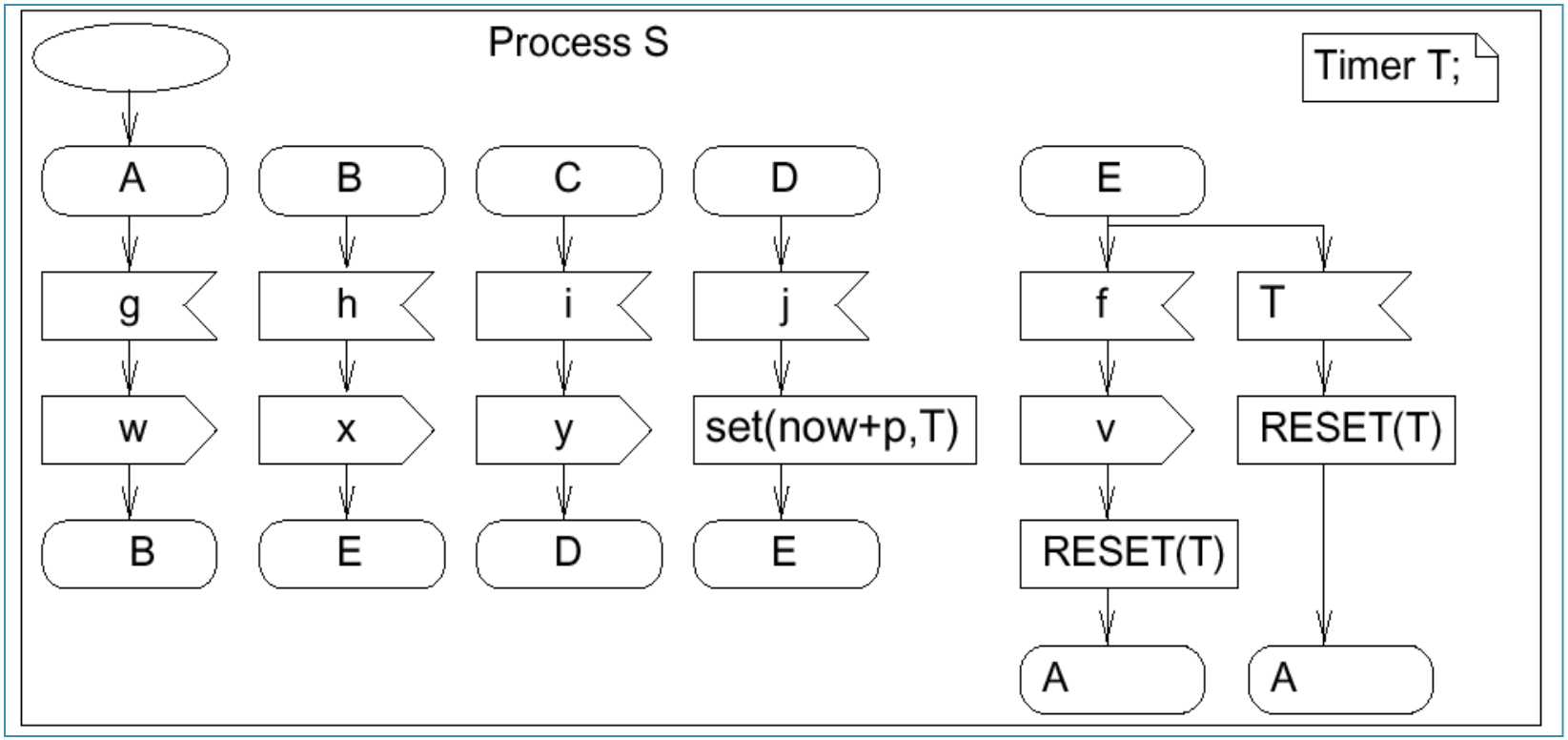


Processes cannot contain other processes, unlike in StateCharts.

Timers

- Timers can be declared locally. Elapsed timers put signal into queue (not necessarily processed immediately).

- RESET also removes timer signal from queue.

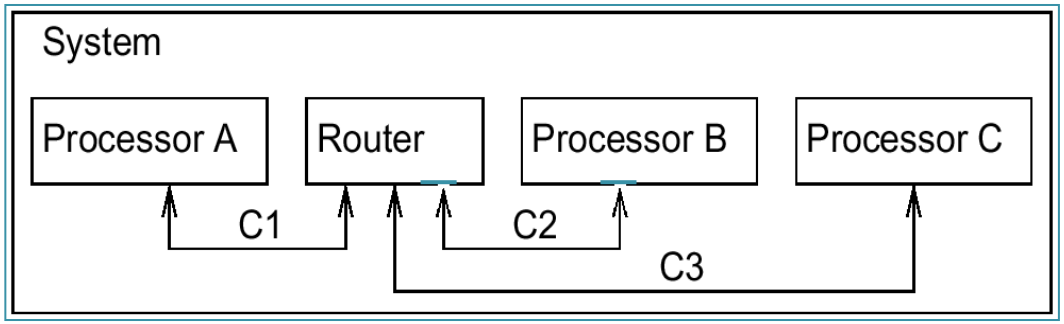


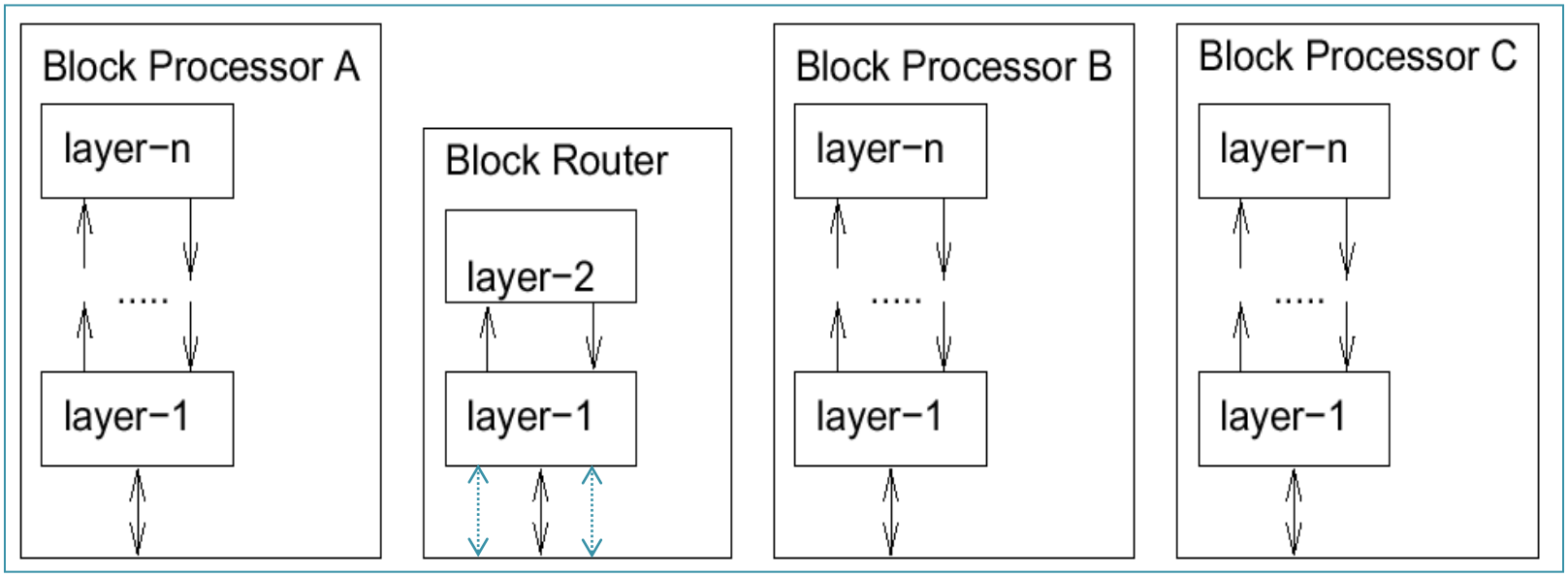
Additional language elements

- SDL includes a number of additional language elements, like

  - procedures
  - creation and termination of processes
  - advanced description of data

Application: description of network protocols





Larger example: vending

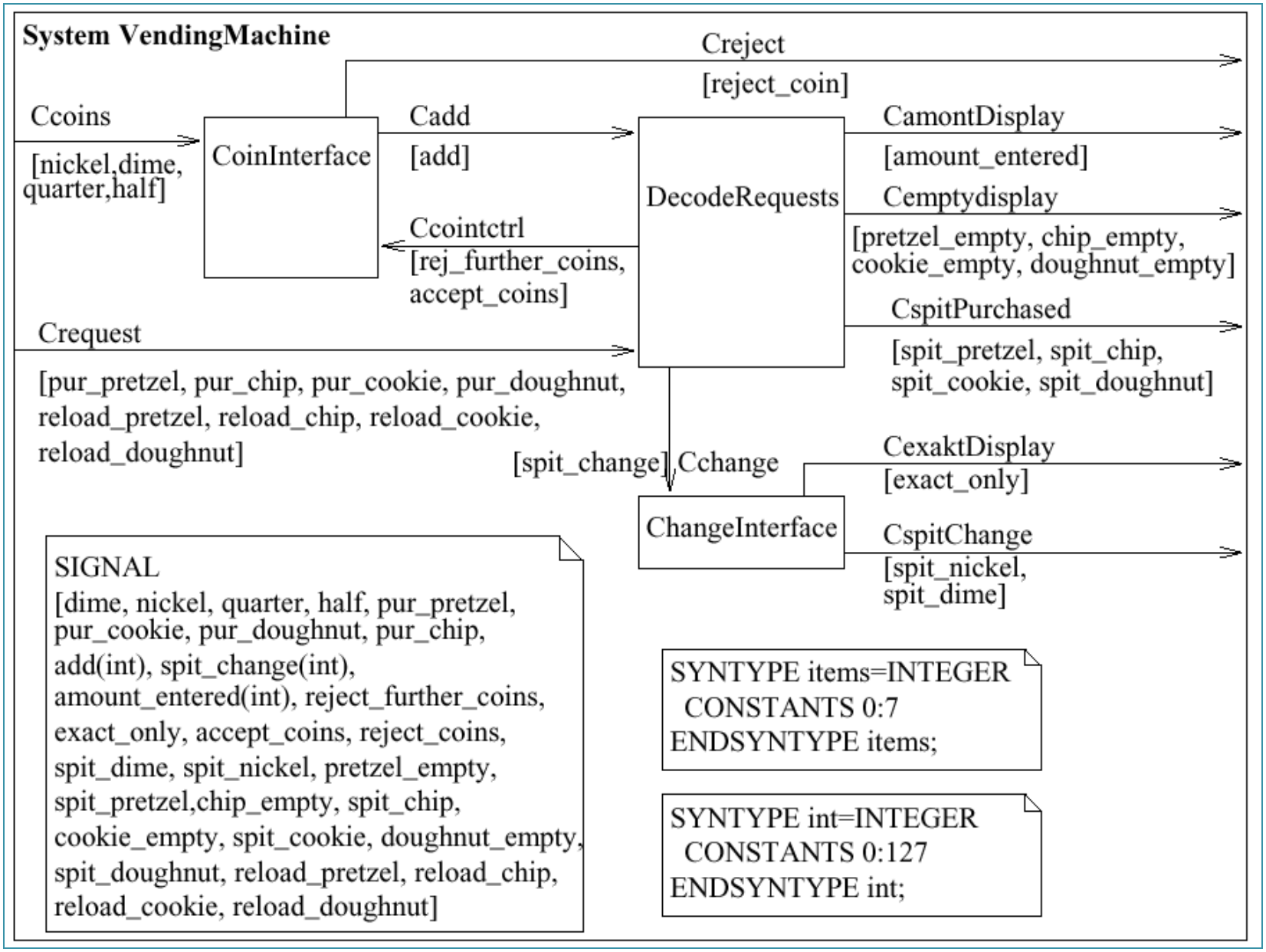
Machine° selling pretzels, (potato) chips, cookies, and doughnuts:

accepts nickels, dime, quarters, and half-dollar coins.

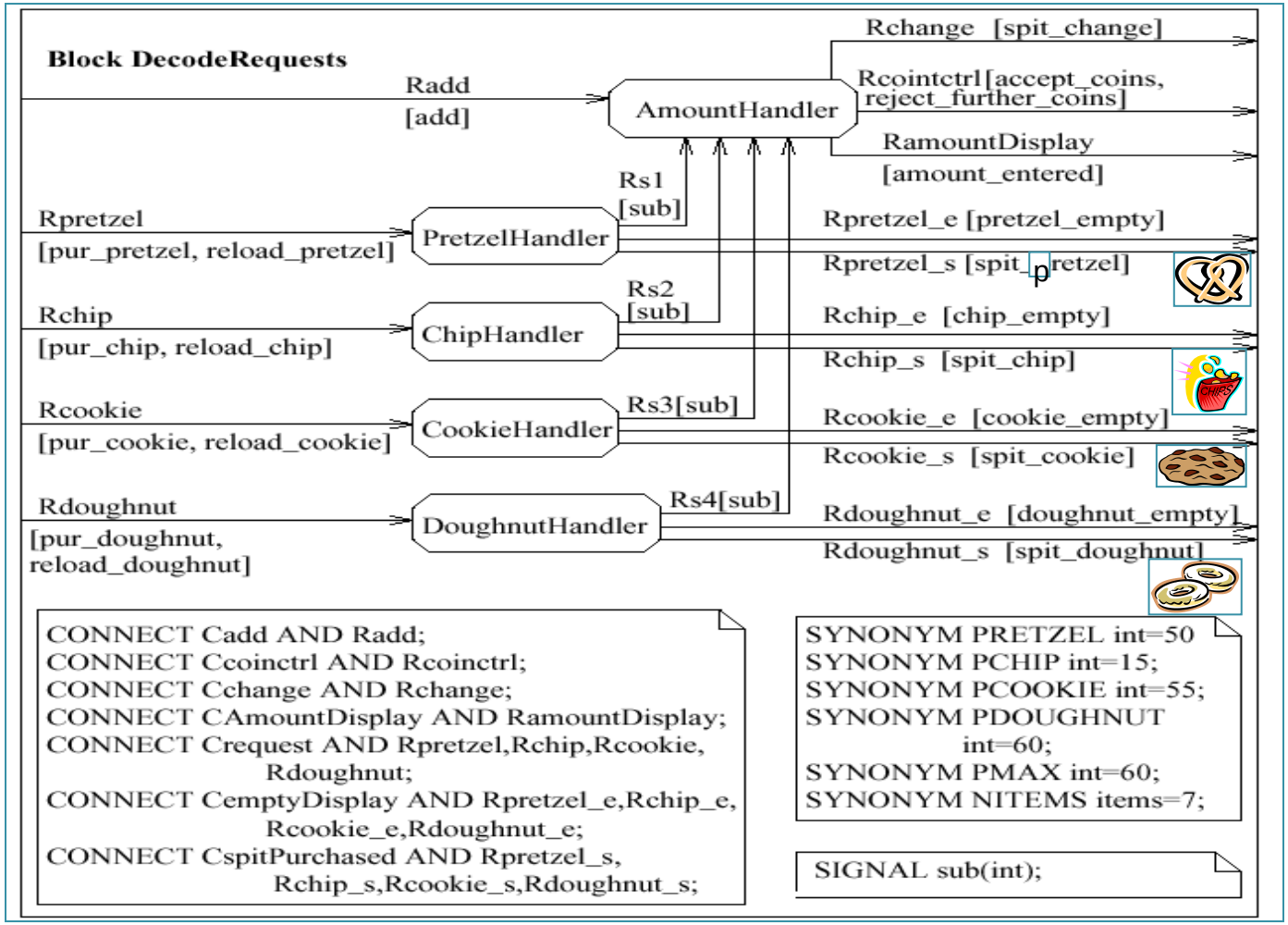
Not a distributed application.

