SDL

Pau Fonseca i Casas; pau@fib.upc.edu

Outline

- Introduction to SDL
  - Purpose & Application
  - Key SDL features
  - SDL grammar
  - SDL history
- Static SDL Components
  - Description of the System Structure
  - Concepts of System, Block and Process
  - Communication Paths: Channels, Signals
- SDL to represent simulation models
  - Discrete simulation models.
  - Agent based models.

Introduction to SDL

Why SDL exists?

Why SDL exists ?

- **Specification and Description Language** (SDL) is a specification language targeted at the unambiguous specification and description of the behaviour of reactive and distributed systems.

An exemple of reactive and distributed system

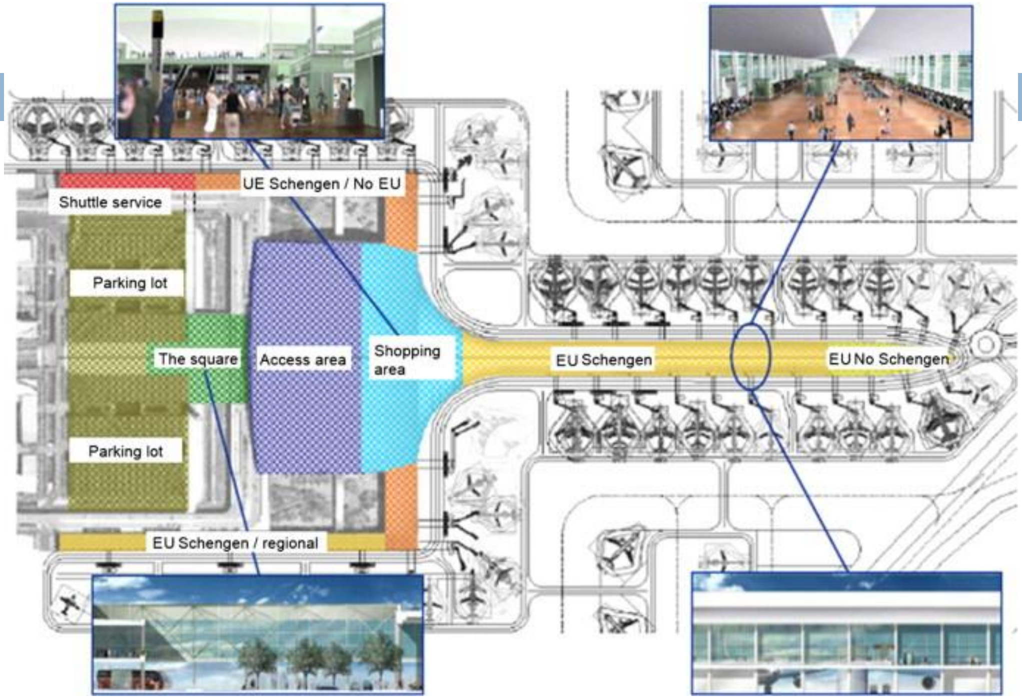


Fig. 8 The new T1 building areas related to the different passenger typologies in one of the proposed configurations.

P. Fonseca i Casas , J. Casanovas , X. Ferran

Passenger flow simulation in a hub airport: An application to the Barcelona International Airport

Simulation Modelling Practice and Theory, Volume 44, 2014, 78 - 94

http://dx.doi.org/10.1016/j.simpat.2014.03.008

An example of reactive and distributed system

The new terminal of the Barcelona International Airport is a reactive and distributed system.

Why SDL exists ?

- The initial purpose of SDL is to be a language for <u>unambiguous</u> specification and description of the <u>structure</u>, <u>behavior</u> and <u>data</u> 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 behavior
  - a description of a system is the description of its actual behavior, that is its implementation

Standarization

- The three largest and most well-established such organizations:
  - International Organization for Standardization (ISO), founded in 1947
  - International Electrotechnical Commission (IEC), founded in 1906
  - International Telecommunication Union (ITU), founded in 1865
- All based in Geneva, Switzerland.
- These three organizations together comprise the World Standards Cooperation (WSC) alliance.

SDL

- O.O Language.
- Defined by the International Telecommunications Union–Telecommunications Standardization Sector (ITU–T) (formerly Comité Consultatif International Telegraphique et Telephonique [CCITT]) as recommendation Z.100.

Series	Description
A	Organization of the work of ITU-T
B	Means of expression: definitions, symbols, classification
C	General telecommunication statistics
D	General tariff principles
E	Overall network operation, telephone service, service operation and human factors
F	Non-telephone telecommunication services
G	Transmission systems and media, digital systems and networks
H	Audiovisual and multimedia systems
I	Integrated services digital network
J	Cable networks and transmission of television, sound programme and other multimedia signals
K	Protection against interference
L	Construction, installation and protection of cables and other elements of outside plant
M	Telecommunication management, including TMN and network maintenance
N	Maintenance: international sound programme and television transmission circuits
O	Specifications of measuring equipment
P	Telephone transmission quality, telephone installations, local line networks
Q	Switching and signalling
R	Telegraph transmission
S	Telegraph services terminal equipment
T	Terminals for telematic services
U	Telegraph switching
V	Data communication over the telephone network
X	Data networks, open system communications and security
Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Z	Languages and general software aspects for telecommunication systems

SDL ITU Recommendations

- The ITU-T Specification and Description Language (SDL) is defined by the following ITU-T Recommendation publications
  - Z.100 (11/99) Specification and description language (SDL) including various annexes and appendices
  - Z.105 (11/99) SDL combined with ASN.1modules;
  - Z.107 (11/99) SDL with embedded ASN.1;
  - Z.109 (11/99) SDL combined with UML.

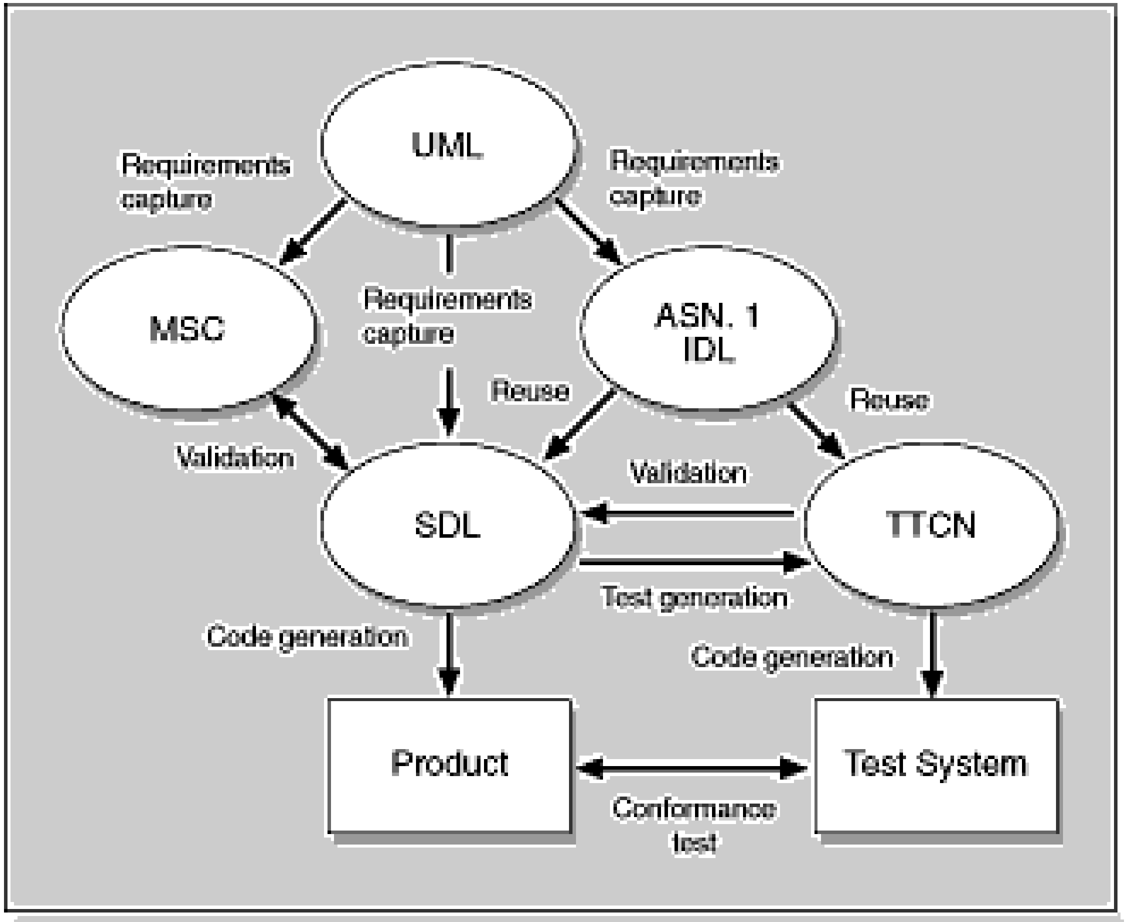
Where SDL may be used ?

- 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,
    - 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 tools (some)

- PragmaDev Real Time Developer Studio (*COMMERCIAL*)
- SDL Suite by IBM (acquired from Telelogic) an SDL Design Tool (*COMMERCIAL*)
- Cinderella SDL Design Tool (*COMMERCIAL*)
- SanDriLa SDL Design Tool (*COMMERCIAL*)
- SAFIRE Integrated Development & Run-Time Environment (*COMMERCIAL*)
- SDL tool from Humboldt University of Berlin
- OpenGEODE, a free and open-source SDL editor from ESA
- PlantUML beta release includes support for a subset of the SDL

SDL



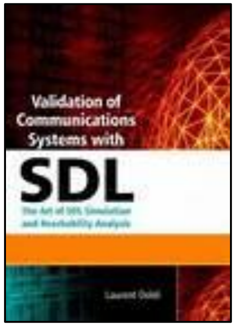
UML
Requirements capture
Requirements capture
MSC
Requirements capture
ASN. 1 IDL
Reuse
Reuse
Validation
SDL
Validation
TTCN
Test generation
Code generation
Code generation
Product
Test System
Conformance test

SDL History (1)

- 1976 Orange Book SDL
  - Basic graphical language
- 1980 Yellow Book SDL
  - Process semantics defined
- 1984 Red Book SDL
  - Structure, data added.
  - Definition more rigorous.
  - Start of tools. User guide.
- 1988 Blue Book SDL (SDL-88)
  - Effective tools.
  - Syntax well defined - formal definition.
  - Language much as 1984.

SDL History (2)

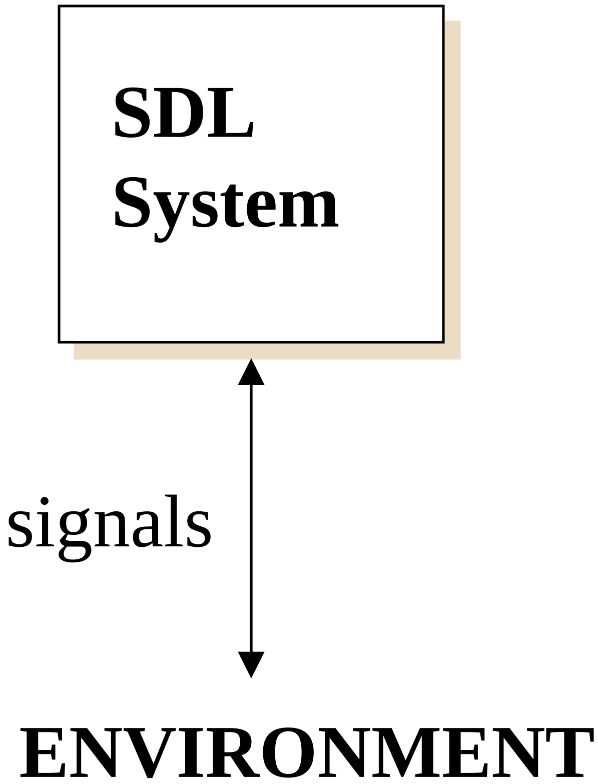
- 1992 White Book SDL-92
  - Object SDL. Types for blocks, processes, services with inheritance and parameterisation.
  - Methodology guidelines.
- 1995 SDL with ASN.1 (Z.105)
- 1996 Addendum 1 to SDL-92
  - Language stable. Some relaxation of rules.
  - SDL+ Methodology.
  - Tools offer SDL-92 features.
- 1999 SDL-2000
  - Object modeling support.
  - Improved implementation support.
  - Data model revised
- 2012 SDL-2010



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

System & 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

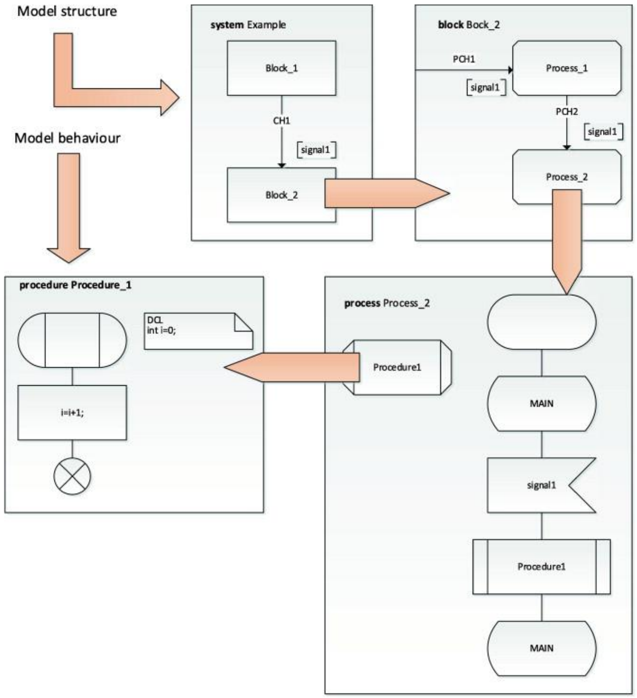


Fig. 1 SDL levels.

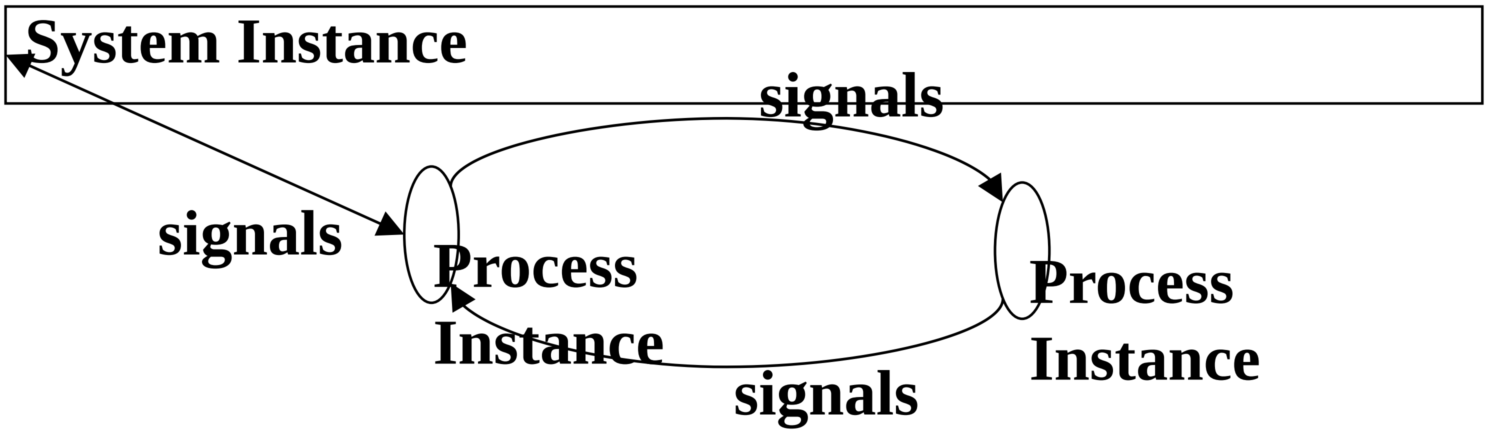
P. Fonseca i Casas , J. Casanovas , X. Ferran

Passenger flow simulation in a hub airport: An application to the Barcelona International Airport

Simulation Modelling Practice and Theory, Volume 44, 2014, 78 - 94

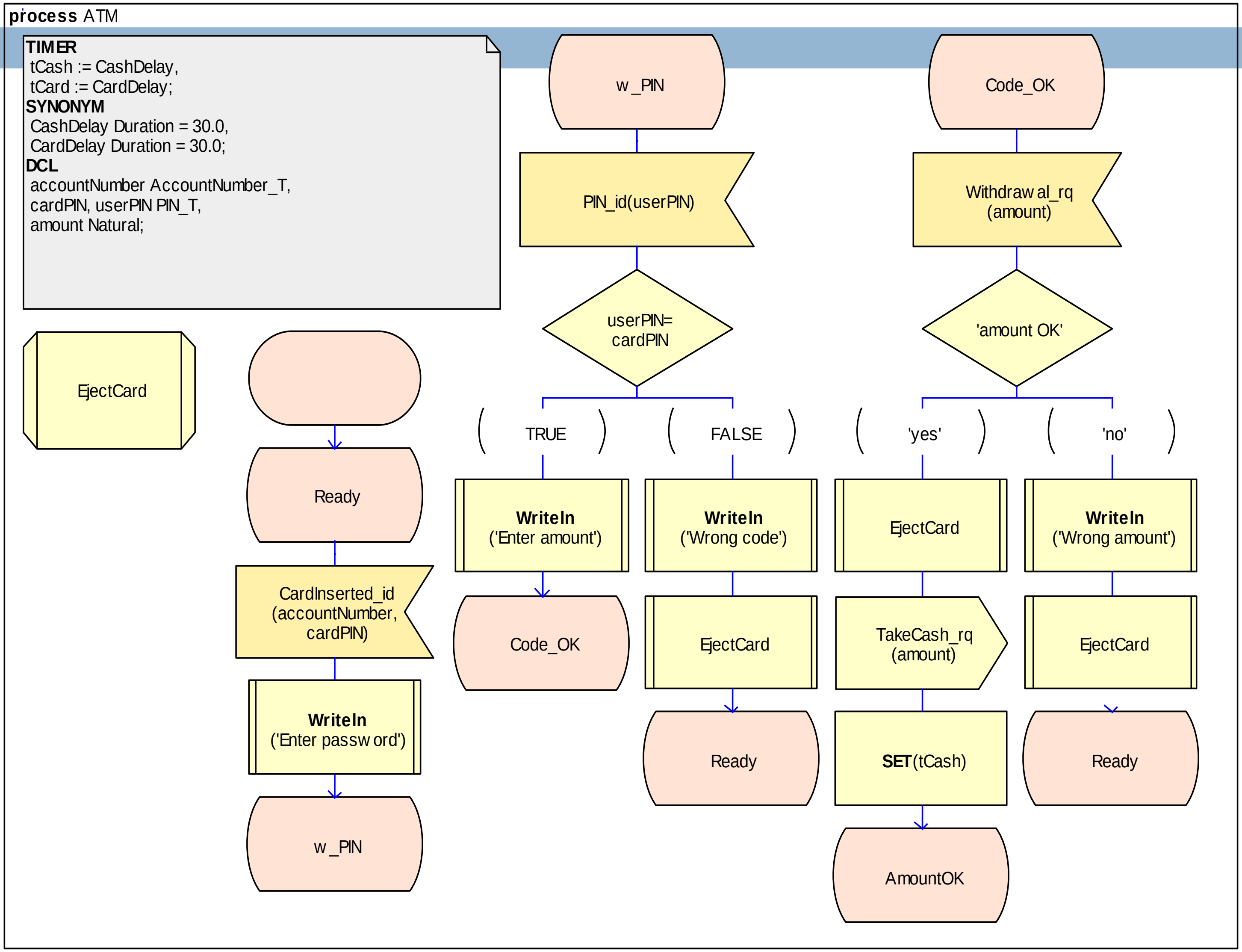
http://dx.doi.org/10.1016/j.simpat.2014.03.008

SDL Overview - Process

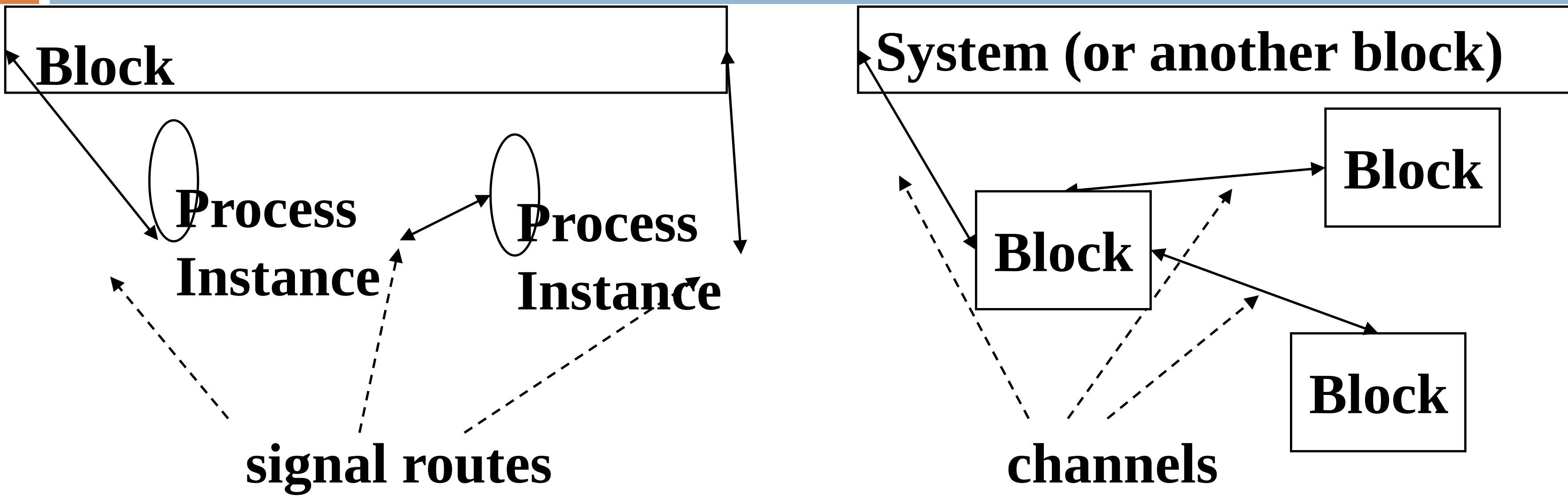


- A <u>process</u> is an agent that contains an extended finite state machine, and may contain other processes.
- A System is composed of a number of communicating process instances

SDL Overview - Process Diagrams



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

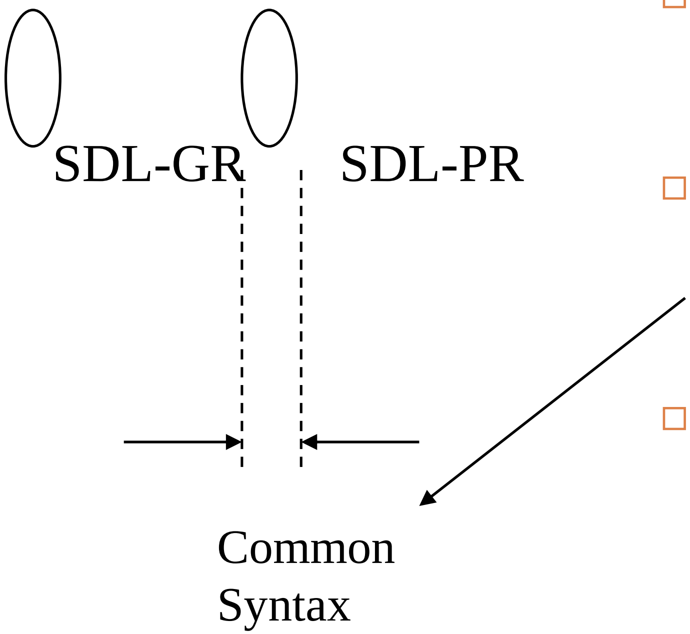
Key SDL Features (1 of 2)

- 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 behavior of the system is characterized by the communication on external interfaces.
- Types
  - Classes can 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.