° [J.M. Bergé, O. Levia, J. Roullard: High-Level System Modeling, Kluwer Academic Publishers, 1995]

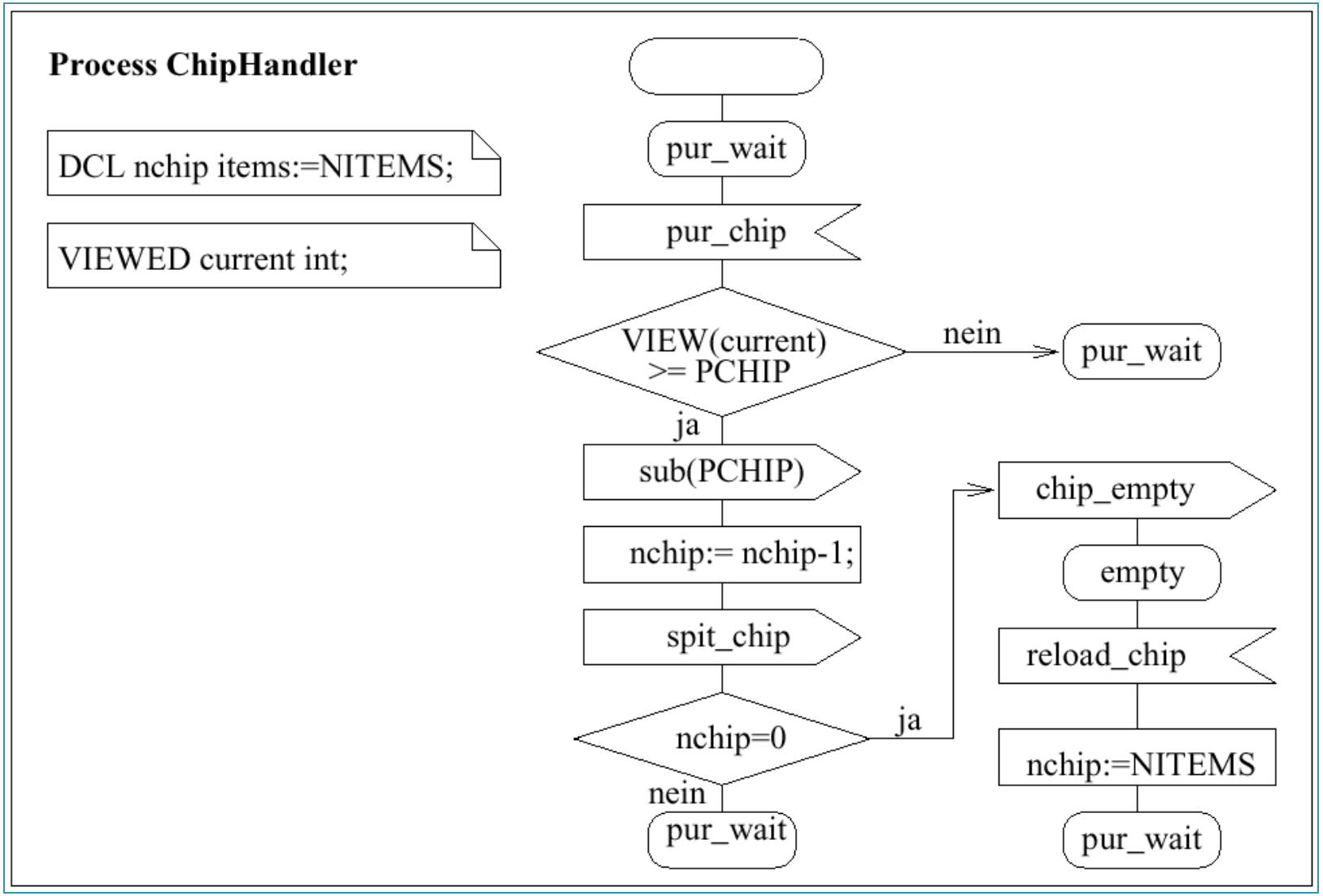
Larger example: vending



Decode Requests



Process ChipHandler



Δυναμική δημιουργία διεργασιών

- Συνήθως, τα συστήματα δομούνται σχεδιάζοντας κάποιες διεργασίες αυθύπαρκτες, οι οποίες αναλαμβάνουν τη δημιουργία κάποιων άλλων.

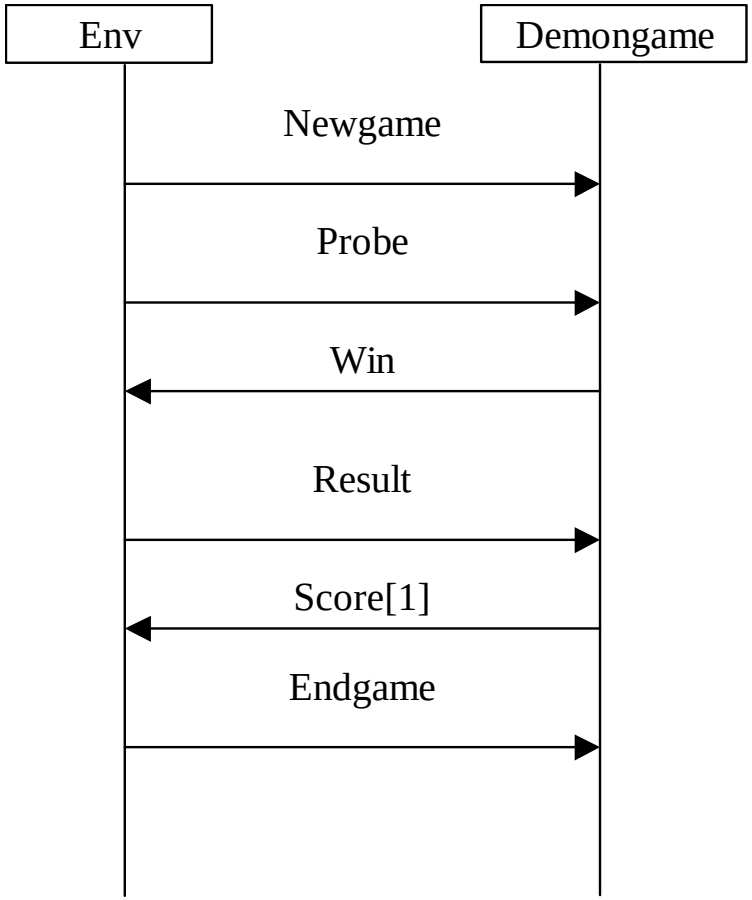
- Η μεν διεργασία, που είναι αυθύπαρκτη και δημιουργεί μία άλλη, καλείται γονέας (parent process), ή δε δημιουργούμενη ονομάζεται διεργασία – παιδί (child process).

- Τα στιγμιότυπα κάθε διεργασίας σέβονται τη δομή και τη συμπεριφορά της διεργασίας αυτής.

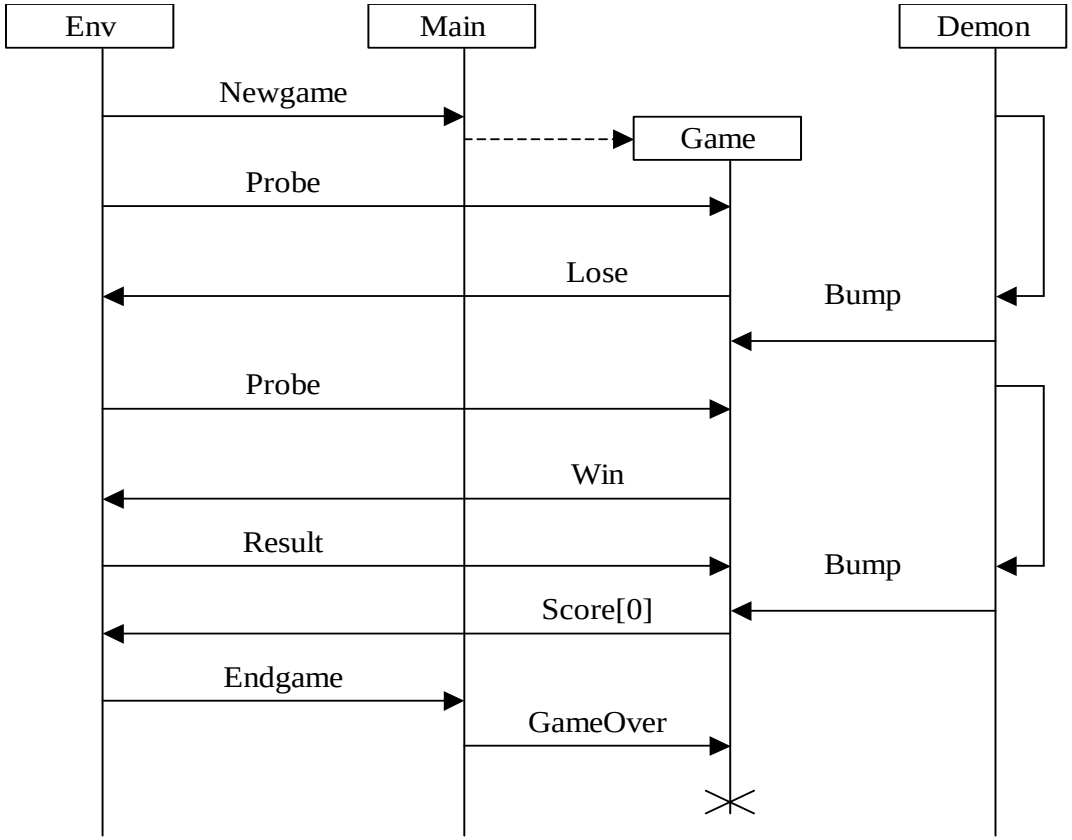
- Είναι αυτόνομα, ωστόσο επικοινωνούν μεταξύ τους, αλλά και με άλλες διεργασίες.

- Στην περίπτωση όπου υπάρχουν περισσότερα του ενός στιγμιότυπα, η αποστολή σημάτων προς ένα συγκεκριμένο στιγμιότυπο γίνεται με ρητή διευθυνσιοδότηση (explicit addressing), δηλαδή με τη χρήση της ταυτότητας του (process identity).

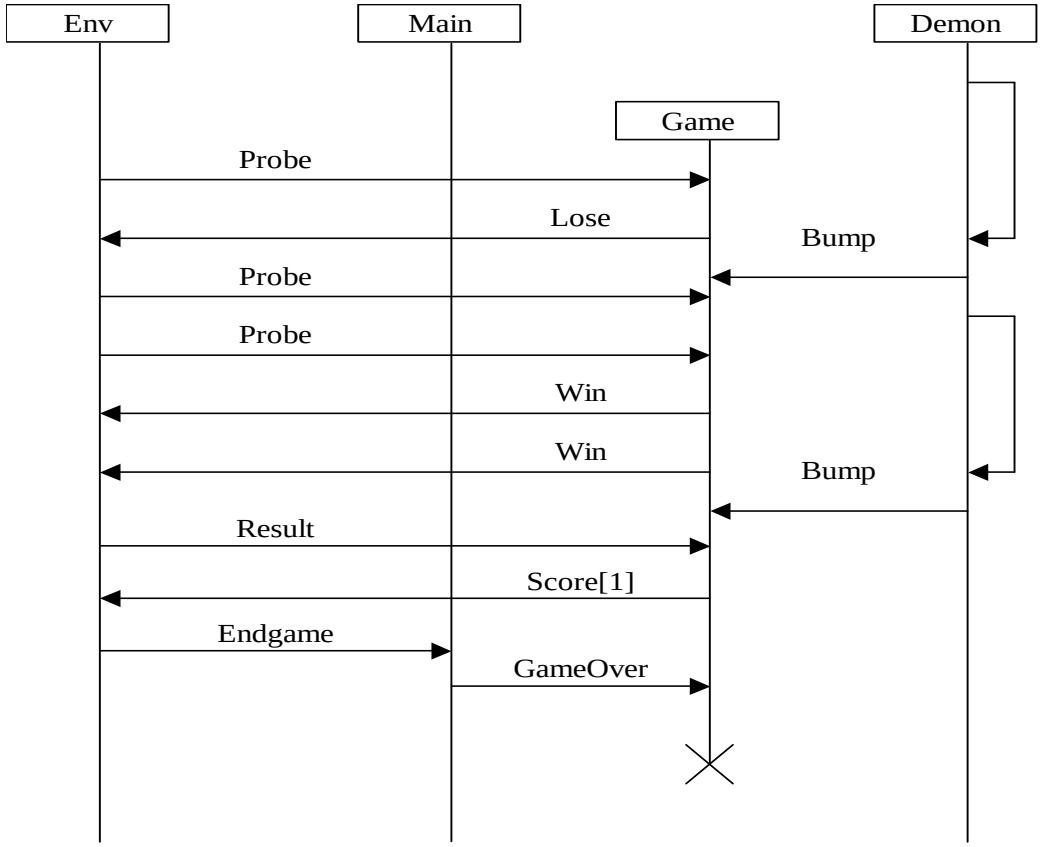
Παράδειγμα



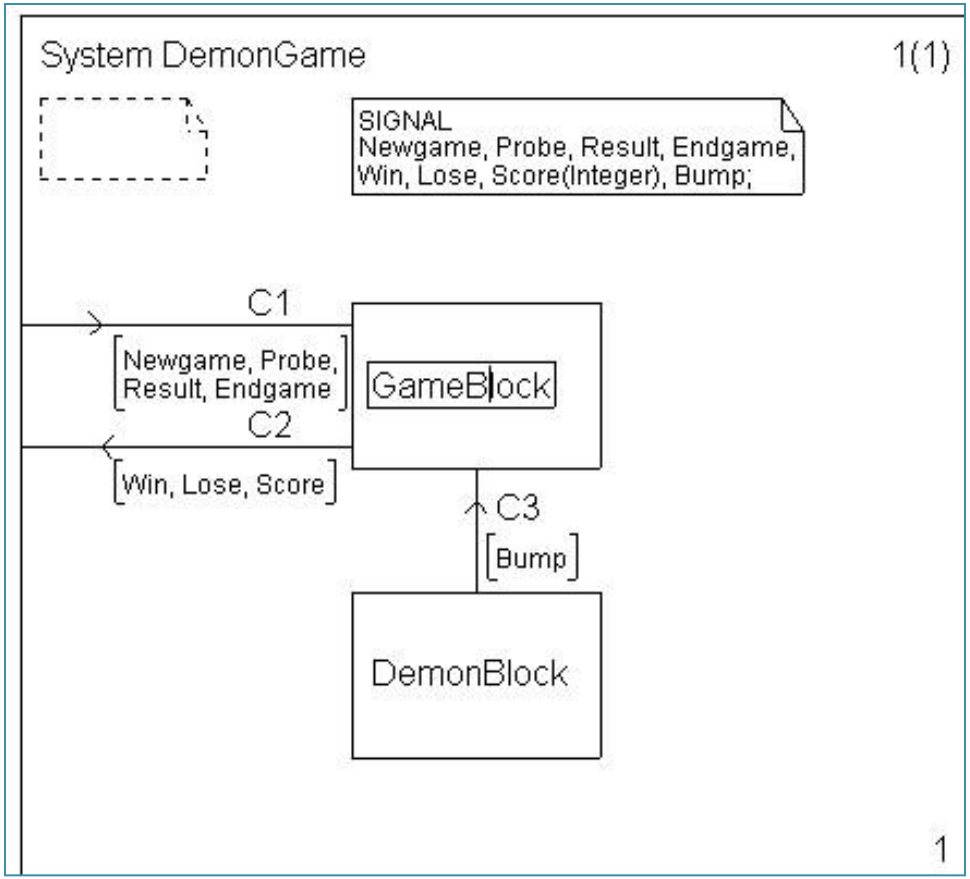
Παράδειγμα


Env
Main
Demon
Newgame
Game
Probe
Lose
Bump
Probe
Win
Result
Bump
Score[0]
Endgame
GameOver

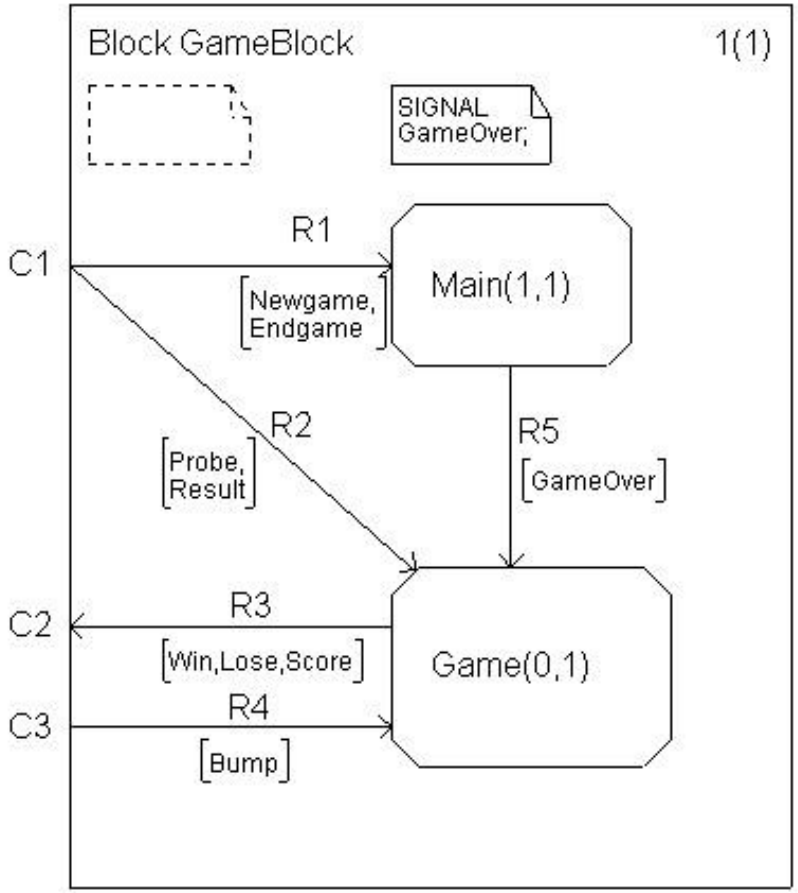
Παράδειγμα


Env
Main
Demon
Game
Probe
Lose
Bump
Probe
Probe
Win
Win
Bump
Result
Score[1]
Endgame
GameOver

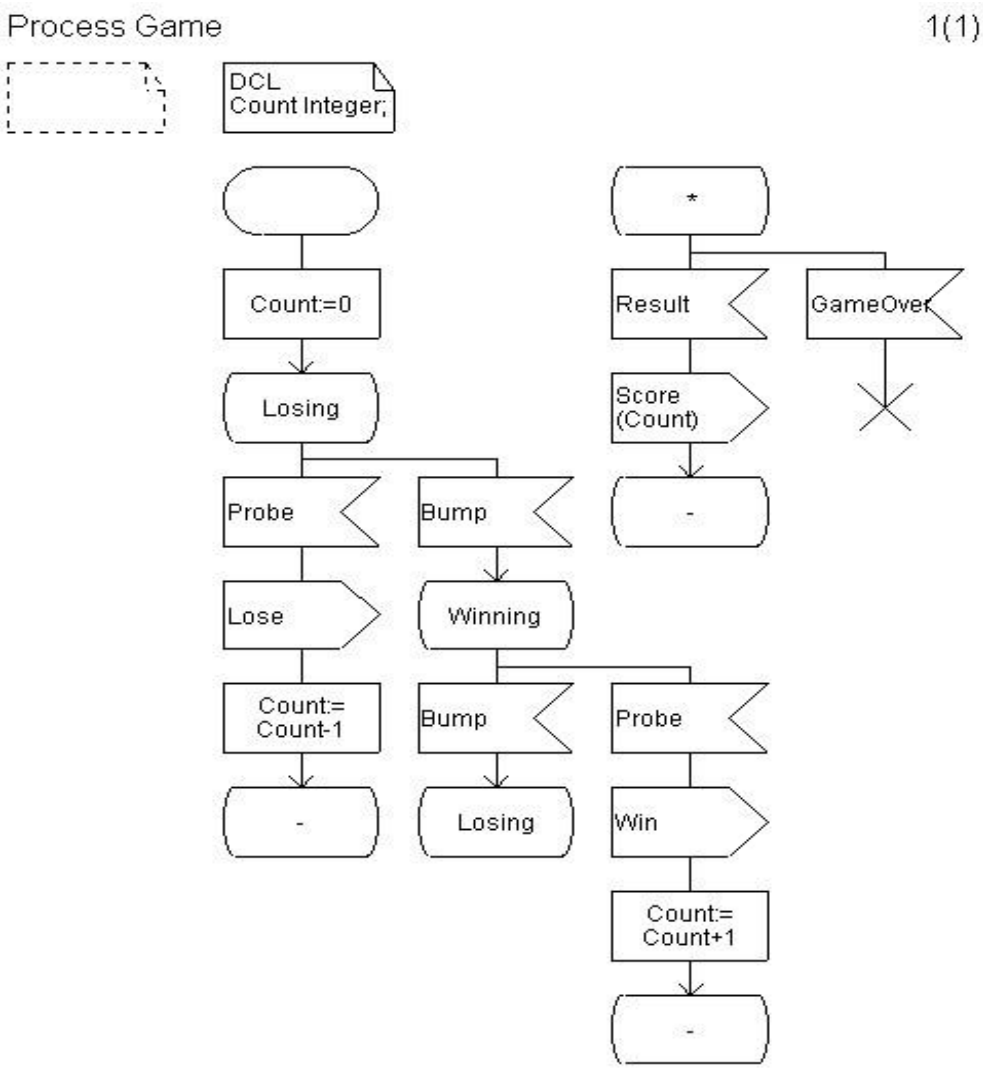
Παράδειγμα


System DemonGame
1(1)
SIGNAL
Newgame, Probe, Result, Endgame,
Win, Lose, Score(Integer), Bump;
C1
Newgame, Probe,
Result, Endgame
GameBlock
C2
Win, Lose, Score
C3
Bump
DemonBlock
1

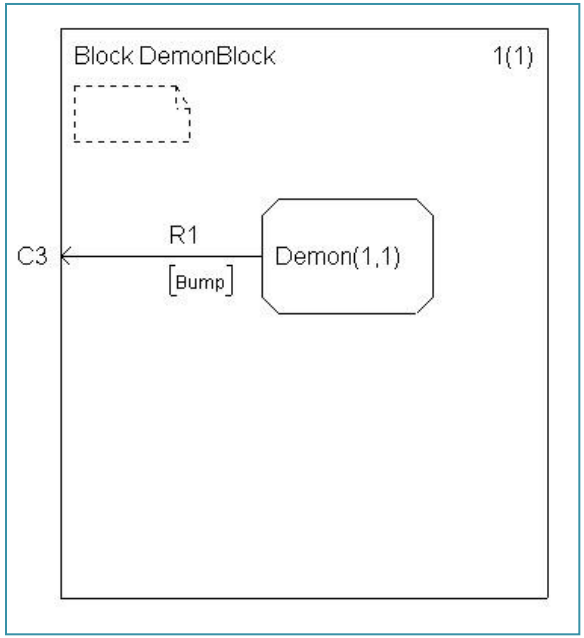
Παράδειγμα

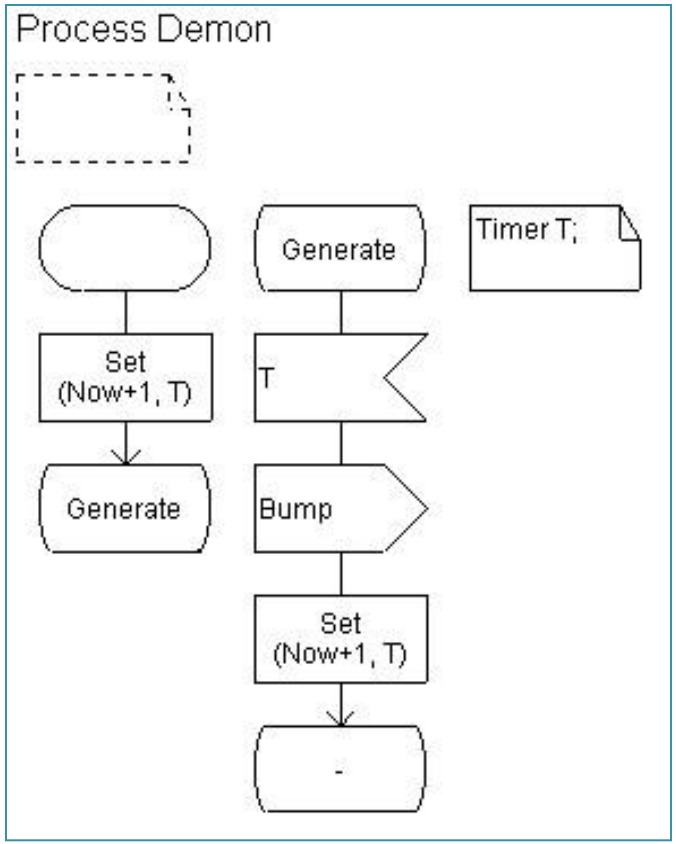

Block GameBlock
1(1)
SIGNAL
GameOver;
R1
C1
Newgame,
Endgame
Main(1,1)
R2
Probe,
Result
R5
GameOver
C2
R3
Win,Lose,Score
Game(0,1)
C3
R4
Bump

Παράδειγμα

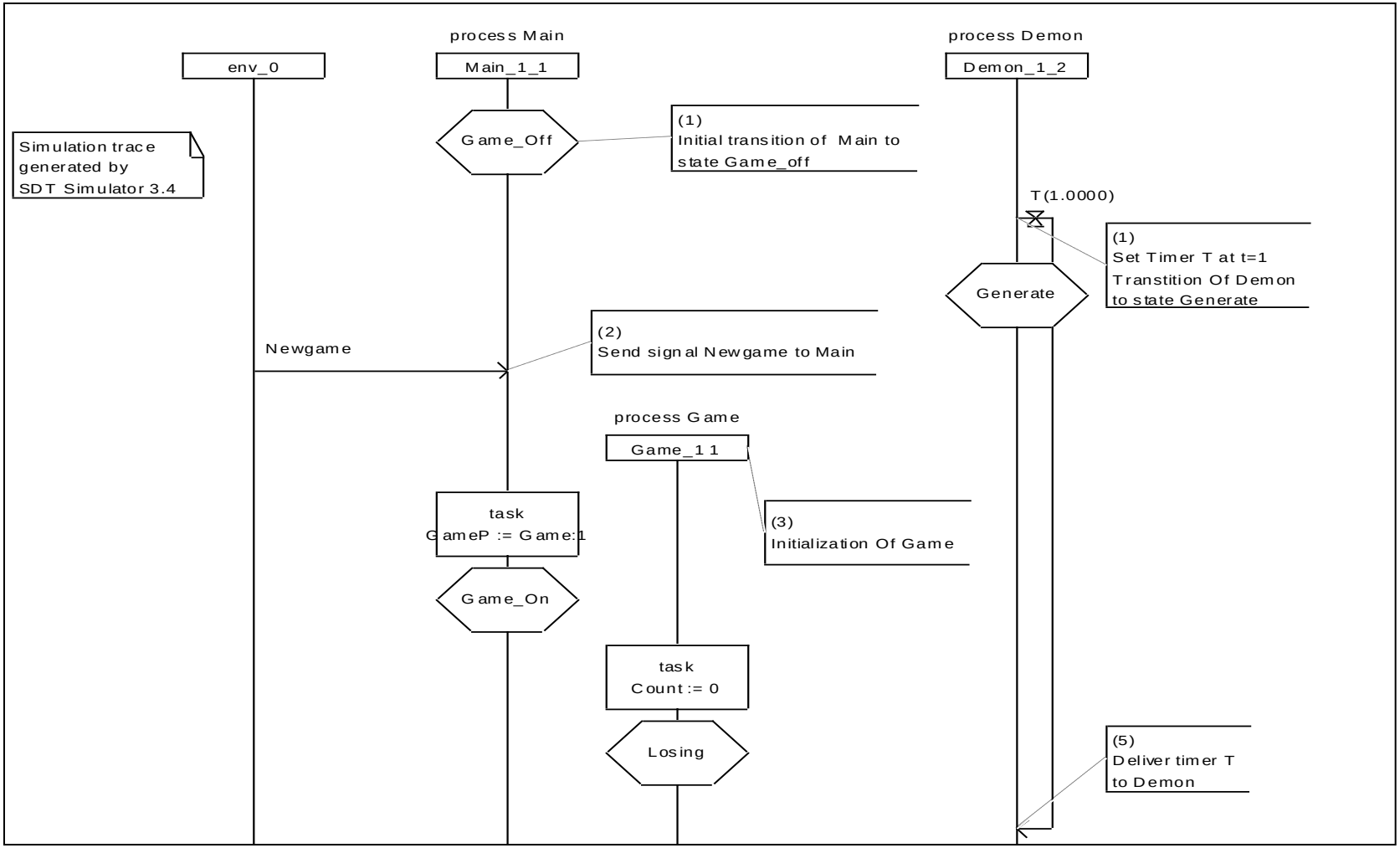

Process Game
1(1)
DCL
Count Integer;
Count:=0
Losing
Probe
Bump
Lose
Winning
Count:=
Count-1
Bump
Probe
-
Losing
Win
Count:=
Count+1
-
*
Result
GameOver
Score
(Count)
-

Παράδειγμα

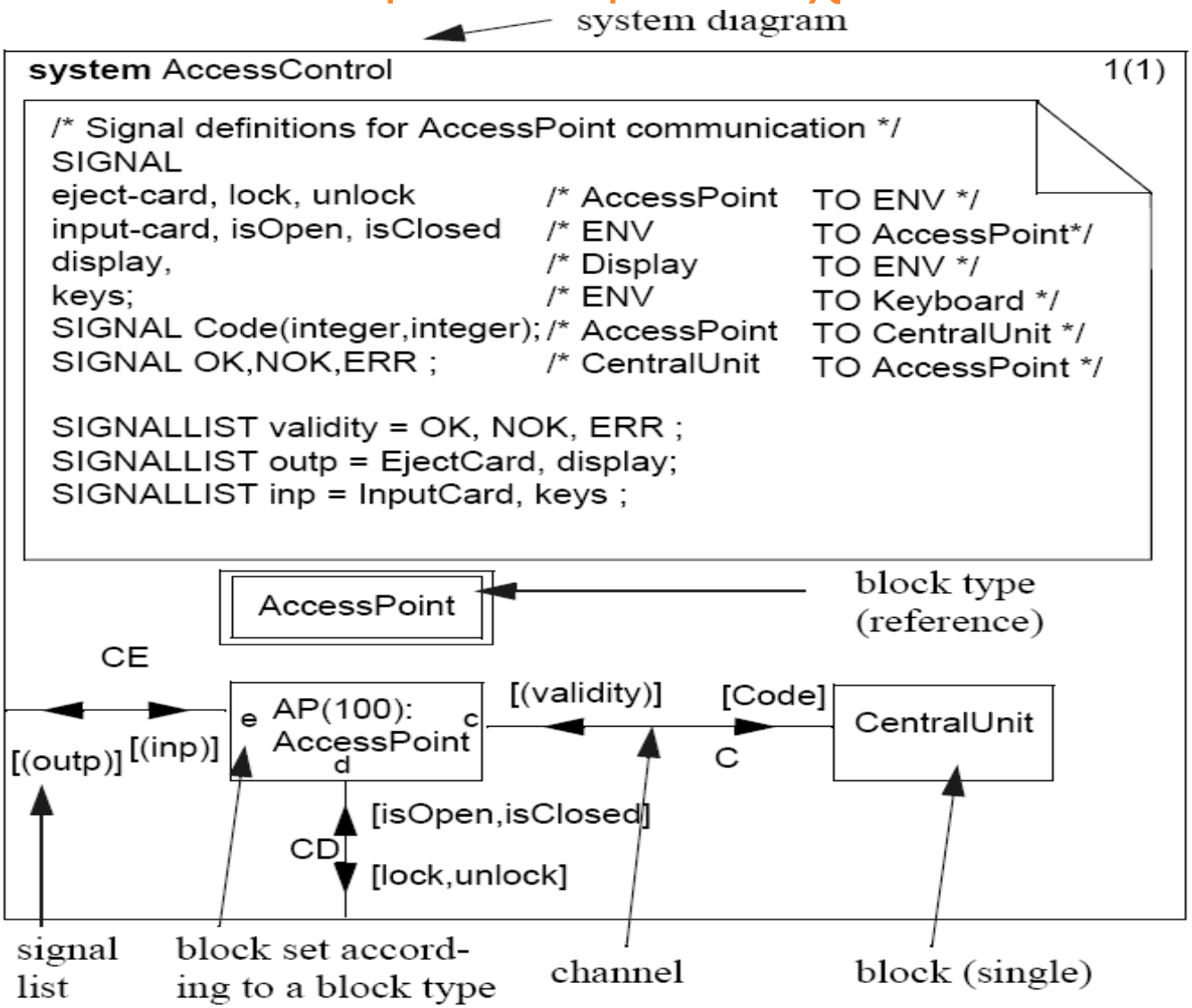

Block DemonBlock
1(1)
C3
R1
[Bump]
Demon(1,1)