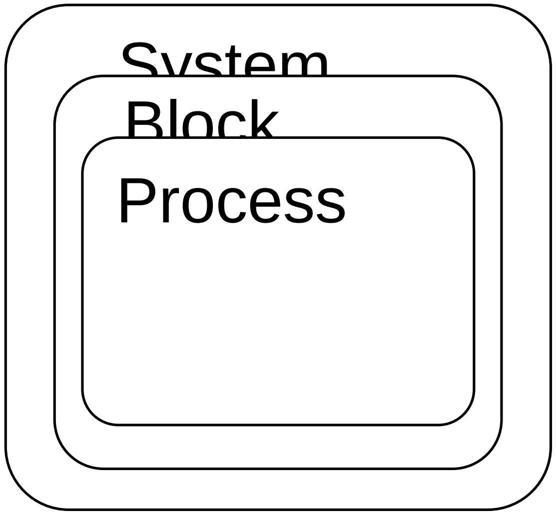
SDL Representations



- SDL has two representation forms
  - SDL-GR - graphical representation
  - SDL-PR - textual, phrase representation
- SDL-PR is conceived as for easily processed by computers - common interchange format (CIF)
- SDL-GR is used as a human interface
  - SDL-GR has some textual elements which are identical to SDL-PR (this is to allow specification of data and signals)
- Z.106 recommendation defines CIF with purpose of preserving all graphical information

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

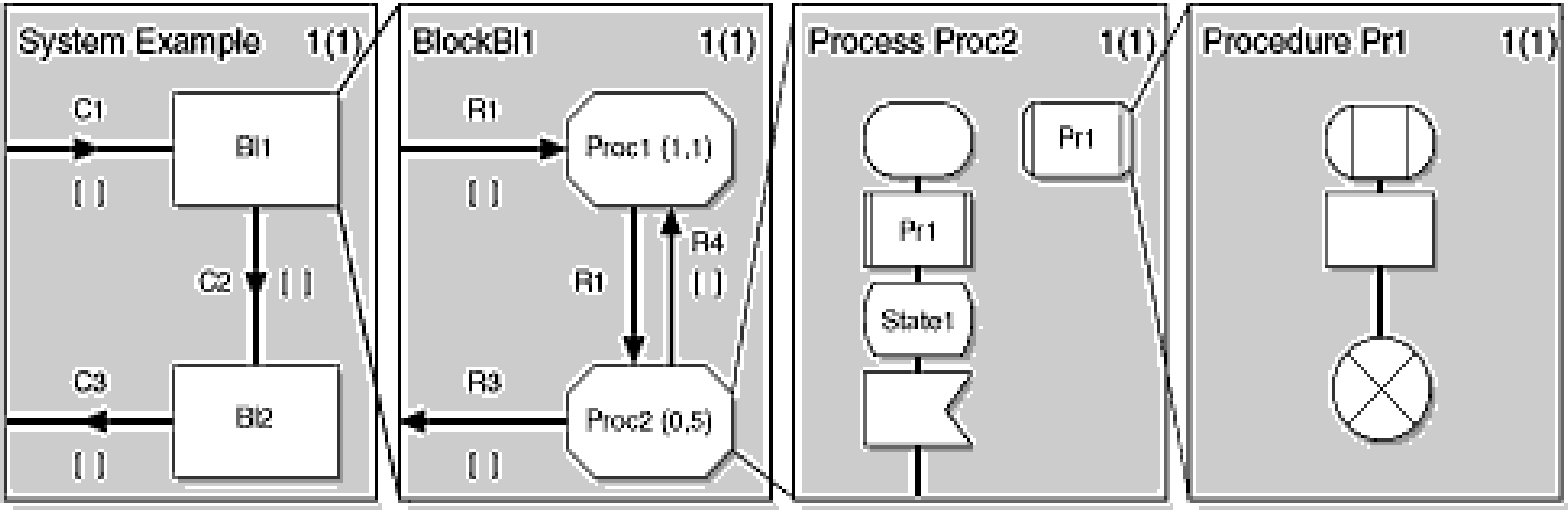


Static SDL Terms

- 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 or processes and may also contain an extended finite state machine that owns and handles data within the block
- process:
  - a process is an agent that contains an extended finite state machine, and may contain other processes
- Procedure
  - A procedure is a piece of programming code.

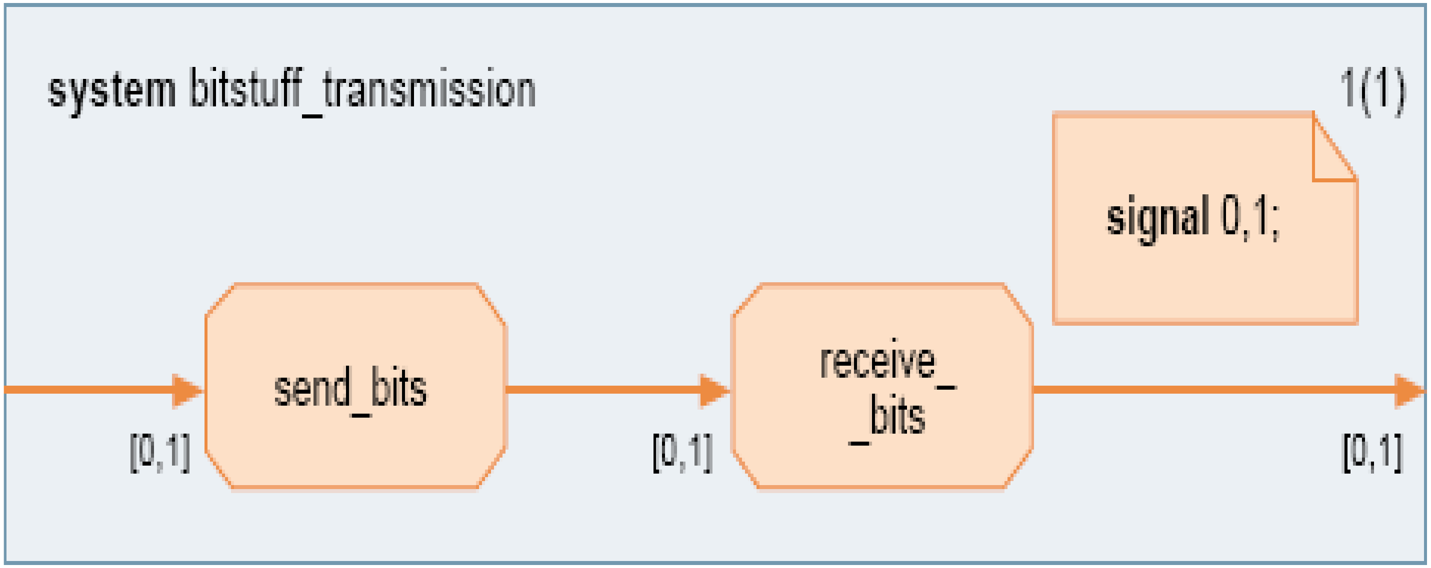
Static SDL Terms

- Source:
  - http://www.iec.org/online/tutorials/sdl/topic04.html



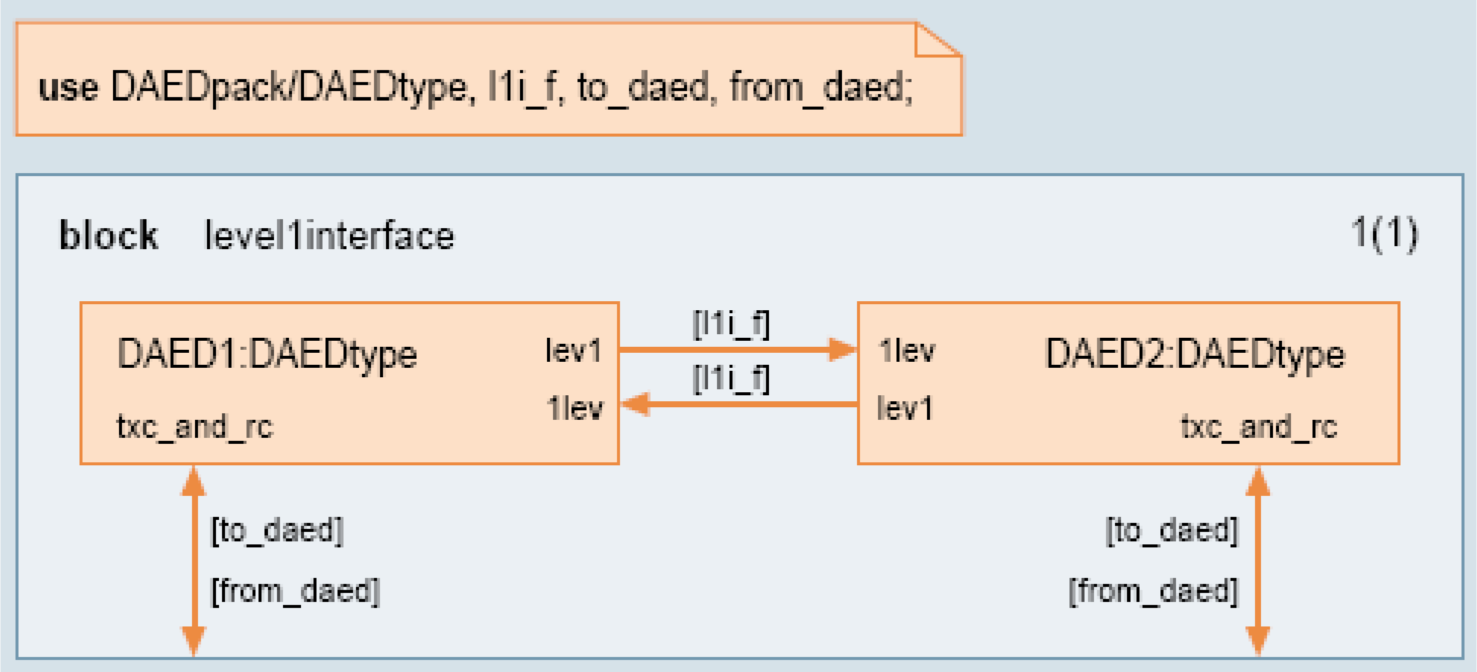
System diagram

- Source (Reed 2000).



SDL Blocks diagram

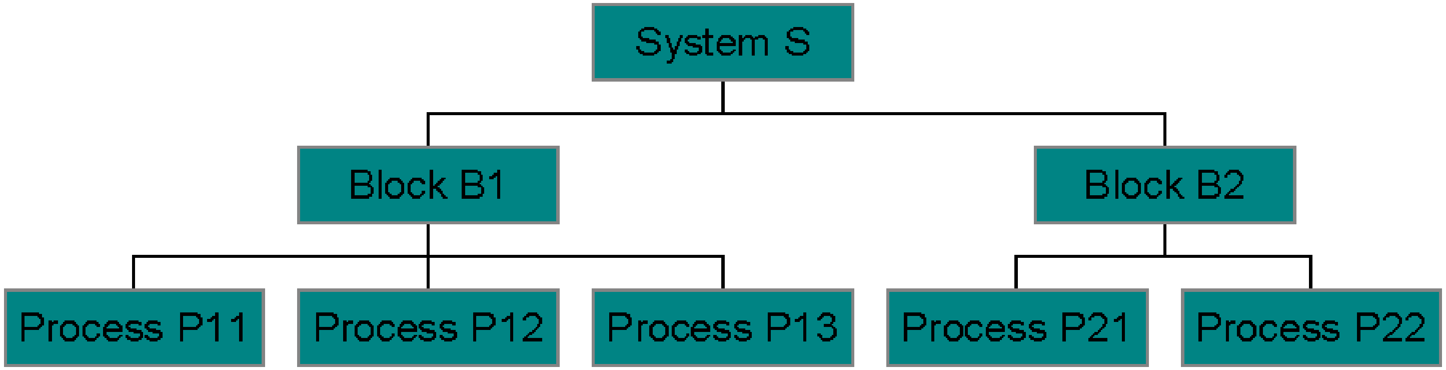
- Source (Reed 2000).