Process Demon
Generate
Timer T;
Set
(Now+1, T)
T
Generate
Bump
Set
(Now+1, T)
-

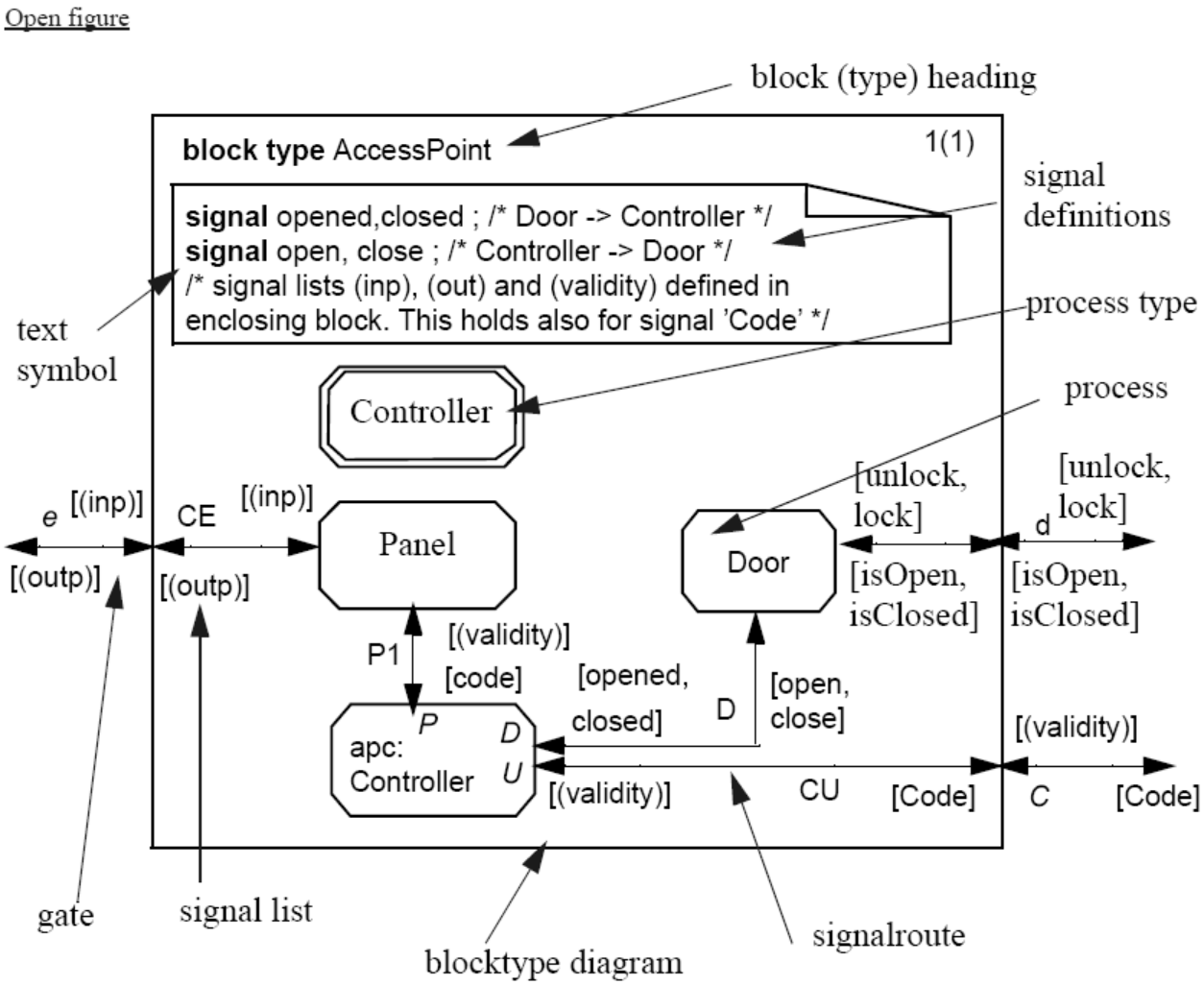
Παράδειγμα


process Main
process Demon
env_0
Main_1_1
Demon_1_2
Simulation trace generated by SDT Simulator 3.4
Game_Off
(1) Initial transition of Main to state Game_off
T(1.0000)
(1) Set Timer T at t=1 Transtition Of Demon to state Generate
Generate
Newgame
(2) Send signal Newgame to Main
process Game
Game_1 1
task GameP := Game:1
(3) Initialization Of Game
Game_On
task Count := 0
Losing
(5) Deliver timer T to Demon

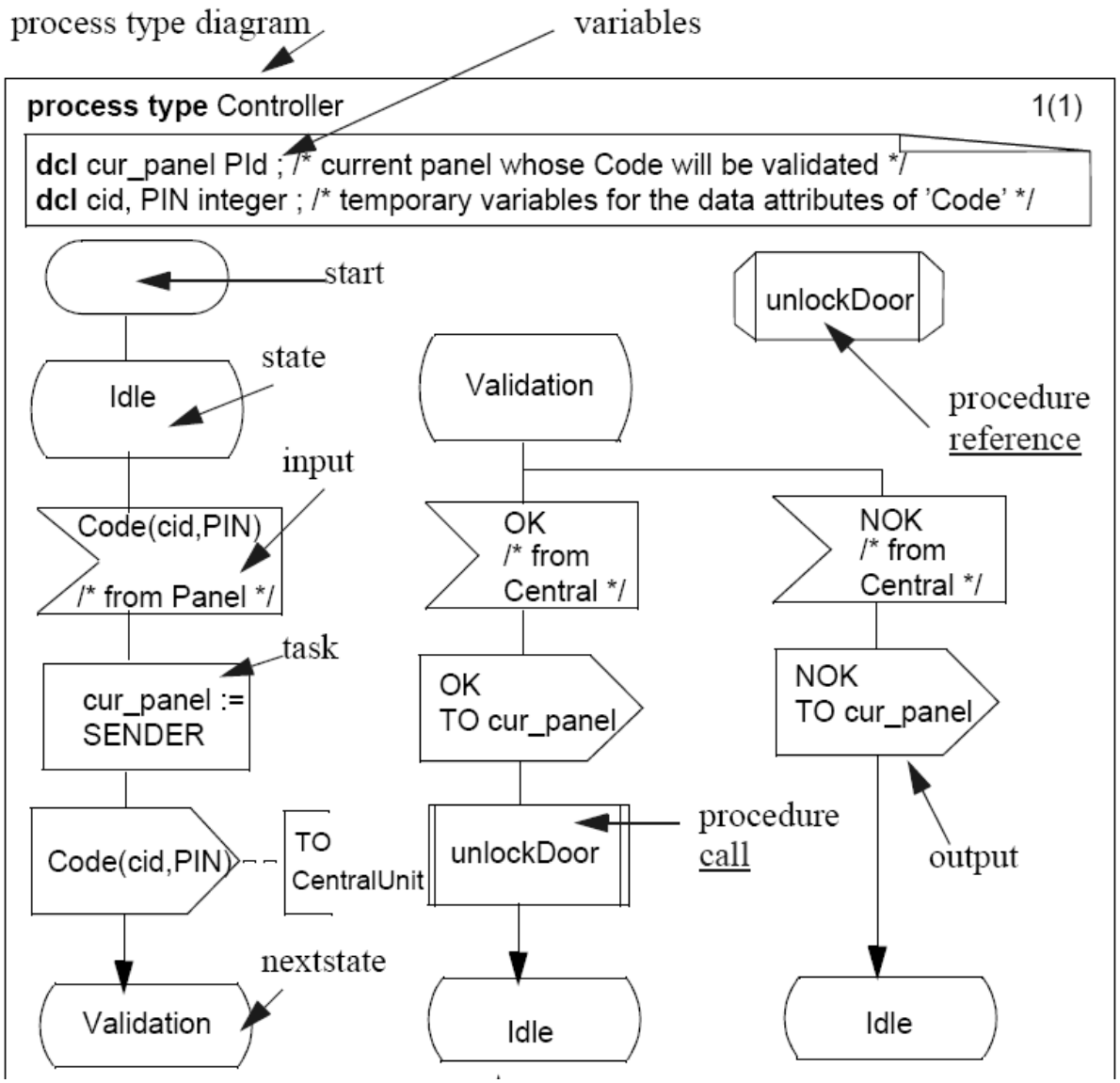
Περισσότερα στοιχεία



Περισσότερα στοιχεία

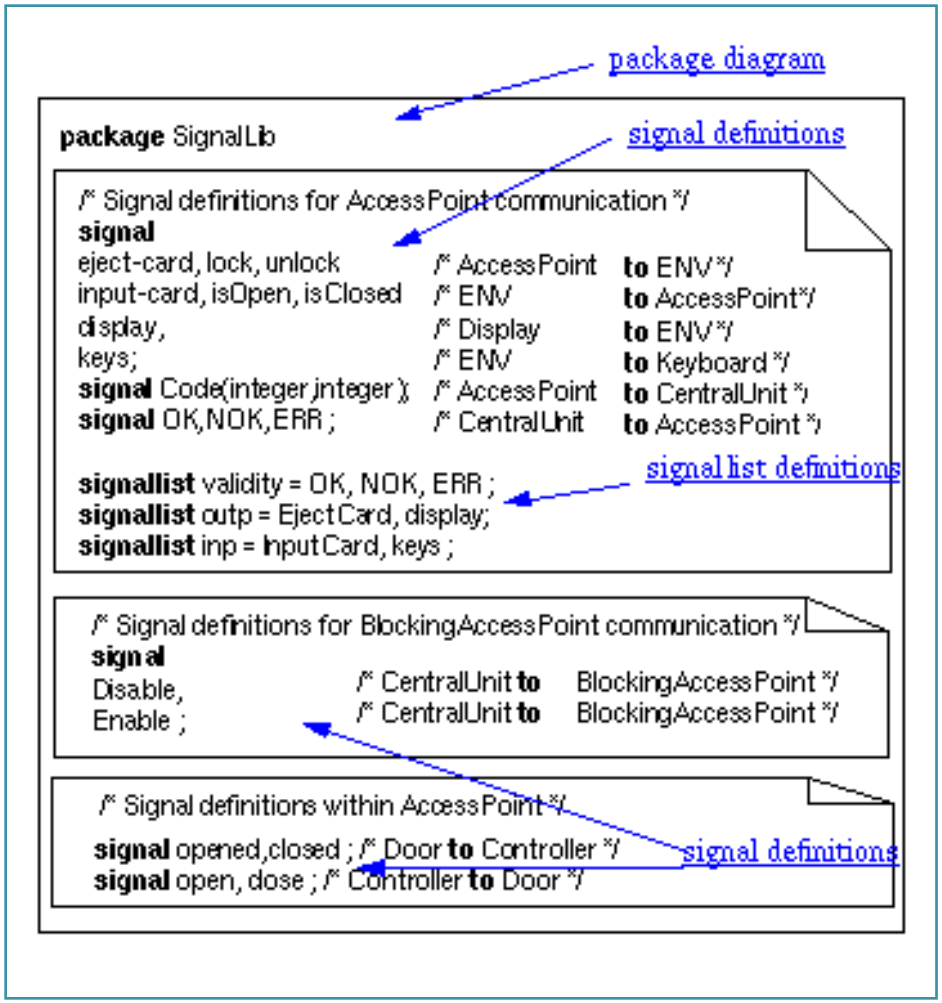


Περισσότερα στοιχεία



Διάγραμμα Πακέτου

A package is a set of types. Types that are only used in one system will normally be defined as part of the system specification, but for convenience they may be collected and defined in a package and then used by the system. If a set of related types are to be used in many systems within a specific application domain, then a package is the right place to define the types



SDL Sorts

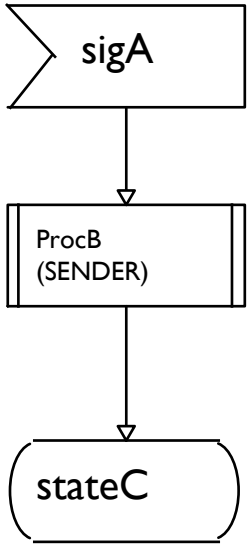
- Each variable is of a particular "sort" (type)
  - Possible values (e.g., integer numbers)
  - Operators on those values (e.g., +, *)
  - Literals (e.g., "zero", "1", "2")

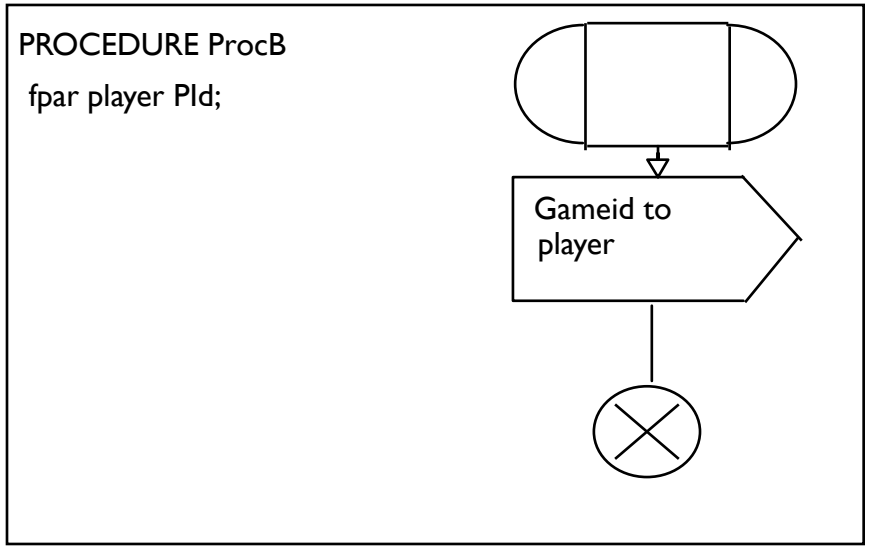
- Built-in sorts: integer, Boolean, real, character, and string

- Can be combined in structures, arrays, enumerations, and sets

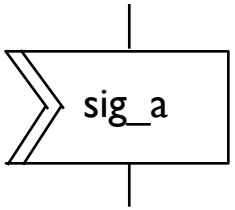
Procedure

- PROCEDURE: similar to a subroutine
  - allow reuse of SDL code sections
  - reduce size of SDL descriptions
  - can pass parameters by value (IN) or by reference (IN/OUT)

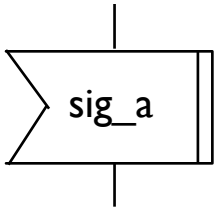




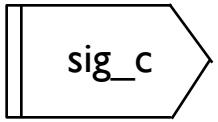
Priority & Internal Inputs



- Priority inputs are inputs that are given priority in a state
- If several signals exist in the input queue for a given state, the signals defined as priority are consumed before others (in order of their arrival)

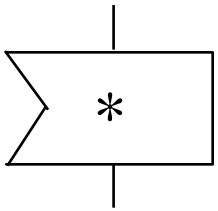


- Internal Input/Outputs signals are used for signals sent/received within a same FSM or SW component

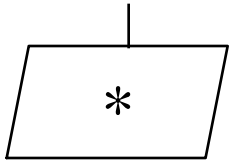


- There is no formal definition when they should be used.

Shorthands - All Other Input/Save

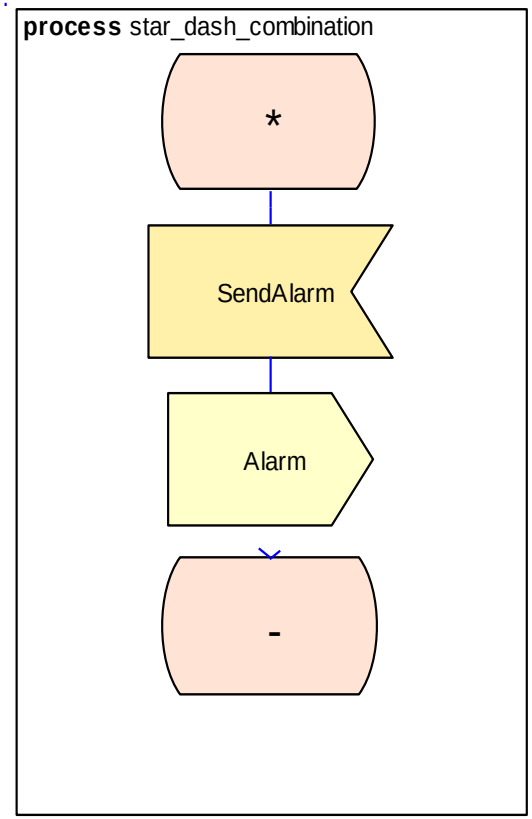


- The input with an asterisk covers all possible input signals which are not explicitly defined for this state in other input or save constructs



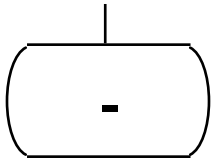
- The Save with an asterisk covers all possible signals which are not explicitly defined for this state in other input or save constructs

Shorthands - Same State

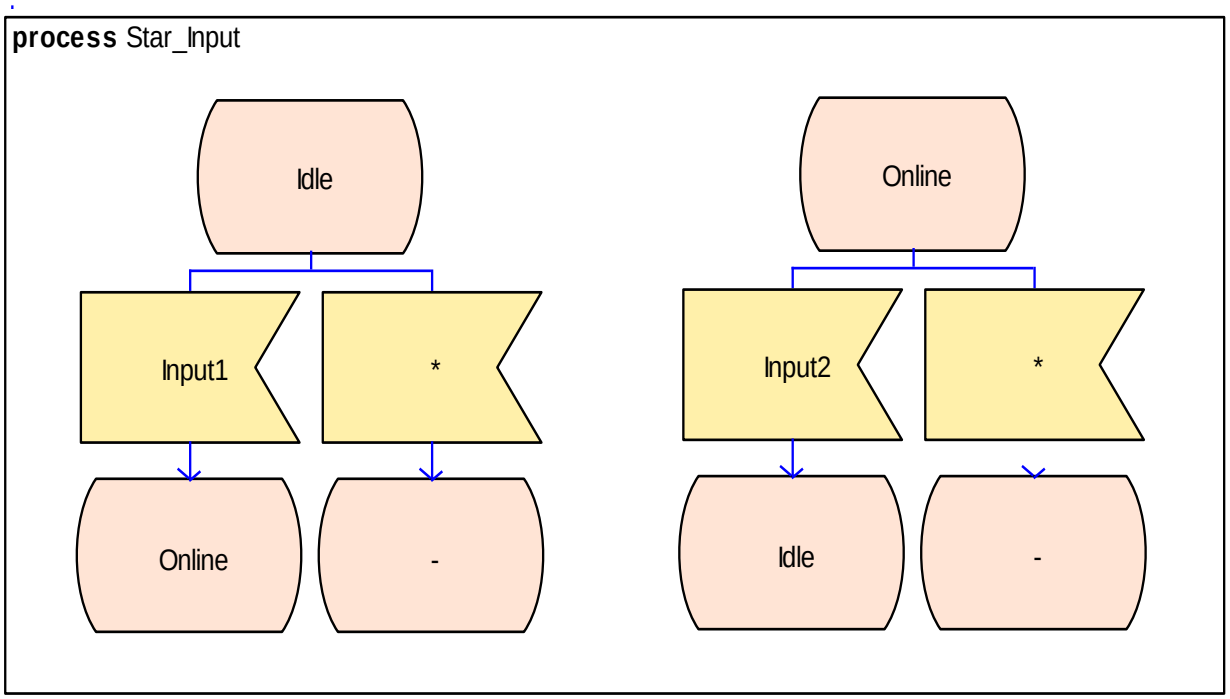


- When next state is same as current state the "dash" symbol may be used instead of state name.

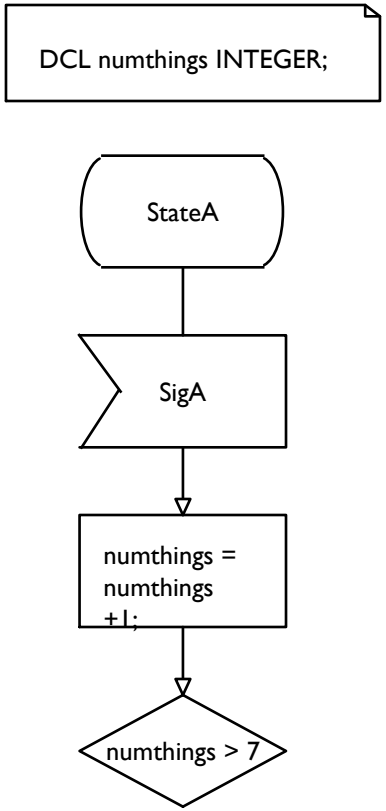
- This is particularly useful in combination with * (any state).



Shorthands Example



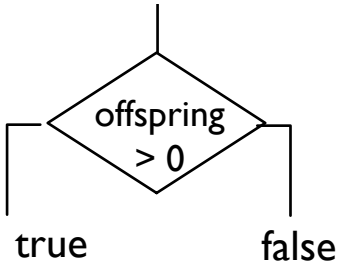
Specification of Data in SDL



- SDL diagrams can contain variables
- Variables are declared using the DCL statement in a text box.
- Variables can set in a task box and read in decisions
- A data type is called a sort in SDL

Dynamic Processes

- Processes can be created and destroyed in SDL
- Each process has a unique process id. The *self* expression returns the process id of the current process.



- Processes are created within a SDL process using the CREATE symbol. The Create body contains the type of the process to create
- The *offspring* expression returns the process id of the last process created by the process.
- The PROCESS that is created must be in the same block as the process that creates it.

- The Stop symbol is used within the SDL PROCESS to signify that the process stops.

Predefined Sorts (types) in SDL

- ❑INTEGER: signed integer
- ❑NATURAL: positive integer
- ❑REAL: real, float
- ❑CHARACTER: 1 character
- ❑CHARSTRING: string of characters
- ❑BOOLEAN: True or False
- ❑TIME: absolute time, date (syntype of REAL)
- ❑DURATION: a TIME minus a TIME (syntype of REAL)
- ❑PID: to identify a process instance

Predefined Sorts (types) in SDL

- Operations := (assignment) , = (equality) and /= (nonequality) are defined for all sorts

- INTEGER -, +, *, /, >, <, >=, <=, Float (Integer to Real), Mod (modulo), Rem (remainder)

- REAL -, +, *, /, >, <, >=, <=, Fix (Real to Integer)

- NATURAL Like Integer

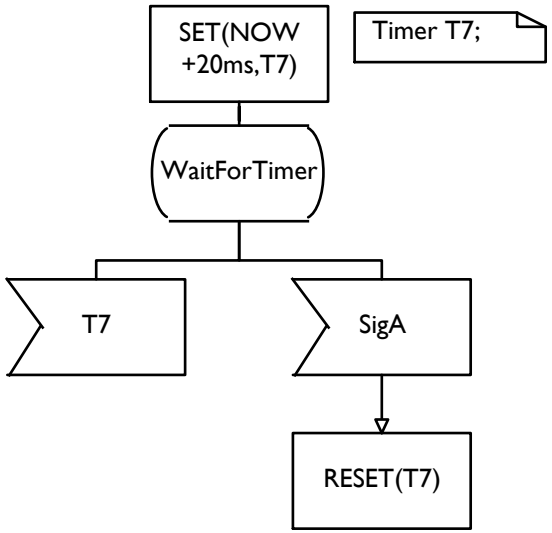
- CHARACTER Chr (Integer to Char), Num (Char to Integer), >,<,>=,<=

- CHARSTRING Mkstring (Char to Charstring), Length, First, Last, // (concatenation), Substring

- BOOLEAN True, False, NOT, AND, OR, XOR

- PID Self, Sender, Offspring, Parent

Specification of Timers in SDL



- Timer is an object capable of generating an input signal and placing this signal to the input queue of the process. Signal is generated on the expiry of pre-set time.

- SET(NOW+20ms,T7): sets a T7 timeout in 20ms time.

- RESET(T7): cancels the specified timeout.

Communication Related SDL Terms

- signal:
  - The primary means of communication is by signals that are output by the sending agent and input by the receiving agent.

- stimulus:
  - A stimulus is an event that can cause an agent that is in a state to enter a transition.

- channel:
  - A channel is a communication path between agents.

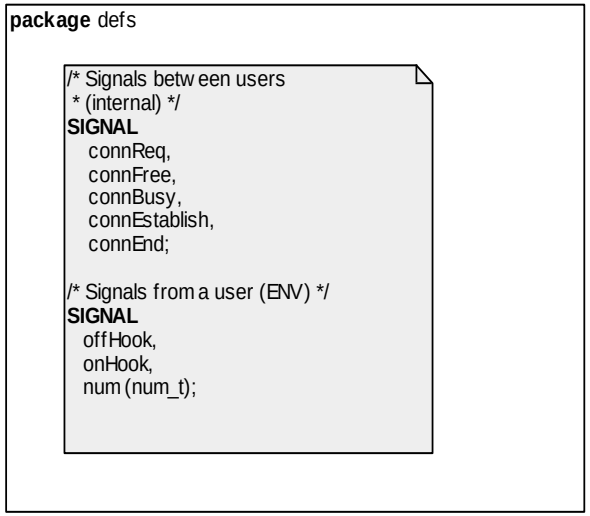
Text Symbol

- Text Symbol is used to group various textual declarations
- It can be located on any type of diagram

Concrete graphical grammar

<text symbol> ::=

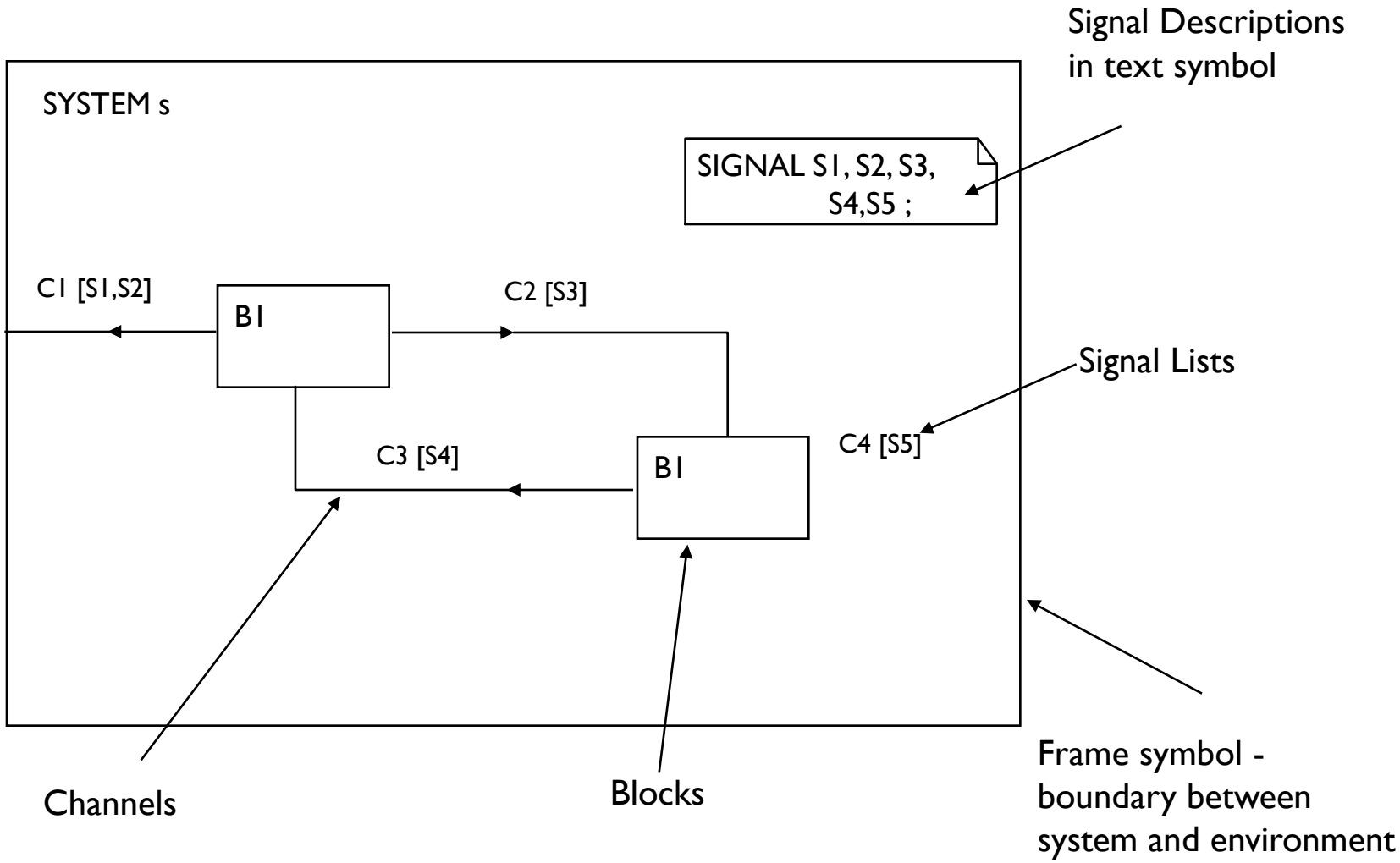
Text Box Example



System Diagram

- ❑Topmost level of abstraction - system level
- ❑Has a name specified by SYSTEM keyword
- ❑Composed of a number of BLOCKs
- ❑BLOCKs communicate via CHANNELs
- ❑Textual Descriptions/Definitions
  - •Signal Descriptions
  - •Channel Descriptions
  - •Data Type Descriptions
  - •Block Descriptions

Example System Diagram



Packages & Libraries

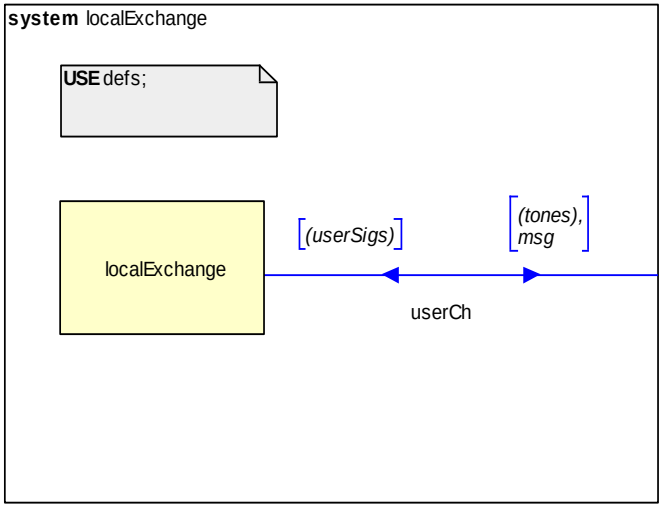
- Since SDL 92 reusable components may be defined as types and placed into libraries called packages.

- This allow the common type specifications to be used in more then a single system.

- Package is defined specifying the package clause followed by the <package name>.

- A system specification imports an external type specification defined in a package with the use clause.

Package Example




```
package defs

/* Signals from a user (ENV) */
SIGNAL
   offHook,
   onHook,
   num (num_t);

SIGNALLIST userSigs =
   offHook,
   onHook,
   num;

/* Signals to a user (ENV) */
SIGNAL
   dialTone,
   ringTone,
   busyTone,
   shortBusyTone,
   connectTone,
   msg (CharString);

SIGNALLIST tones =
   dialTone, ringTone,
   busyTone, shortBusyTone,
   connectTone;
```

SDL Entity Visibility Rules

- Entities are
  - Packages, agents (system, blocks, processes), agent types, channels, signals, timers, interfaces, data types, variables, sorts, signal lists;

- Possible Scope Units are
  - Agent diagrams (System, Block, Process), Data Type Definitions, Package diagrams, task areas, interface definitions ...