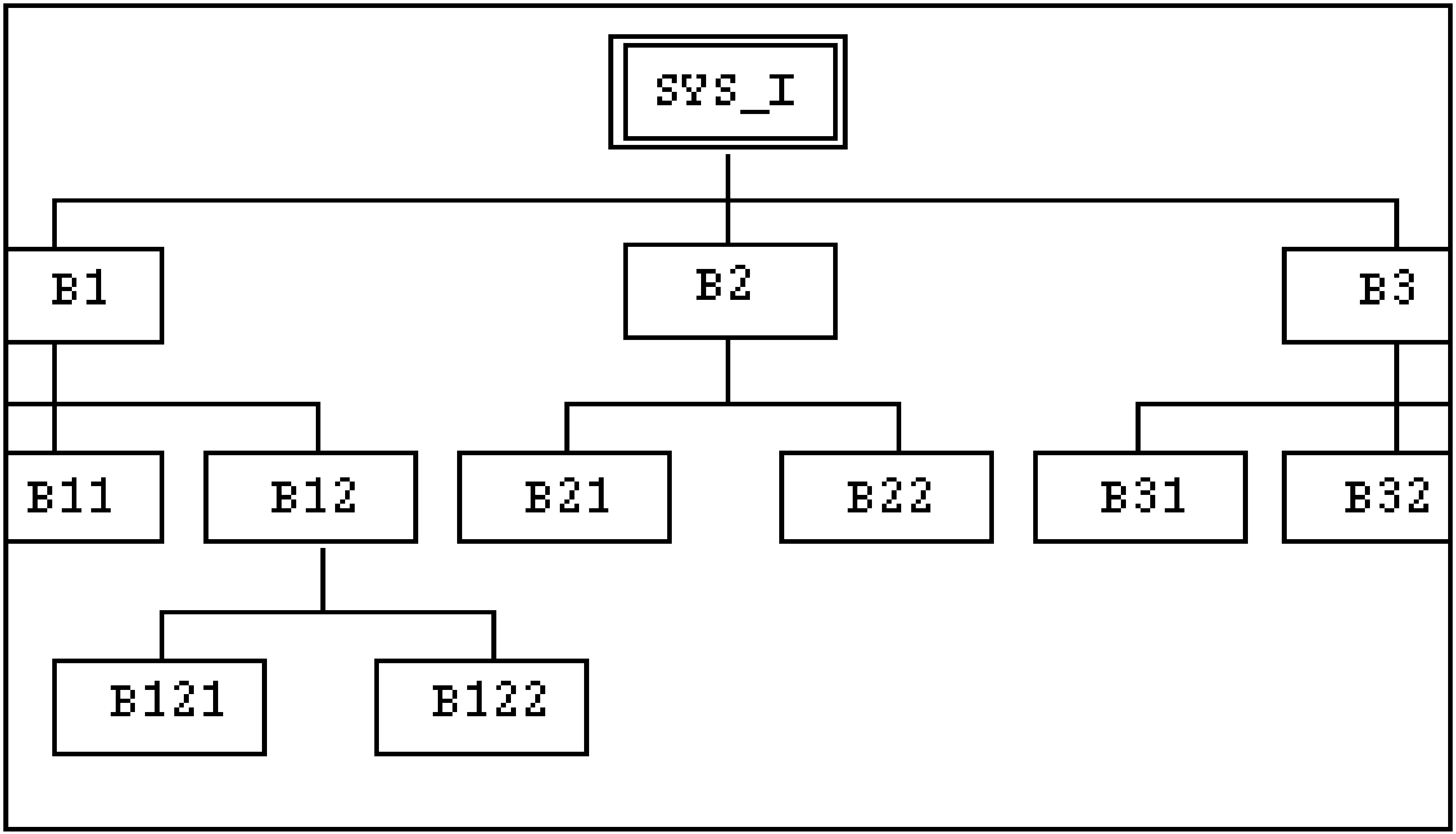
System Decomposition

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

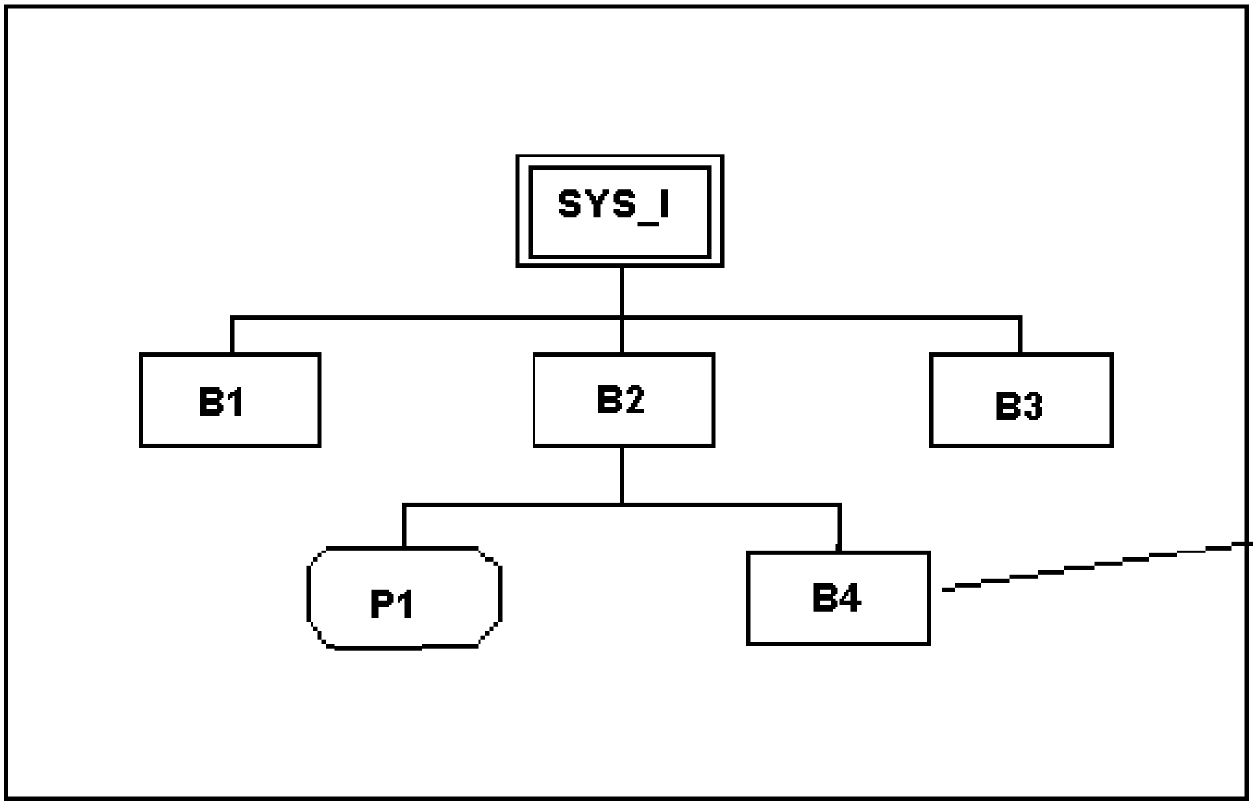
Structuring of the System Description



Decomposition Rules:
No Limit in number of Block levels



Decomposition Rules: Blocks and Process cannot share a level



A Block cannot be decomposed into a Process and anohter Block

Communication Related SDL Terms

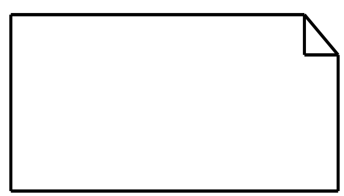
- signal:
  - The primary means of communication is by signals that are output by the sending agent and input by the receiving agent.
- stimulus:
  - A stimulus is an event that can cause an agent that is in a state to enter a transition.
- channel:
  - A channel is a communication path between agents.

Text Symbol

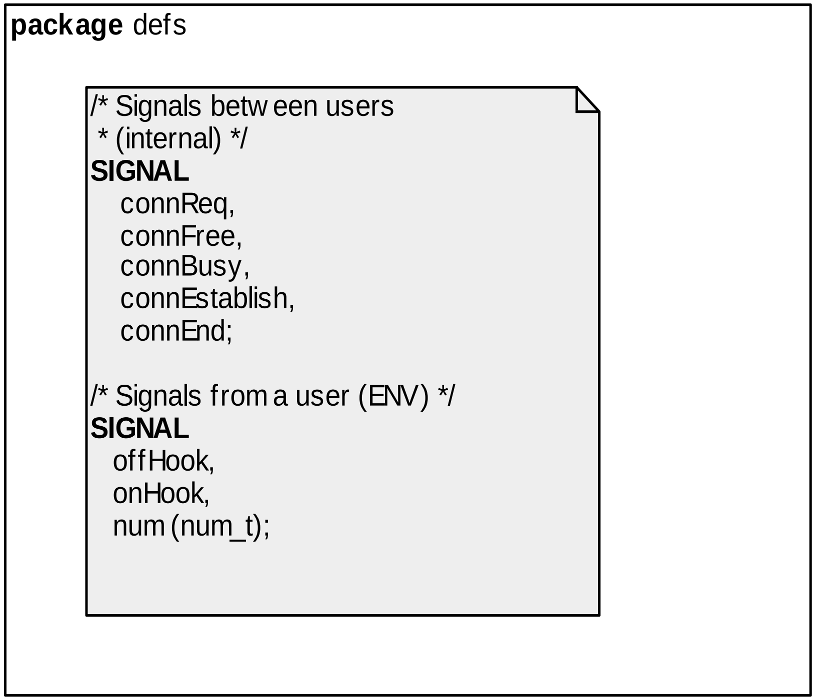
- Text Symbol is used to group various textual declarations
- It can be located on any type of diagram

Concrete graphical grammar

<text symbol> ::=



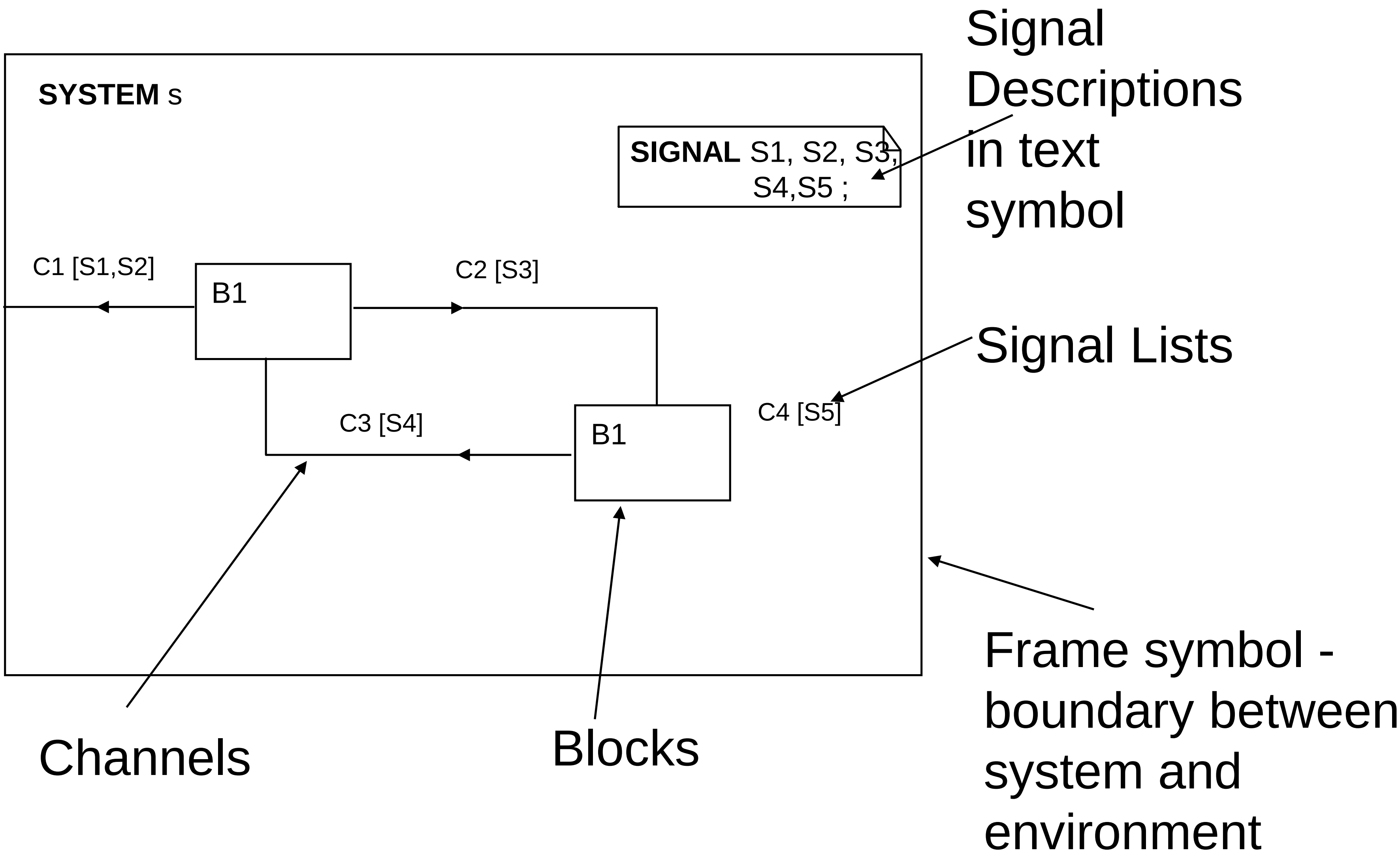
Text Box Example



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



Signals

- Signals are the actual messages sent between entities
- Signals must be defined before they can be used:

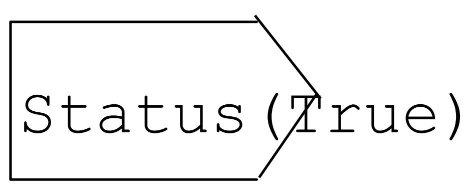
  <signal specification> ::= signal <signal name> [(<sort name>{,<sort name>}*)]
  {, <signal name> [(<sort name>{,<sort name>}*)]}*;

Example:

```
SIGNAL
  doc (CHARSTRING), conf,
  ind (MsgTyp), req (MsgTyp);
```

Signals with parameters

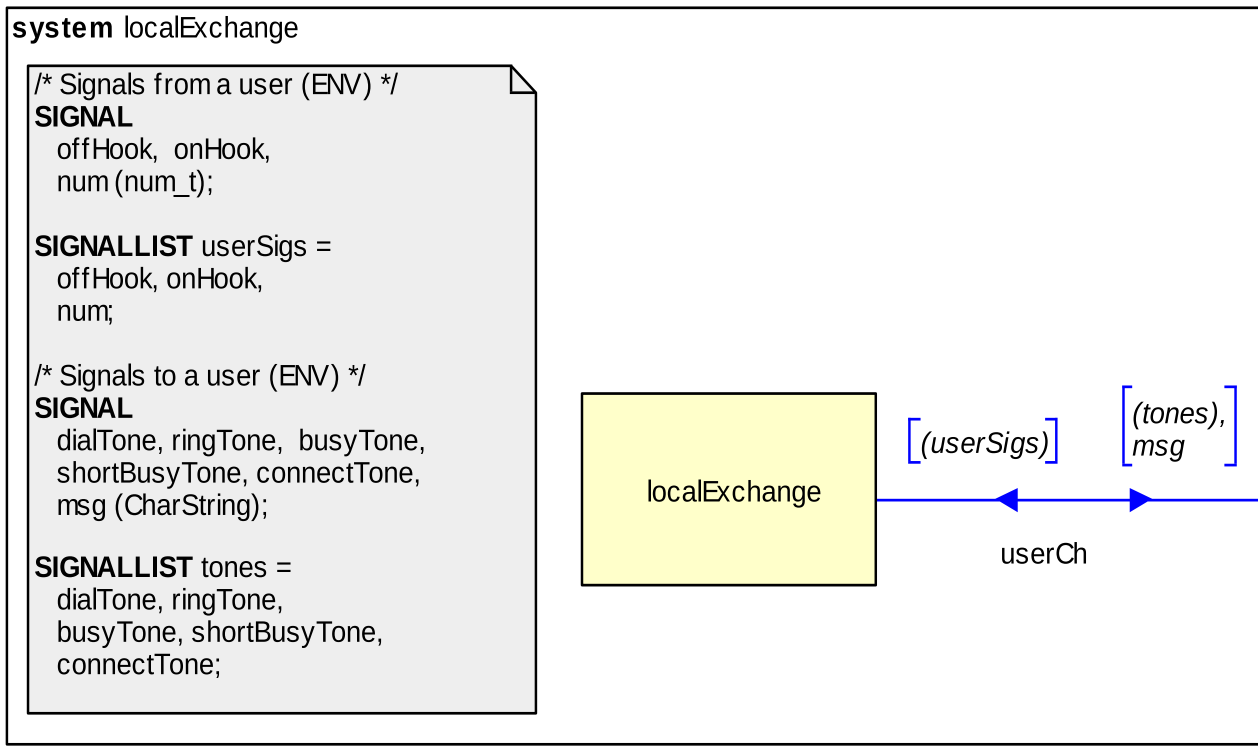
- Signals can have parameters known as a sortlist
- The signal specification identifies the name of the signal type and the sorts of the parameters to be caried by the signal
  - Example: `signal Status(Boolean);`
- When signals are specified to be carried on certain channels only signal names are required
- When signals are actually generated in the process the actual parameters must be given
  - Example:



Signal Lists

- A <u>signal lists</u> may be used as shorthand for a list of signal identifiers

Example:

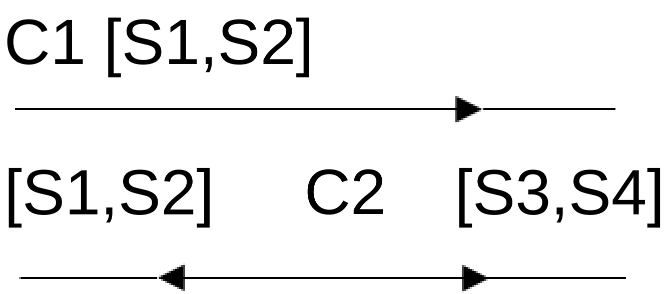


Channel

- CHANNEL is connected between Blocks or Block and the Environment.
- May be uni- or bi-directional
- It may have an identifier (C1) and may have list of all signals it caries
- It is an FIFO queue which may introduce an variable delay

Delaying Channels

- Delaying channels introduce a delay in transmission of signals.
- Delaying channel is specified by a channel symbol with the arrows at the middle of the channel.
- The delay of signals is non-deterministic, but the order of signals is maintained.



Uni-directional delaying Channel

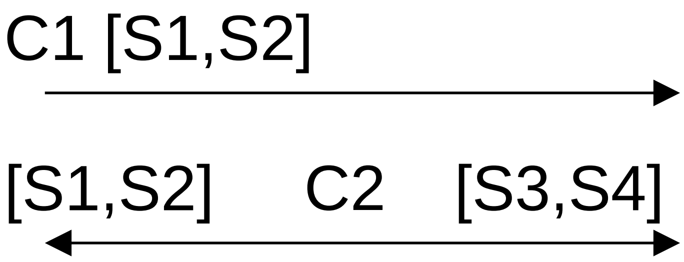
Bi-directional delaying Channel

Signal Route

- SIGNALROUTE: provide a signal path between processes
  - similar to CHANNELs except there is no delay involved
- Can be bi-directional or unidirectional
- Contains a signal list, constraining what signals can sent through it.
- In SDL2000 Signal-Route concept is obsolete. Signal Routes are replaced by non-delaying Channels

Non-Delaying Channels

- Non delaying channels do not introduce any delay in transmission of signals



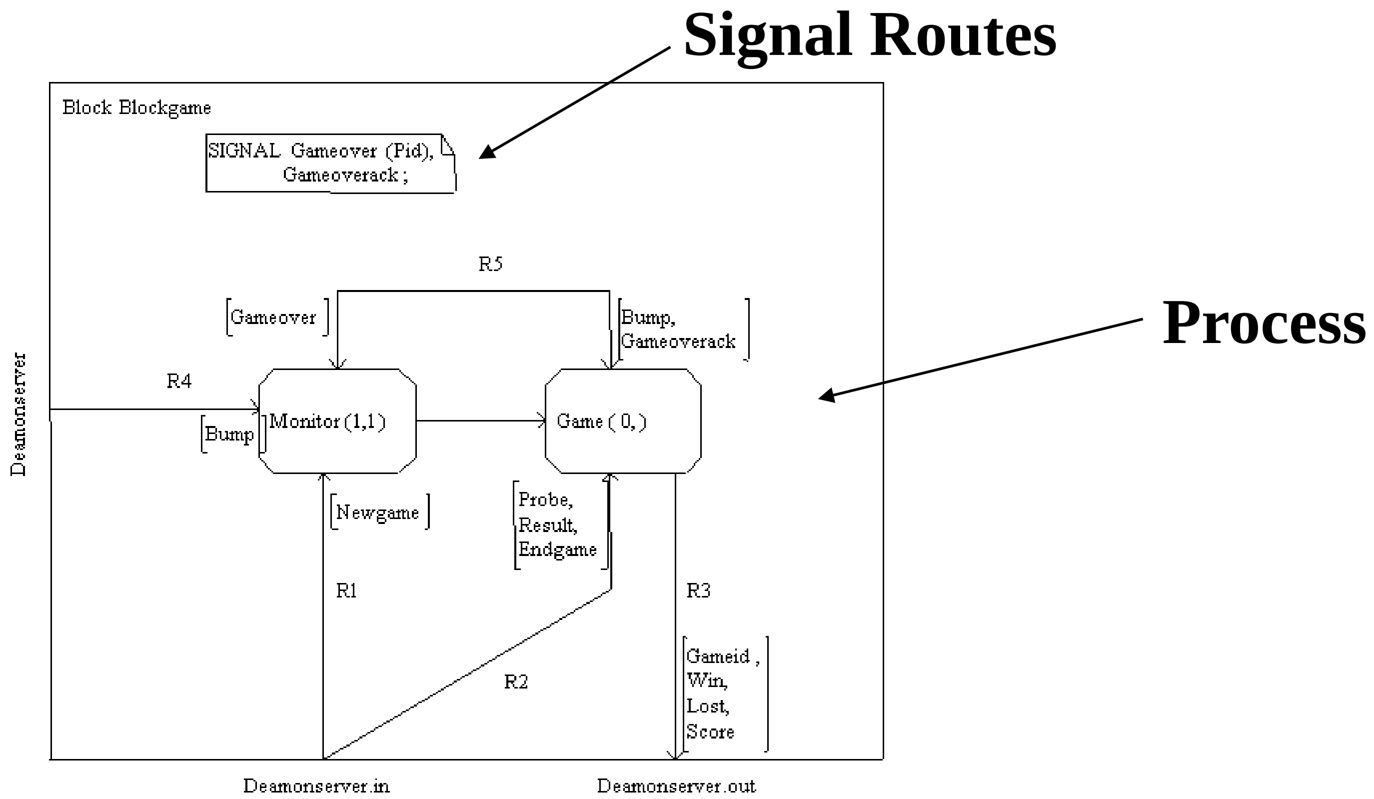
Uni-directional non-delaying Channel

Bi-directional non-delaying Channel

Block Diagram

- Has a name specified by BLOCK keyword
- Contains a number of Processes
- May also possibly contain other BLOCKs (but not mixed with Processes)
- Processes communicate via Signal Routes, which connect to other Processes or to Channels external to the Block
- Textual Descriptions/Definitions
  - Signal Descriptions for signals local to the BLOCK
  - Signal Route Descriptions
  - Data Type Descriptions
  - Process Descriptions