- The Entity is <u>visible</u> in the scope unit if
  - is defined in a scope unit
  - the scope unit is specialisation and the entity is visible in base type
  - the scope unit has a "package use clause" of a package where entity is defined
  - the scope unit contains an <interface definition> where entity is defined
  - the entity is visible in the scope unit that defines that scope unit

Additional Structural Concepts in SDL

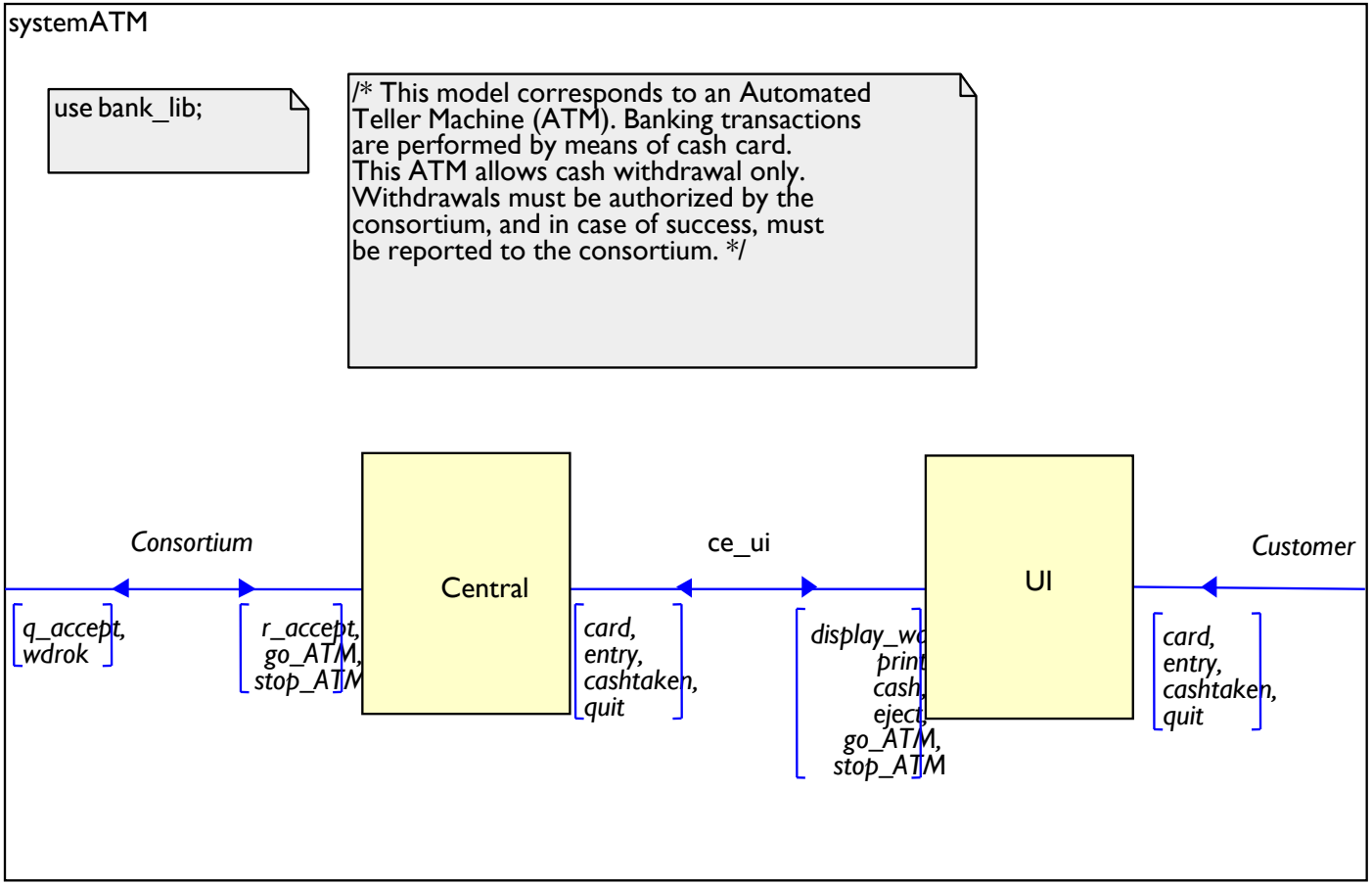
- A tree diagram can be constructed to illustrate the hierarchy of the entire SYSTEM .

- Macros can be used to repeat a definition or a structure. They are defined using the MACRODEFINITION syntax .

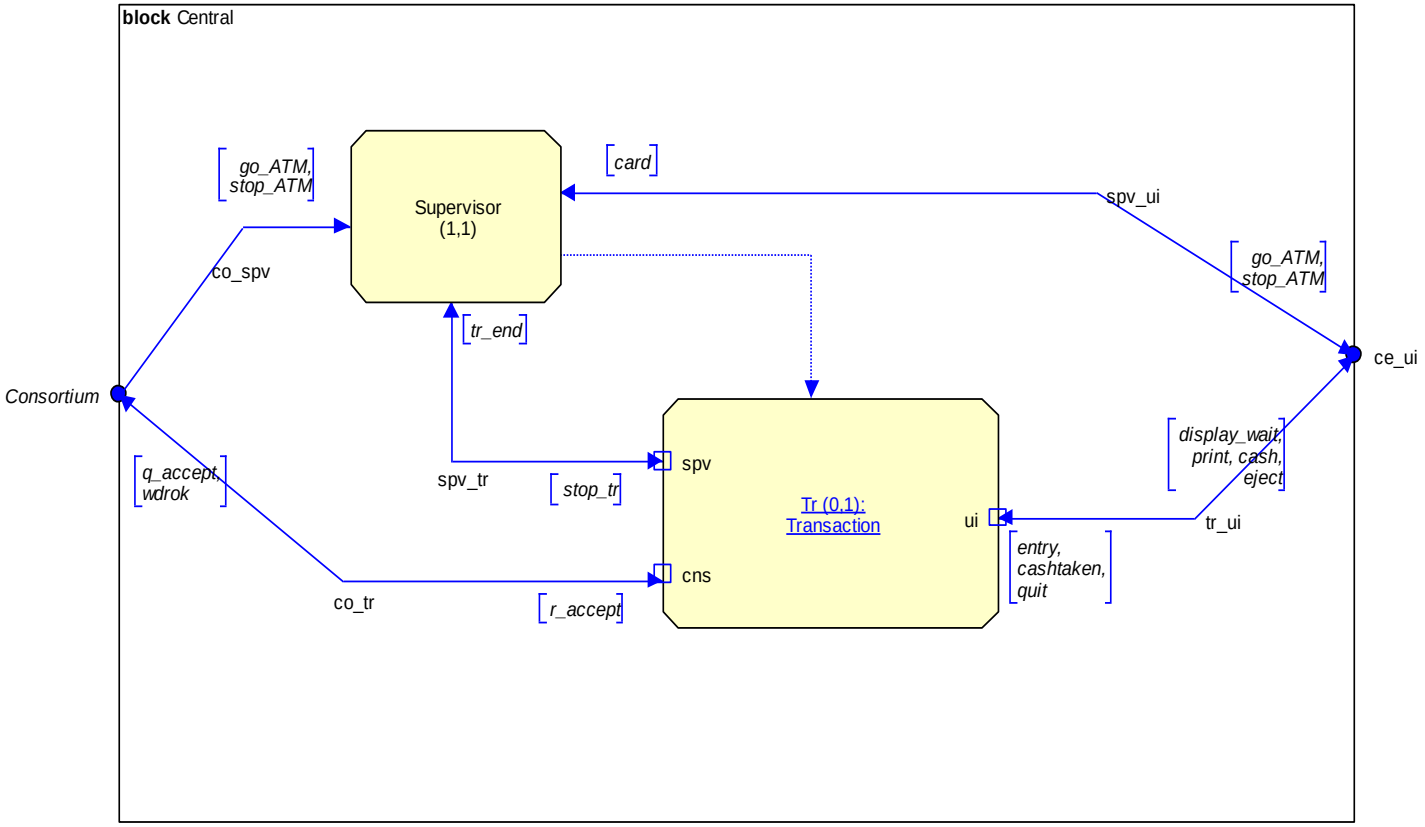
- Paramaterised types exist using the generator construct.

- Gates
  - A gate represents a connection point for communication with an agent type, and when the type is instantiated it determines the connection of the agent instance with other instances.

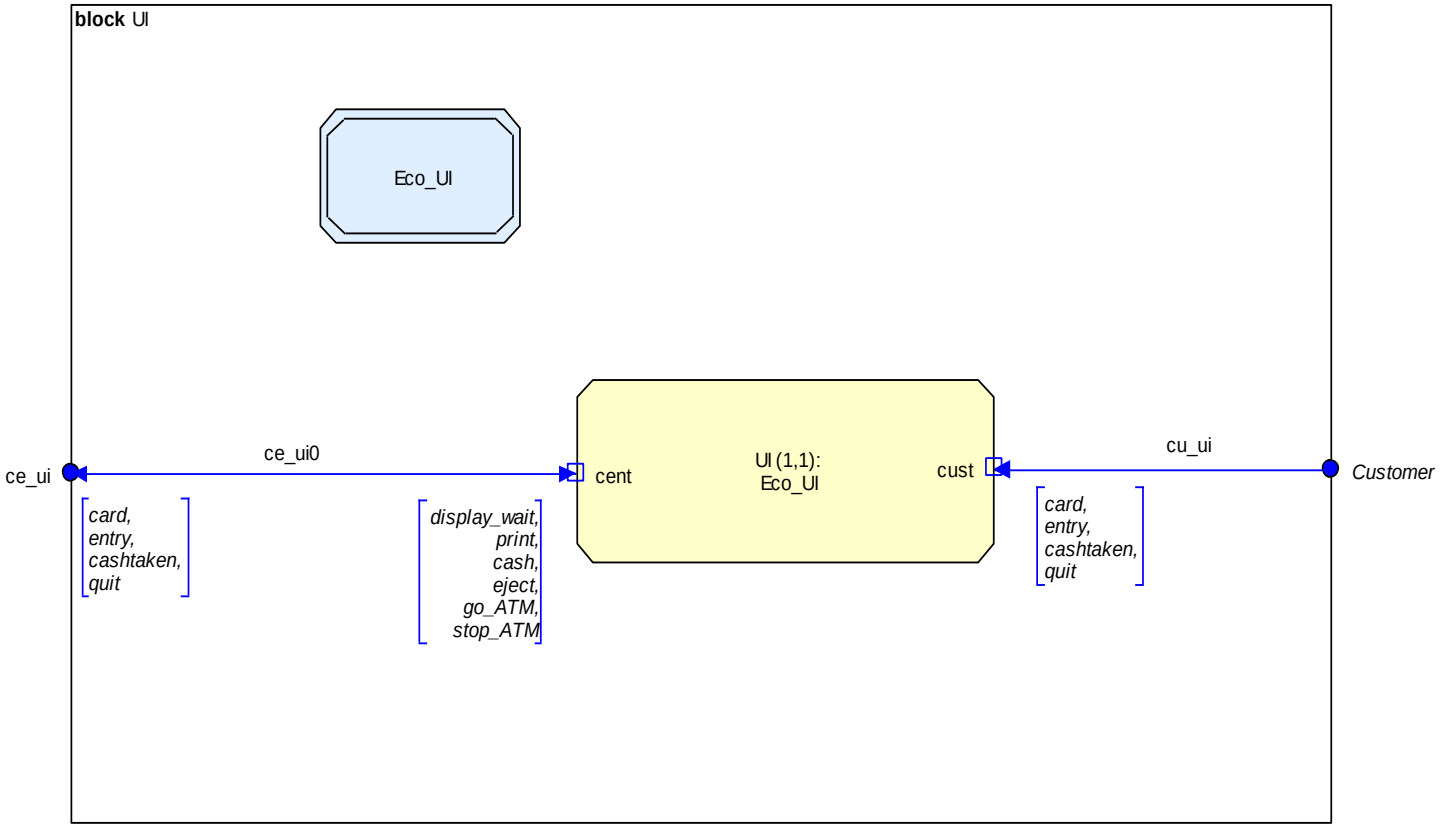
ATM Example - System Diagram



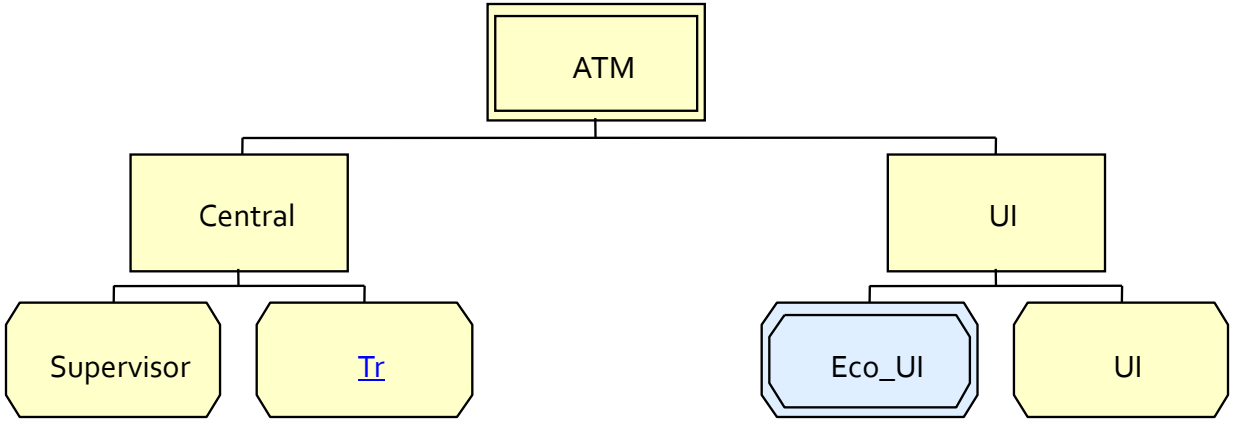
ATM Example - Central Block Diagram



ATM Example – UI Block Diagram



ATM Example – Hierarchy Diagram



ATM Example – Package Bank_lib

package bank_lib

```
/* This SDL components library
contains SDL block and process
types which are useful to
develop banking systems. */
```

```
/* Signals received by the
Transaction Process Type */
signal
entry (Charstring),
cashtaken,
quit,
r_accept (RespConso),
stop_tr;

/* Signals sent by the
Transaction Process Type */
signal
display_wait (Charstring),
print (Charstring),
cash (Charstring),
eject,
tr_end,
q_accept (QuestConso),
wdrok (CashCard, Charstring);

/* Additional signals for
Basic_ATM_UI */
signal
card (CashCard),
go_ATM,
stop_ATM;
```

```
/* Types used by the Transaction Process */
newtype CashCard
struct
  id Integer;
  expirDate Integer;
  pswd Charstring;
operators
  checkCard: CashCard -> Boolean;
  checkPswd: CashCard, Charstring -> Boolean;
operator checkCard;
  fpar cc CashCard;
  returns res Boolean;
  start;
  task res := (cc!expirDate > 9701) and (cc!id /= 0);
  return;
endoperator;
operator checkPswd;
  fpar cc CashCard, cpw  Charstring;
  returns res Boolean;
  start;
  task res := (cc!pswd = cpw);
  return;
endoperator;
endnewtype;

QuestConso::= sequence {
  cardData CashCard,
  amount Charstring};

RespConso ::= sequence {
  cardData CashCard,
  accept Boolean,
  amount Charstring optional};
```

```
/* This package contains:
- ASN.1 declarations (QuestConso, RespConso)
mixed into SDL declarations
- Process types (Transaction, Basic_ATM_UI)
- Virtual transitions (in Transaction)
- Axioms (Newtype CashCard)
*/
```

Transaction

```
/* This implements a
simplified banking
transaction. */
```

Basic_ATM_UI

```
/* This implements a
basic terminal
interacting with the
customer. */
```

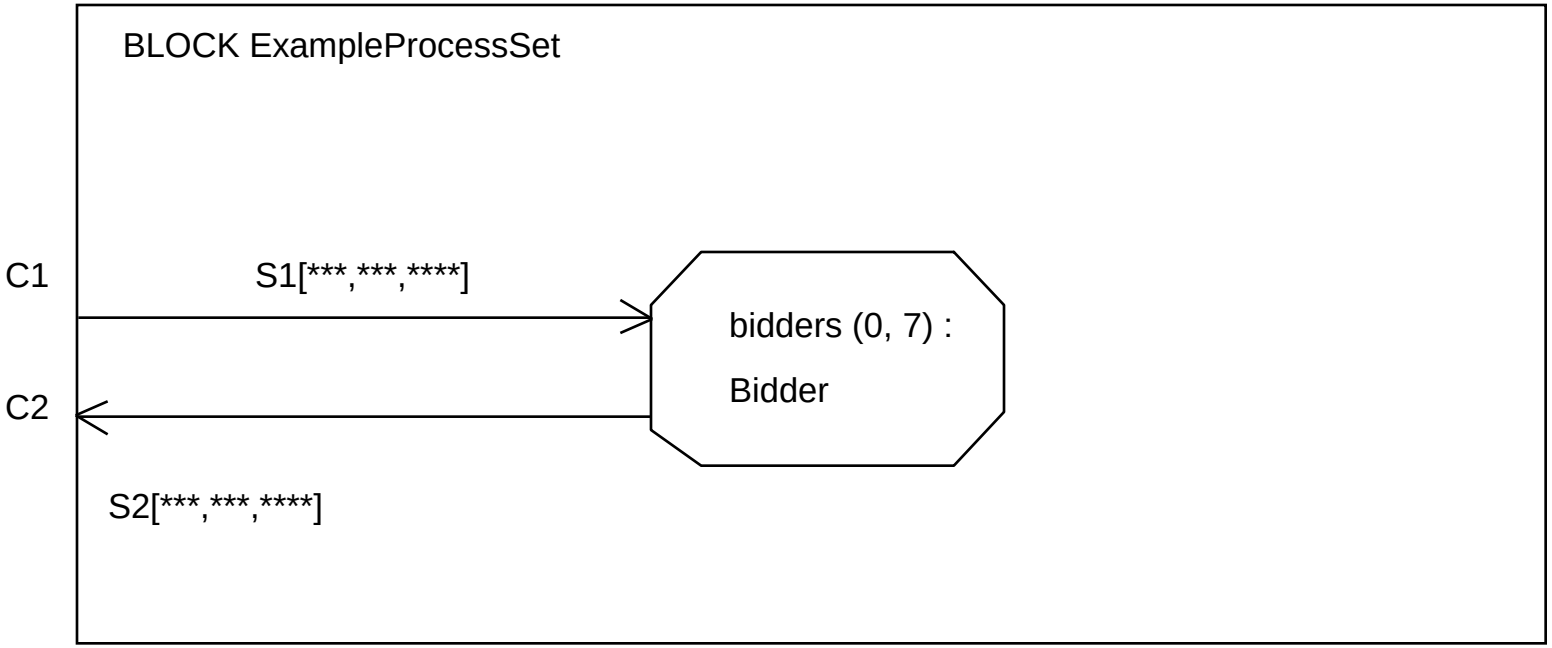
Dynamic Processes

- Dynamically created processes become part of an instance set.