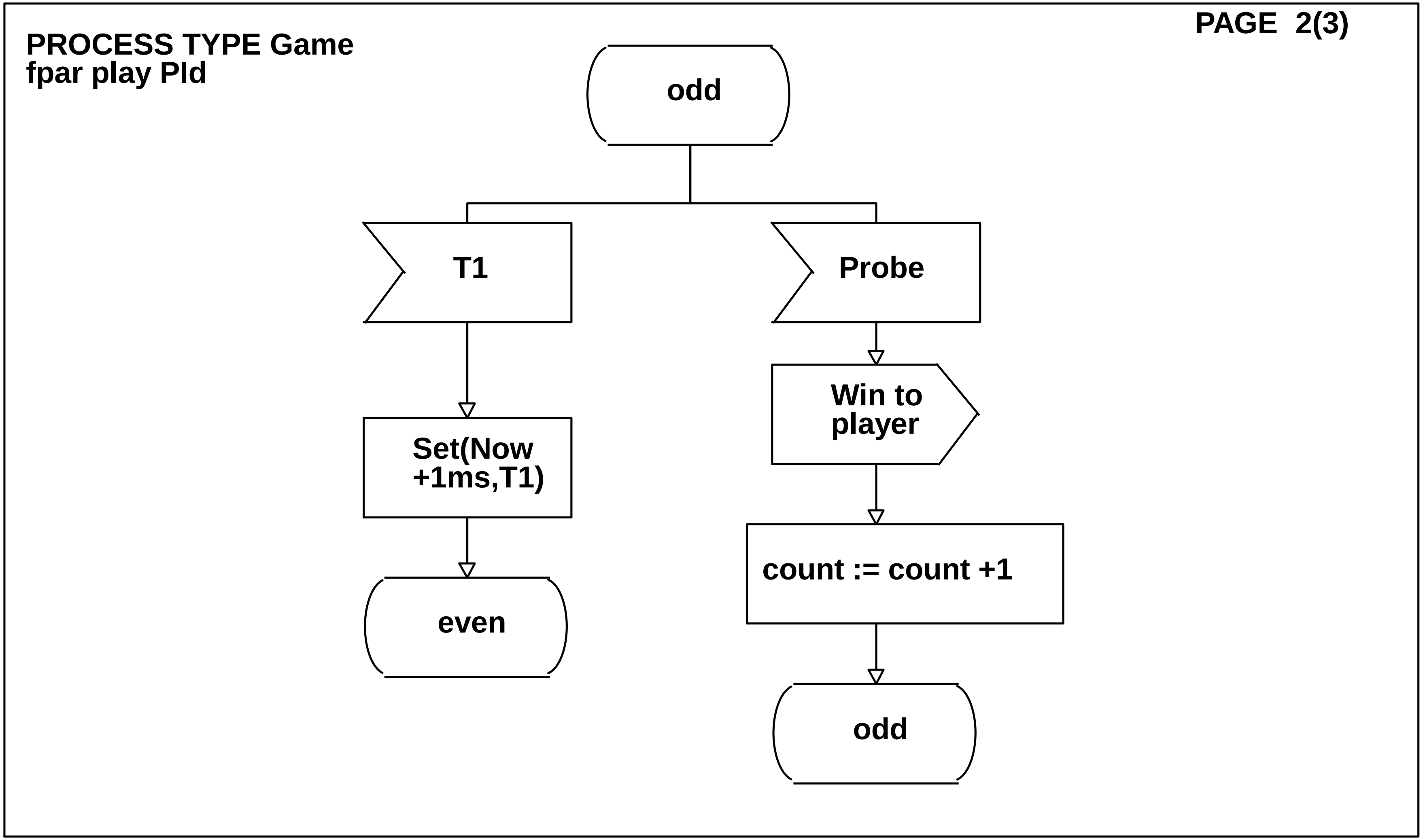
Example Block Diagram



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

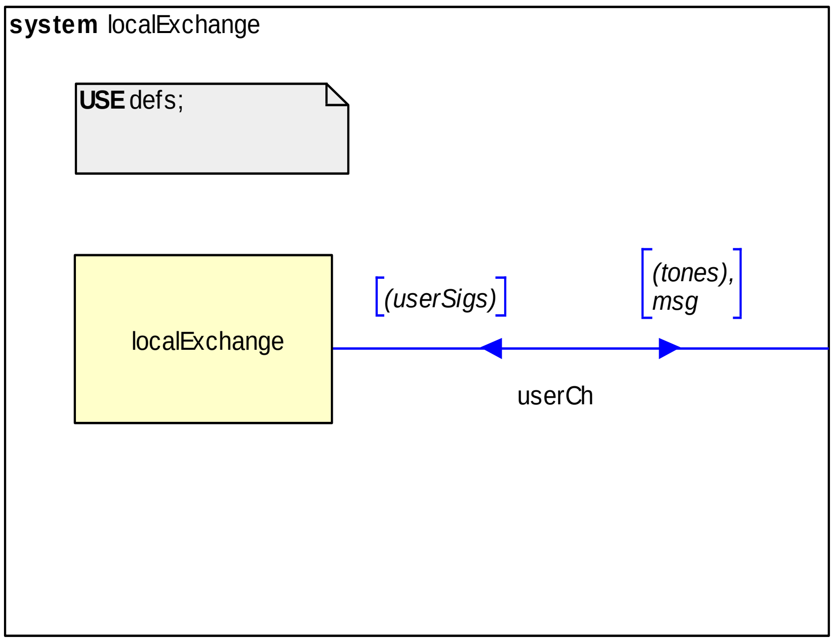
Example Process Diagram

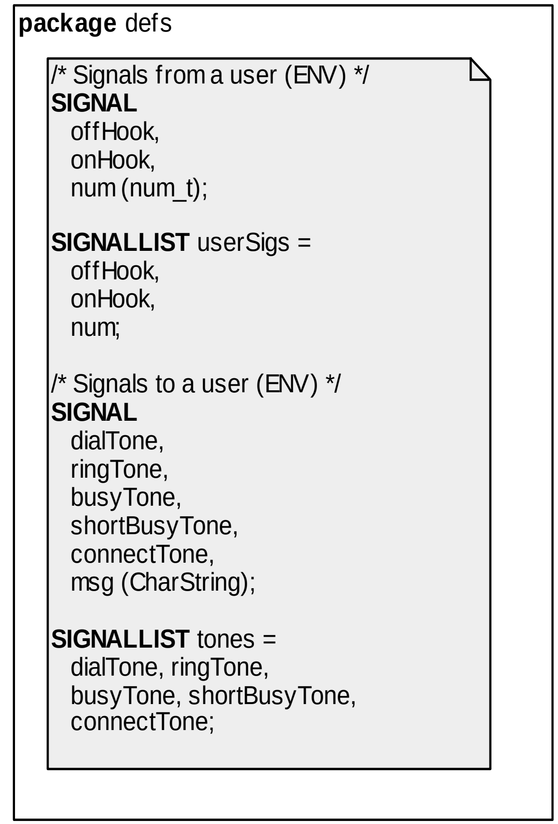


Packages & Libraries

- 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.

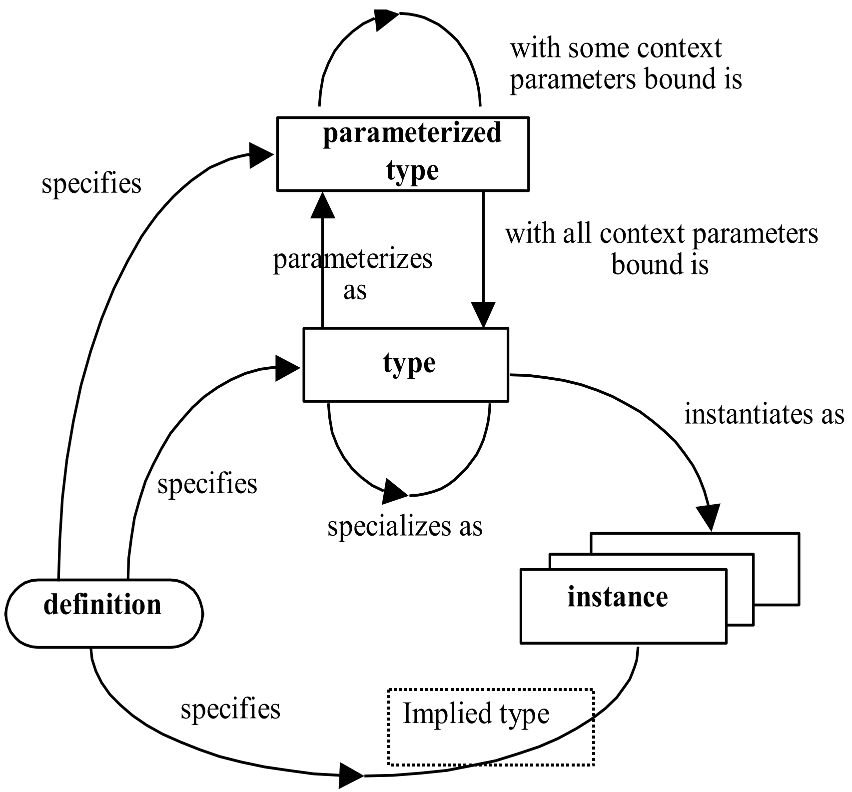
Package Example





Definition, Type & Instance

- <u>Definitions</u> introduce named entities, which are either types or instances with implied types. A definition of a <u>type</u> defines all properties associated with that type.
- A <u>type</u> may be instantiated in any number of instances. An instance of a particular type has all the properties defined for that type.
- An <u>instance</u> is defined either directly or by the instantiation of a type. An example of an instance is a system instance, which can be defined by a system definition, or is an instantiation of a system type.