- The instance set in the block diagram contains two variables, the number initial process instances and the maximum number of instances.

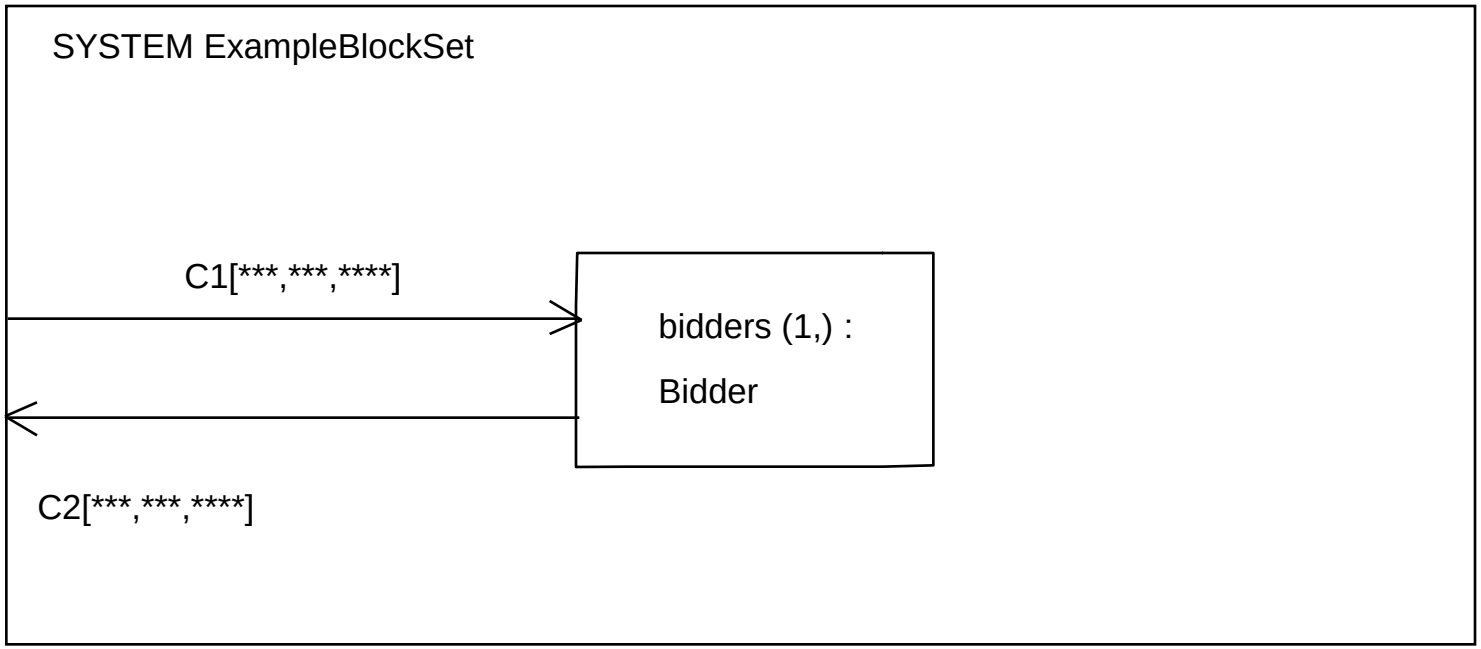
Process Sets

- The following Describes a set of Identical Processes
- Initially there are no members of the set
- Can be up to 7 members in the set



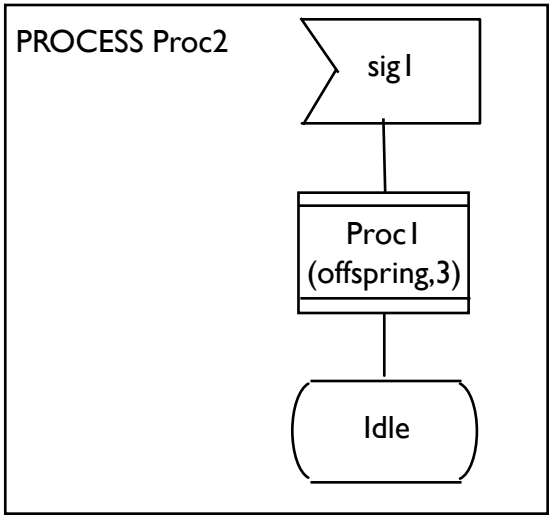
Block Sets

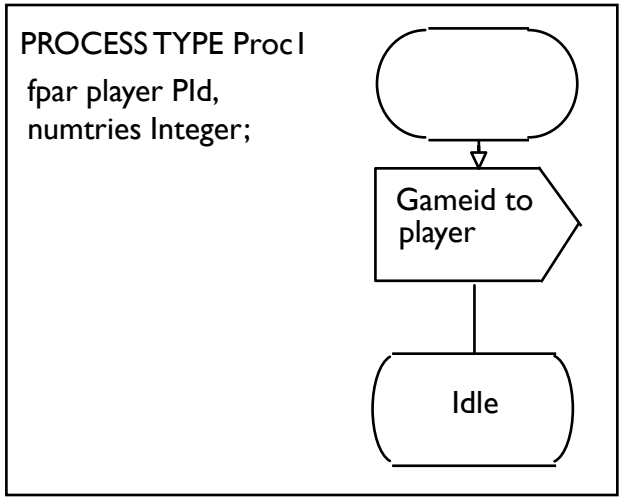
- The following Describes a set of Identical Blocks
- Initially there is one member of the set
- There is no limit to the number of members in the set



Formal Parameters

- Dynamic processes can have data passed into them at creation time using Formal Parameters
- Similar to C++ constructor



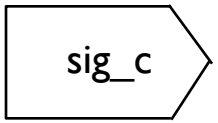


Addressing Signals

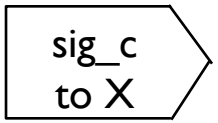
- The destination of an output can be defined in a number of ways:

- Implicit when only one destination is possible

- An explicit destination can be named using the keyword *to* X, where X is of type Pid.
  - SELF, giving the address of the process itself
  - SENDER, giving the address of the process from which the last consumed signal has been sent;
  - OFFSPRING, giving the address of the process that has been most recently created by the process; and
  - PARENT, giving the address of the creating process.

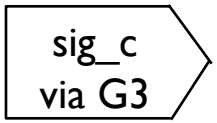


Implicit Addressing

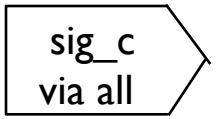


Explicit Addressing

Addressing Signals



- The term "via" can be used followed by a signal route or channel. This means it can be sent to all process attached to a particular channel or signal route(multicasting).



- Or it can be sent everywhere it possibly can using the "via all" qualifier (broadcasting).

Daemon Game Example

- The Z.100 standard partially defines an example of SDL in the form of a game called DaemonGame. A modified version is described here.

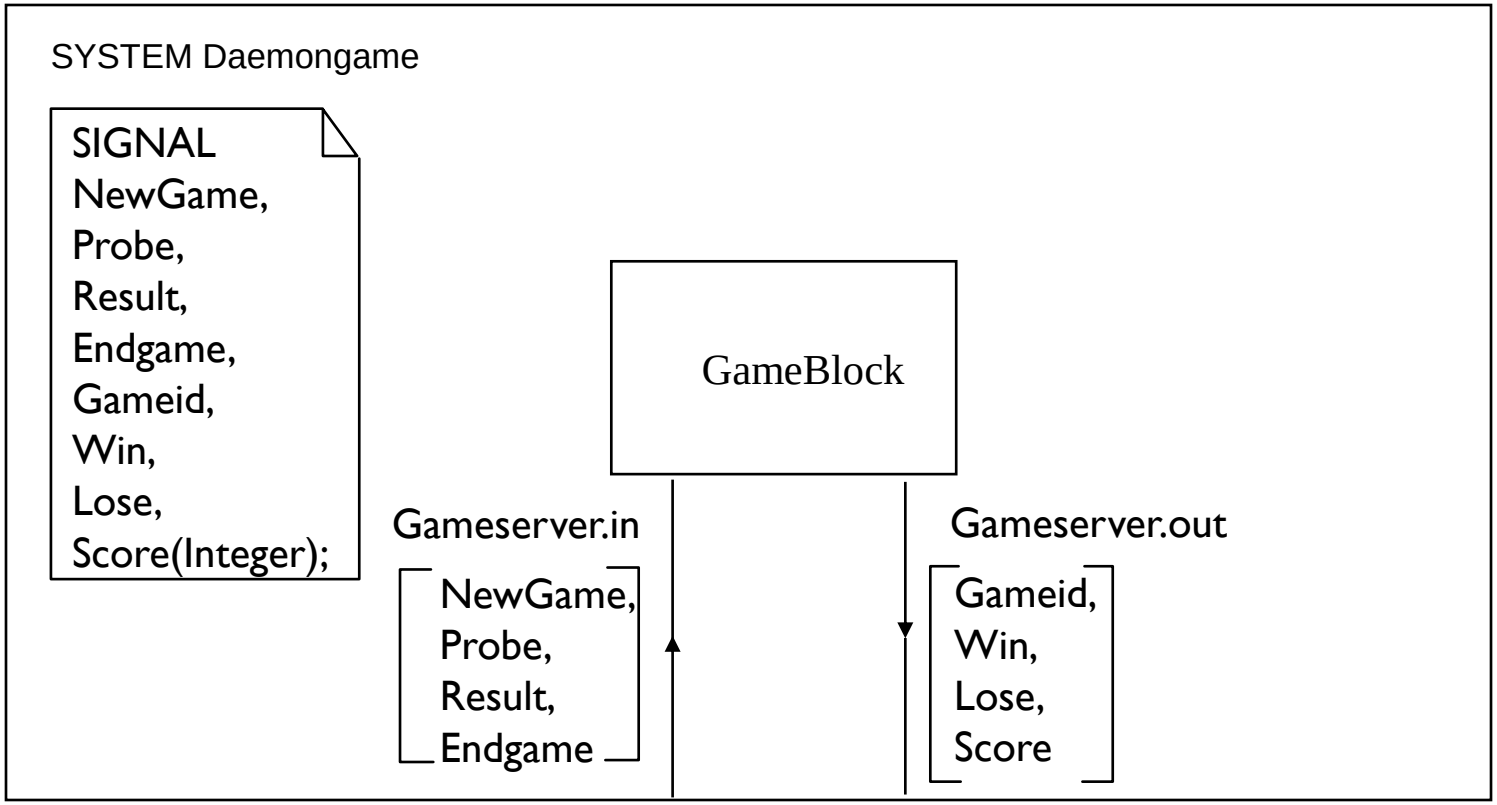
- The game consists of a quickly oscillating state machine, oscillating between odd and even.

- At random intervals the player queries the state machine.

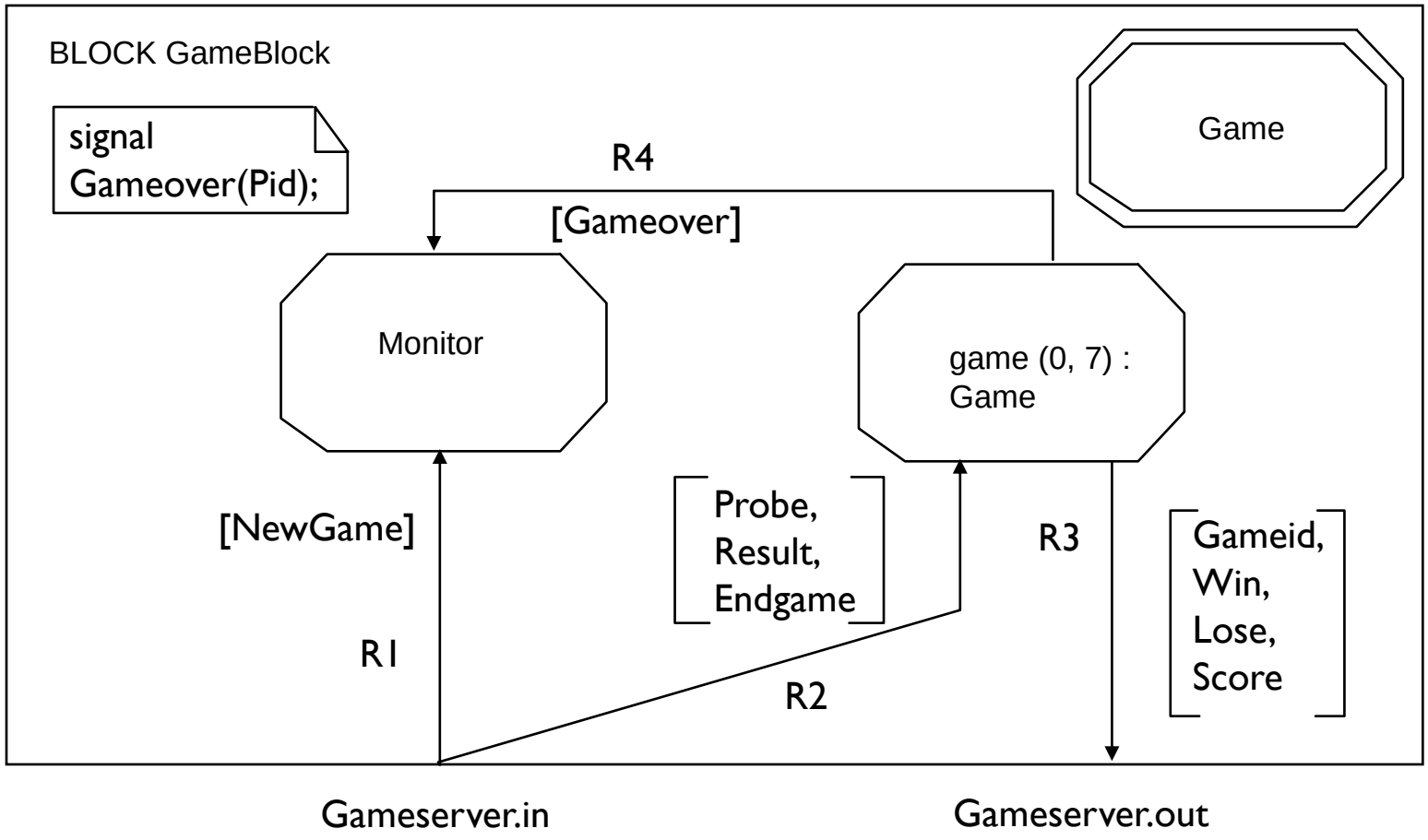
- If the machine is in the odd state the player wins.