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

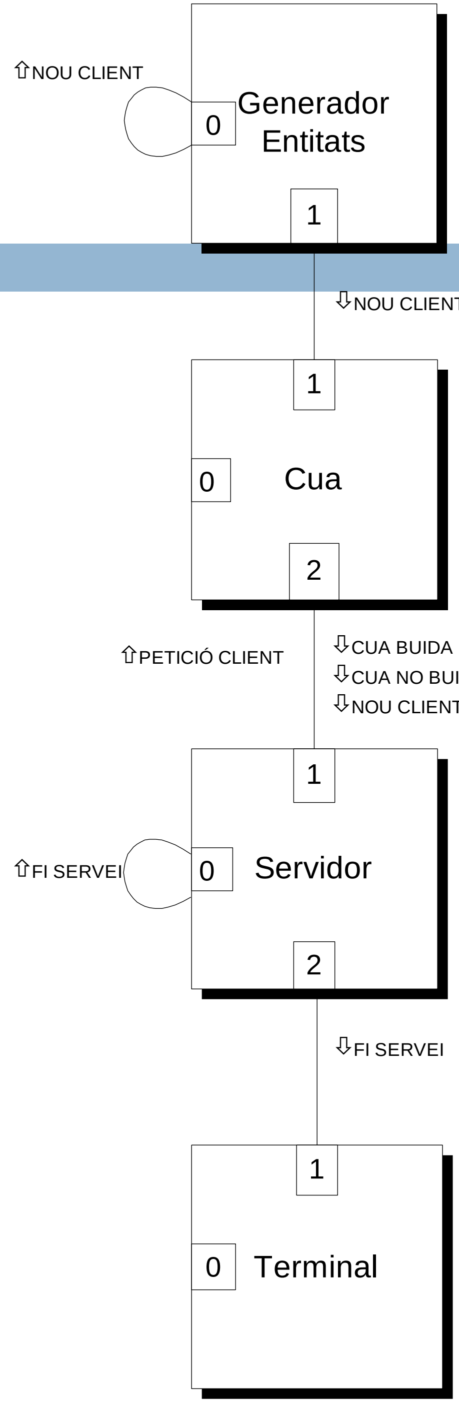
Examples

MM1

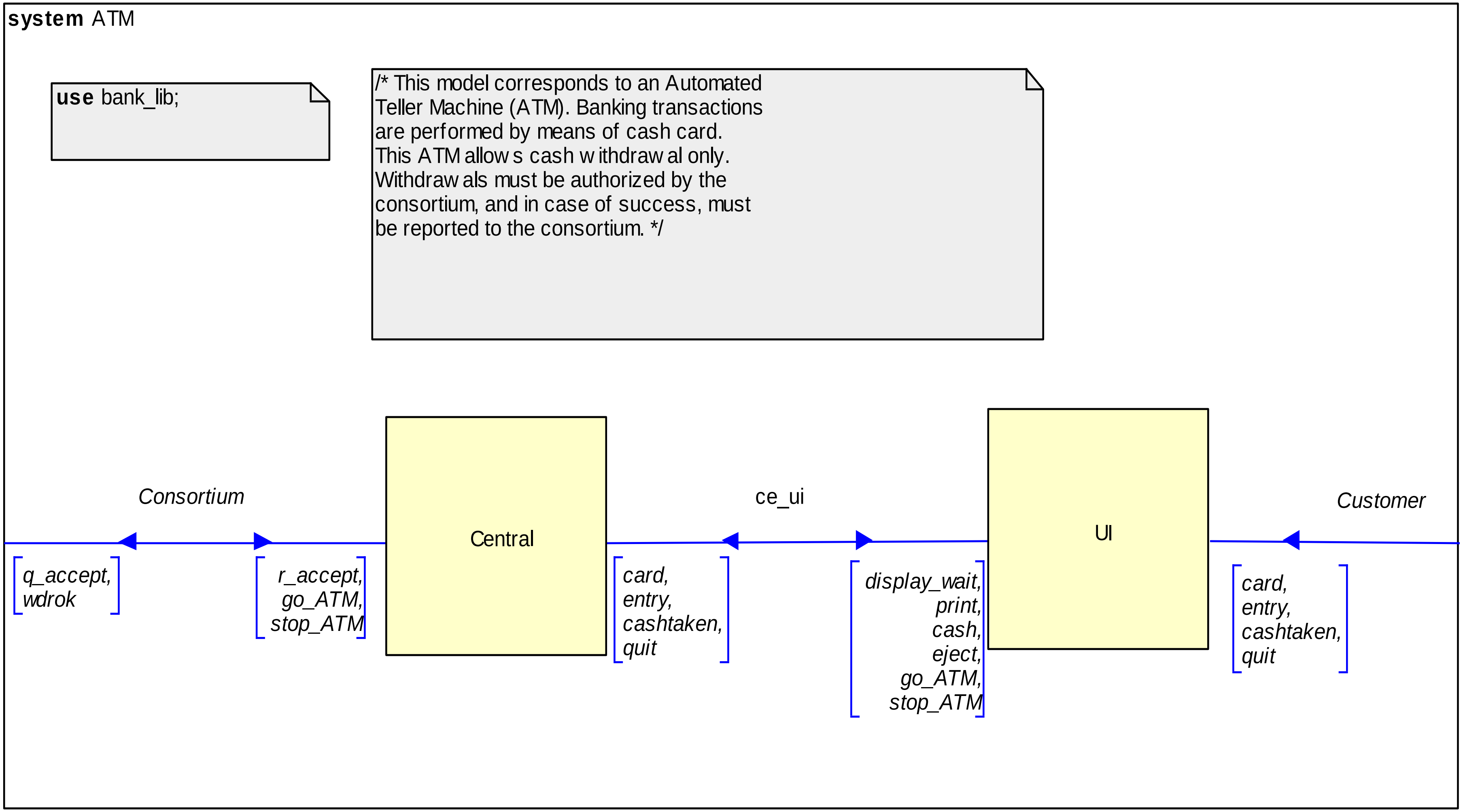
ATM

MM1 Example

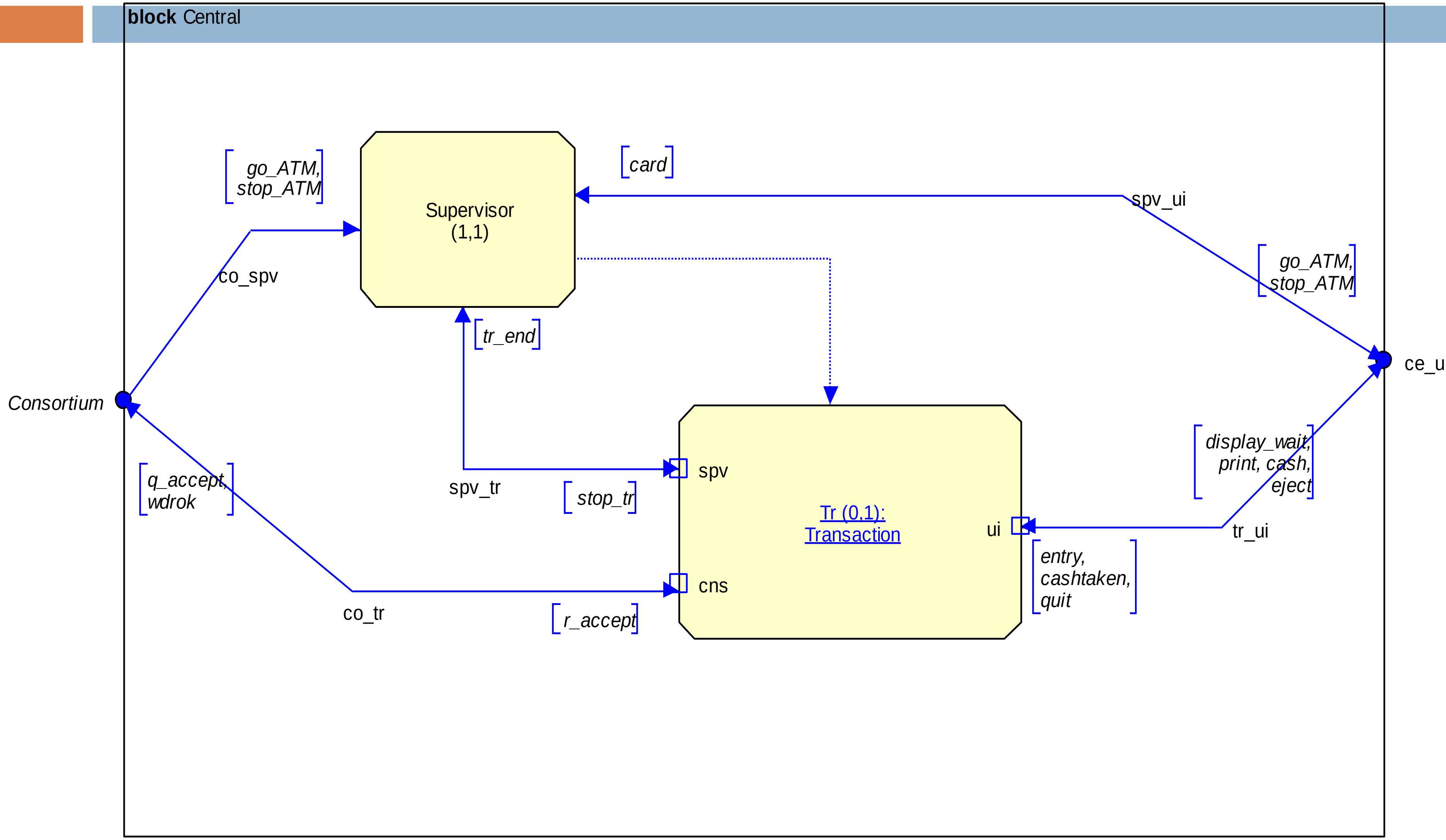
- 4 Elements
  - Generator
  - Queue
  - Server
  - Terminate