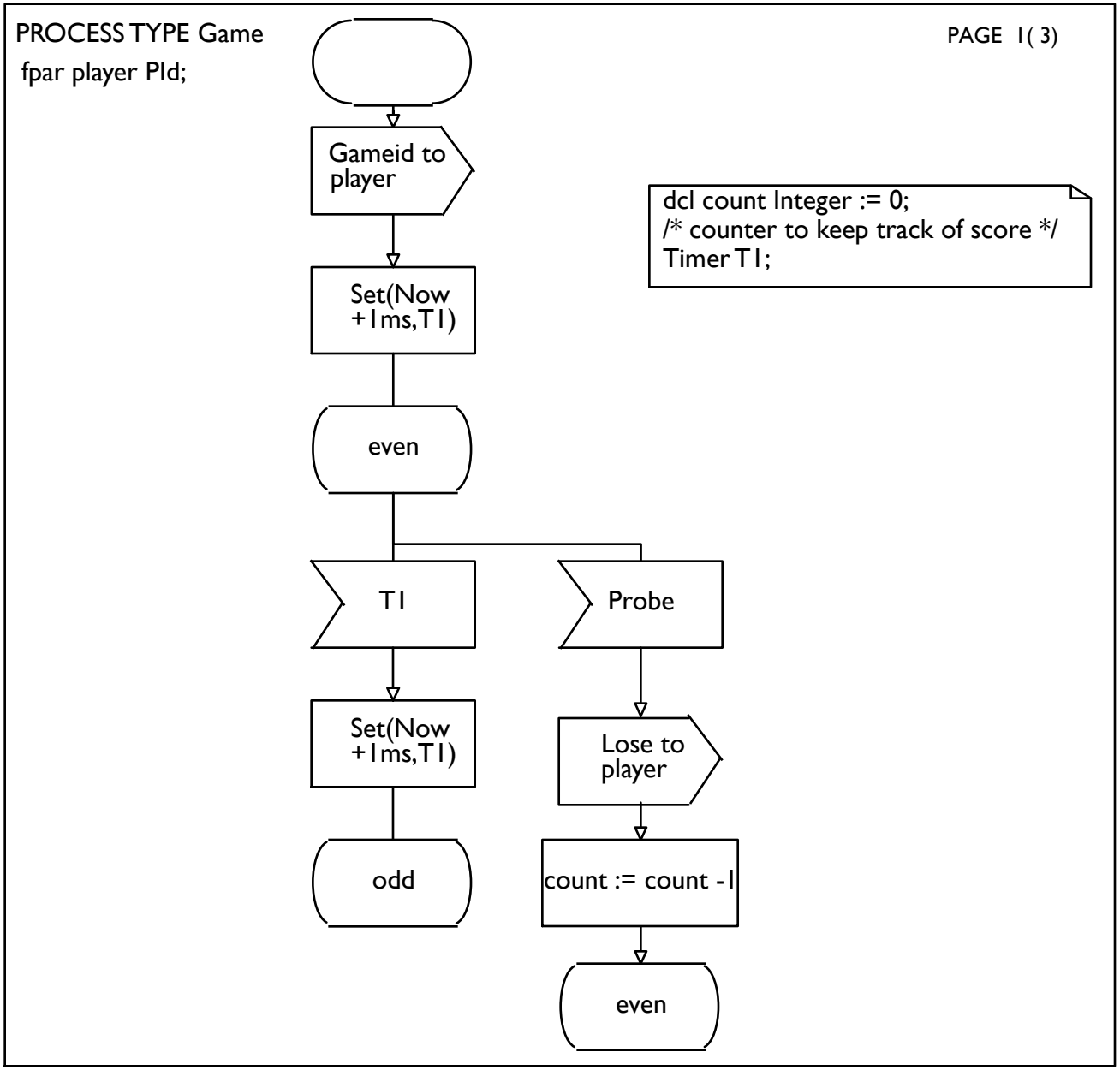
- If the machine is in the even state the player looses.

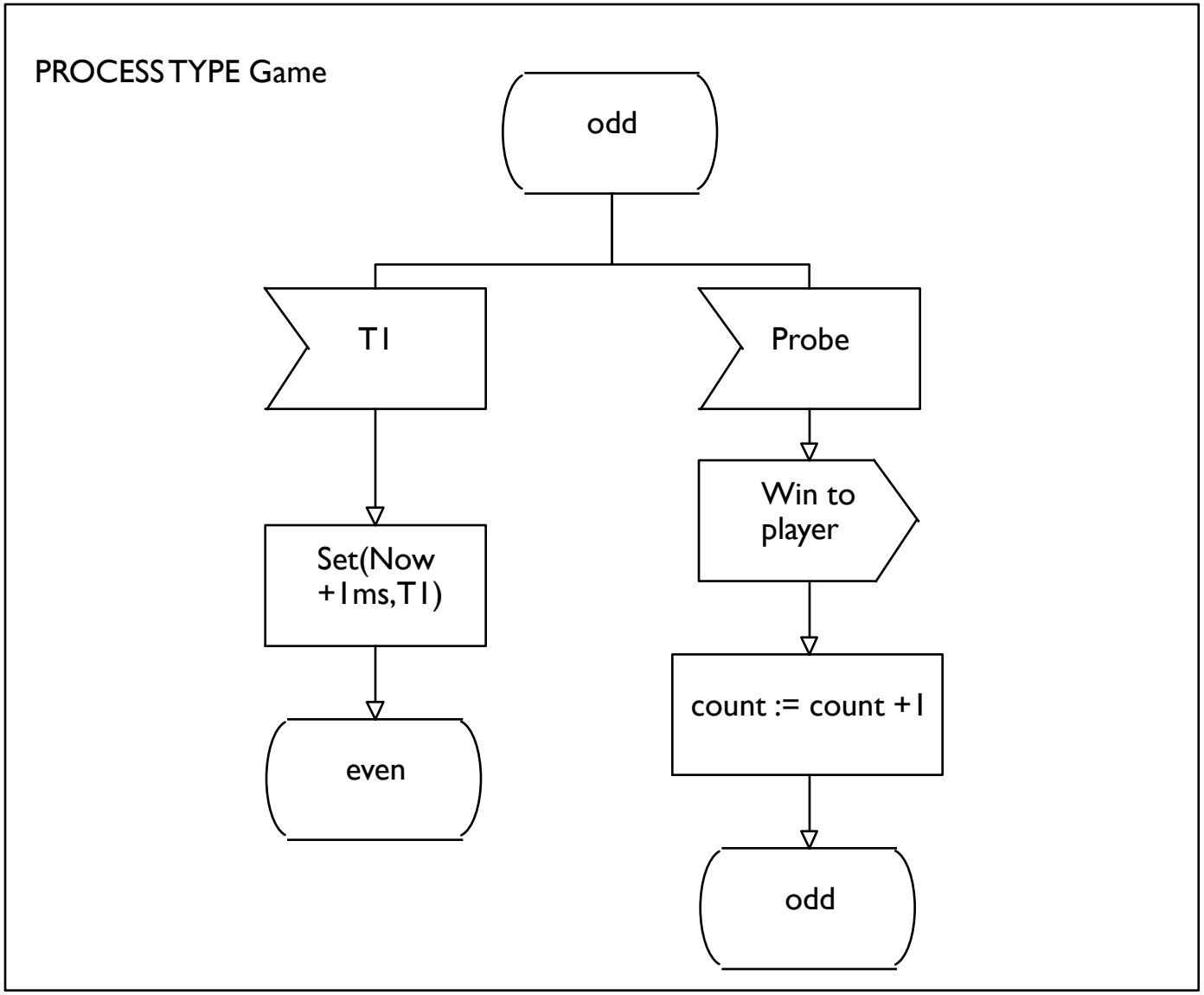
System Diagram

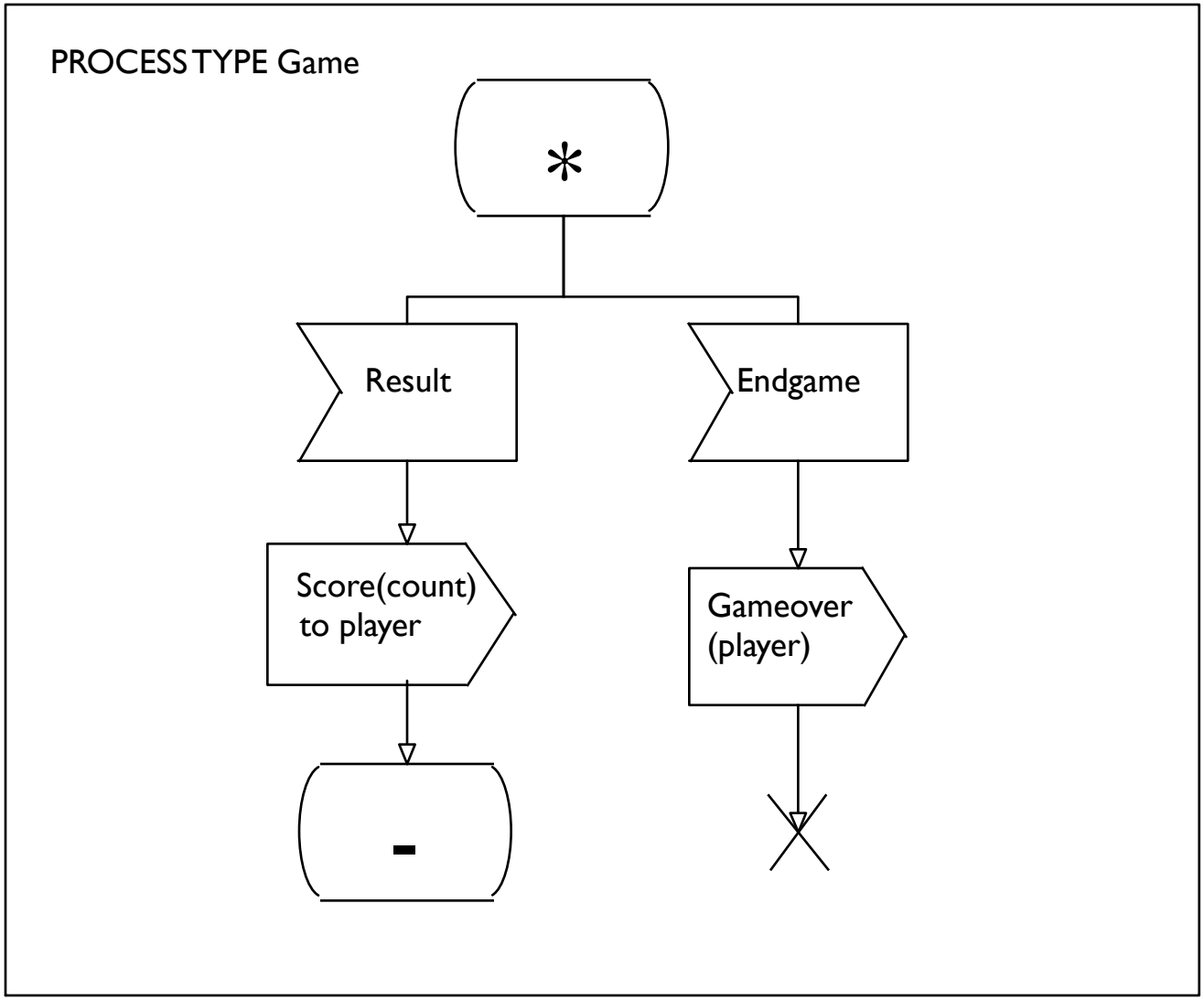


Block Diagram




PROCESS TYPE Game
fpar player PId;
PAGE 1(3)
Gameid to player
dcl count Integer := 0;
/* counter to keep track of score */
Timer T1;
Set(Now +1ms,T1)
even
T1
Probe
Set(Now +1ms,T1)
Lose to player
odd
count := count -1
even


PROCESS TYPE Game
odd
T1
Probe
Set(Now +1ms,T1)
Win to player
count := count +1
even
odd


PROCESS TYPE Game
*
Result
Endgame
Score(count)
to player
Gameover
(player)
-

Transition Table

State	Input	Task	Output	NextState
even	T1	Set(Now+1ms T1)	———	odd
even	Probe	count := count -1	Lose to player	even
odd	T1	Set(Now +1ms T1)	———	even
odd	Probe	count := count +1	Win to player	odd
odd	Result	———	Score(count) to player	odd
odd	Endgame	———	Gameover	STOP
even	Result	———	Score(count) to player	even
even	Endgame	———	Gameover	STOP

Notes on Example

- SDL is case insensitive
- One Block Diagram for each Block in System Diagram
- One Process Diagram for each Process in Block Diagram
- Only Signals listed on SignalRoute used in Process Diagram
- * State used to represent any state
- - NextState means return to the previous state (i.e. no state change)

Notes on Example

- To transition out of state requires input.
- Process Diagrams are of type PROCESS TYPE rather than PROCESS because they are part of a Process Set.
- Gameover message always sent to Monitor so no need for explicit destination address.
- Lose, Score, Win GameId require explicit destination address.
- player passed in as a formal parameter, like a C++ constructor.

Creating new Data Types

- New data types can be defined in SDL.
- An example data definition is shown below

```
newtype even literals 0;
    operators
        plusee: even, even -> even;
        plusoo: odd, odd -> even;
    axioms
        plusee(a,0) == a;
        plusee(a,b) == plusee(b,a);
        plusoo(a,b) == plusoo(b,a);
endnewtype even; /* even "numbers" with plus–
depends on odd
```

Creating new Data Types

- A *syntype* definition introduces a new type name which is fully compatible with the base type

- An *enumeration sort* is a sort containing only the values enumerated in the sort

- The *struct* concept in SDL can be used to make an aggregate of data that belongs together

- The predefined generator *Array* represents a set of indexed elements

Creating new Data Types Data Types and Inheritance

- New Data types can inherit from other data types in SDL

```
newtype bit inherits Boolea
   literals 1 = True, 0 = False;
   operators ("not", "and", "or")
   adding operators
      Exor: bit,bit -> bit;
   axioms
      Exor(a,b) == (a and (not b)) or ((not a) and b));
endnewtype bit;
```

True, False are renamed to 1 & 0

Operators that are perserved

From this point new items are defined

- Most SDL protocol specifications used ASN.1 to describe data.
- Z.105 describes how SDL and ASN.1 can be used together.

Key SDL Features (1 of 2)

- Structure
  - Concerned with the composition of blocks and process agents.
  - SDL is structured either to make the system easier to understand or to reflect the structure (required or as realised) of a system.
  - Structure is a strongly related to interfaces.
- Behavior
  - Concerns the sending and receiving of signals and the interpretation of transitions within agents.
  - The dynamic interpretation of agents and signals communication is the base of the semantics of SDL.
- Data
  - Data used to store information.
  - The data stored in signals and processes is used to make decisions within processes.

Key SDL Features (2 of 2)

- Interfaces
  - Concerned with signals and the communication paths for signals.
  - Communication is asynchronous: when a signal is sent from one agent there may be a delay before it reaches its destination and the signal may be queued at the destination.
  - Communication is constrained to the paths in the structure.
  - The behaviour of the system is characterised by the communication on external interfaces.
- Types
  - Classes can be be used to define general cases of entities (such as agents, signals and data).
  - Instances are based on the types, filling in parameters where they are used.
  - A type can also inherit from another type of the same kind, add and (where permitted) change properties.

References

1. A. Mitschele-Thiel, Systems Engineering with SDL“, John Wiley & Sons, Ltd, 2001, Print ISBN: 9780471498759, Online ISBN: 9780470841969

2. J. Ellsberger, D. Hogrefe, A. Sarma, SDL Formal Object-Oriented Language for Communication Systems“, Prentice Hall 2007, ISBN: 0136328865

3. Oleg Chistokhvalov, http://www.it.lut.fi/kurssit/05-06/Ti5315800/Slides/Lecture_7/lecture7.html

4. Dr. Junzhao Sun http://www.ee.oulu.fi/research/tklab/courses/521265A/lectures/ch5_SDL.pdf

5. SDL forum society: http://www.sdl-forum.org/SDL/index.htm