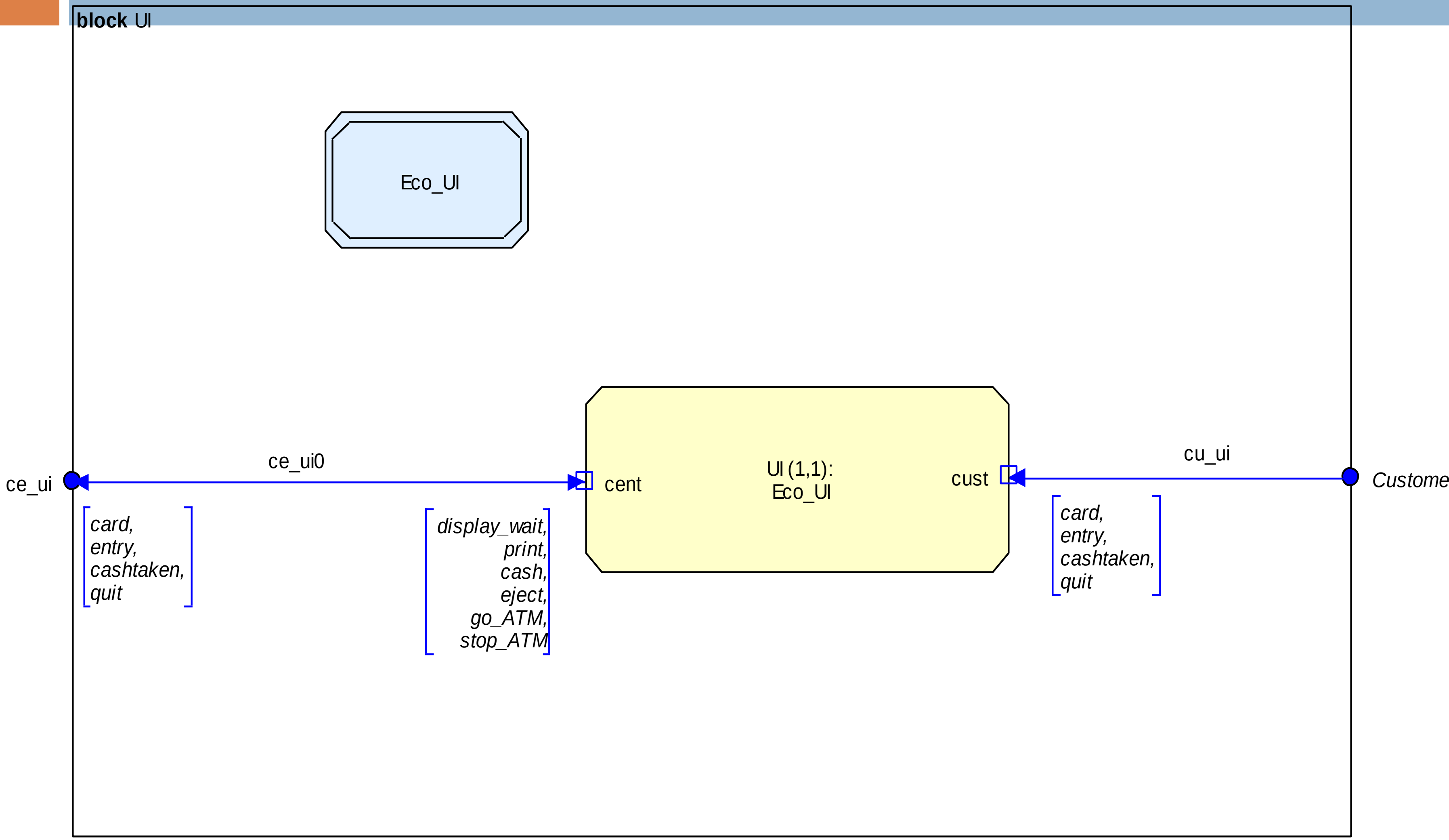
ATM Example - System Diagram



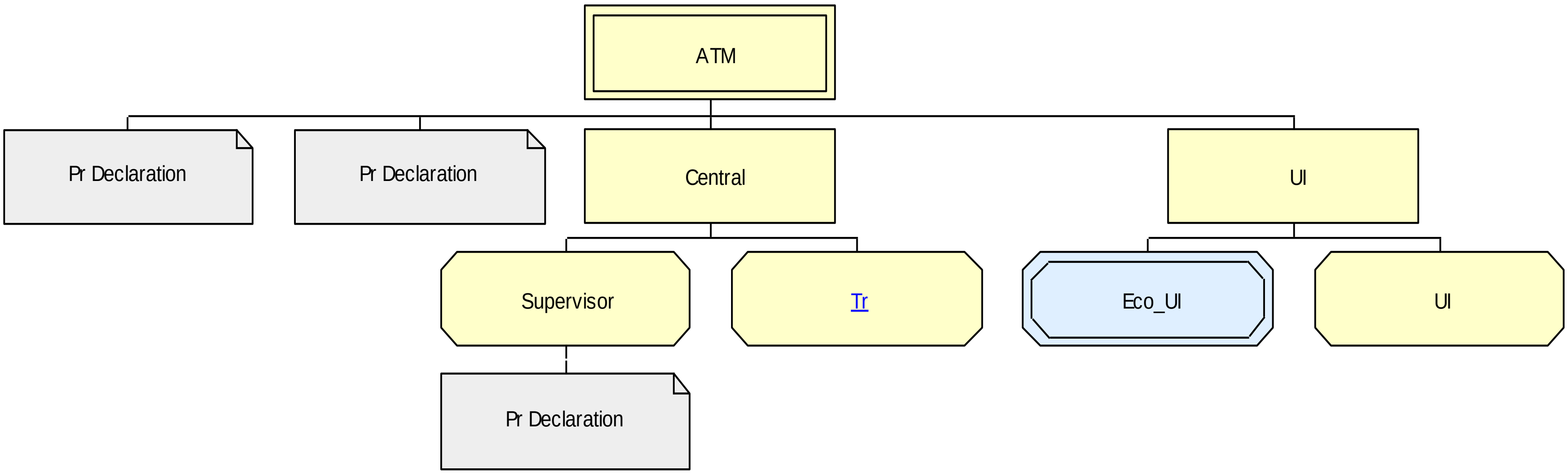
ATM Example - Central Block Diagram



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;
  psswd Charstring;
operators
  checkCard: CashCard -> Boolean;
  checkPsswd: CashCard, Charstring -> Boolean;
operator checkCard;
  fpar cc CashCard;
  returns res Boolean;
  start;
  task res := (cc!expirDate > 9701) and (cc!id /= 0);
  return;
endoperator;
operator checkPsswd;
  fpar cc CashCard, cpw Charstring;
  returns res Boolean;
  start;
  task res := (cc!psswd = 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. */

Static SDL - Summary

- Structure of the system is hierarchically defined using System, Block and Process diagrams connected via channels (signal routes)
- Channels carry Signals which convey information (stimulus) between agents (Environment, System, Blocks, Processes)
- The ultimate goal of the SDL is to specify overall behavior of the system - but this is not done on the system level
- The system is defined by behavior of its constituent blocks/processes

Specification & Description Language (SDL)

Dynamic SDL

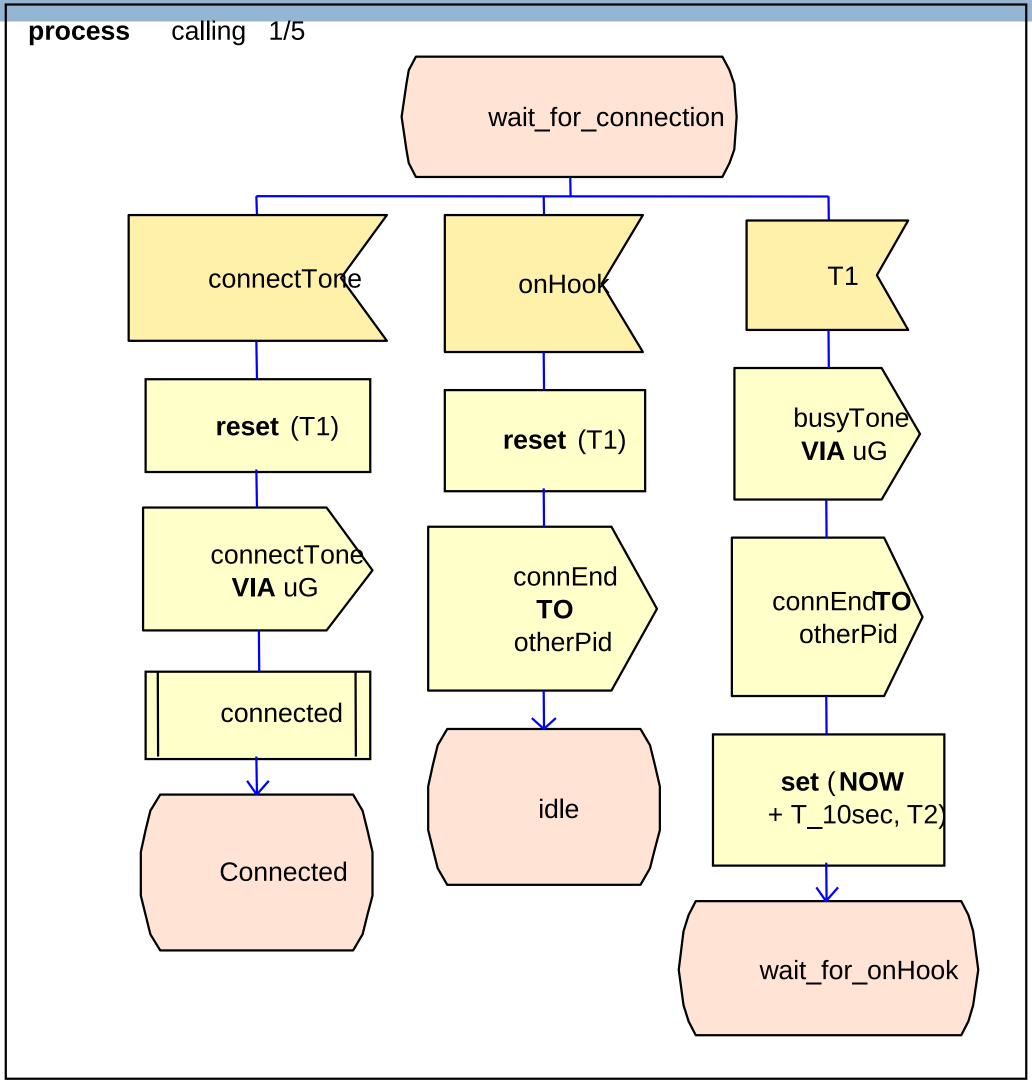
Outline

- Dynamic SDL Component
  - State, Input, Output, Process, Task, Decision, Procedure …
  - Data in SDL
  - Inheritance
  - Block and Process Sets
- Examples

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)

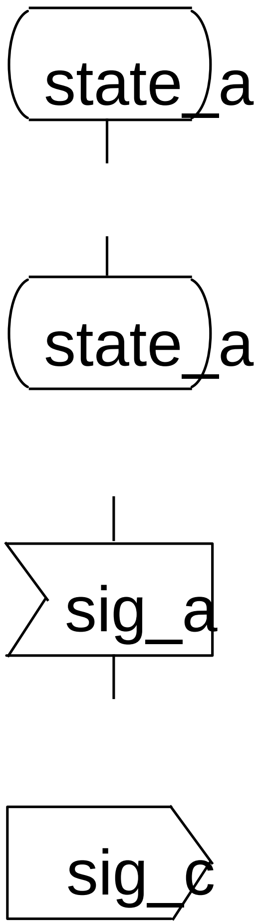
Process Diagram Example



Process diagram

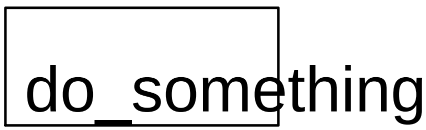
- Describes **for each state of each object** its **behavior** on receiving different events.
- An object can react in a different way receiving the same event, depending on the port used to receive the event.

Process Diagram Components

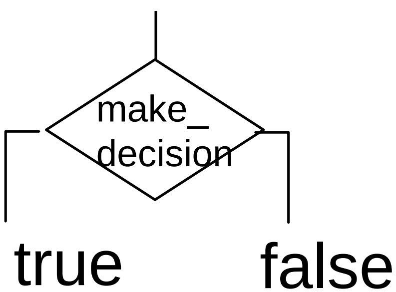


- STATEs: point in PROCESS where input queue is being monitored for arrived SIGNALs
  - subsequent state transition may or may not have a NEXTSTATE
- INPUT: indicates that the subsequent state transition should be executed if the SIGNAL matching the INPUT arrives
  - INPUTs may specify SIGNALs and values within those SIGNALs
  - Inputs can also specify timer expiry
- OUTPUT: specifies the sending of a SIGNAL to another PROCESS

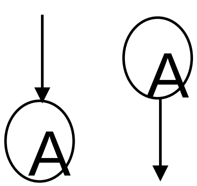
Some Additional Process Diagram Components



- TASK: description of operations on variables or special operations
- The text within the TASK body can contain assign statements.

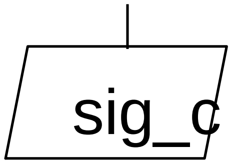


DECISION: tests a condition to determine subsequent PROCESS flow

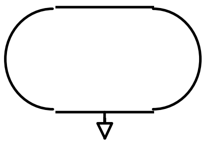


- JOIN: equivalent to GOTO.

More Process Diagram Components …



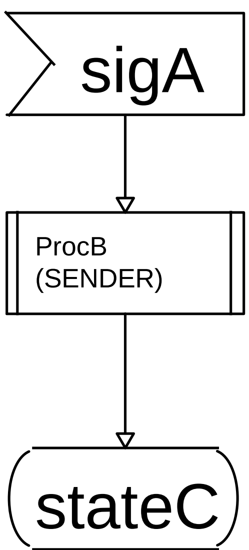
- SAVE: specifies that the consumption of a SIGNAL be delayed until subsequent SIGNALs have been consumed
  - the effect is that the SAVEd SIGNAL is not consumed until the next STATE
  - no transition follows a SAVE
  - the SAVEd SIGNAL is put at the end of the queue and is processed after other SIGNALs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

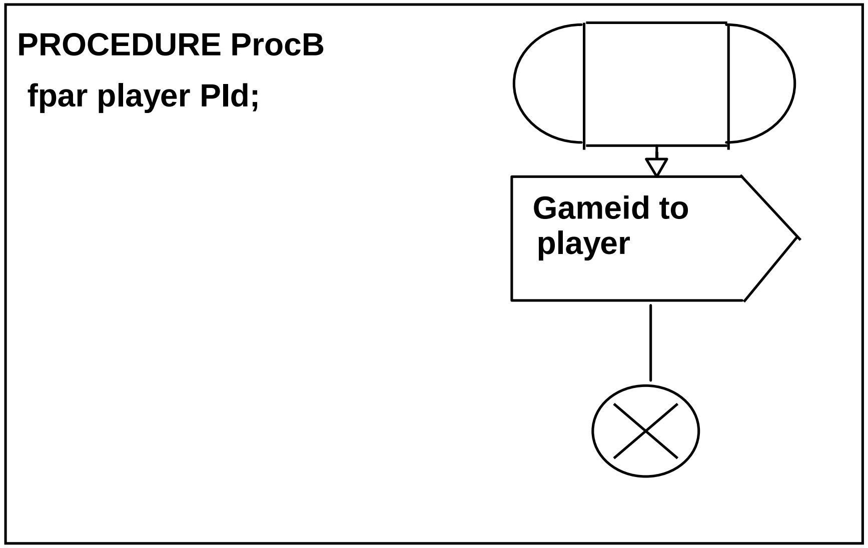


- START: used to describe behavior on creation as well as indicating initial state
  - Similar shape to state only with semi-circular sides

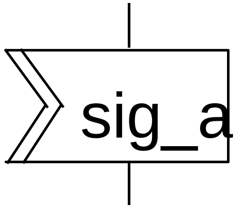
Procedure

- 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)

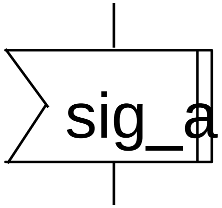




Priority & Internal Inputs

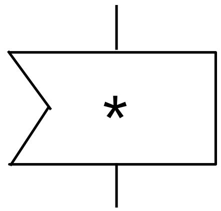


- Priority inputs are inputs that are given priority in a state
- If several signals exist in the input queue for a given state, the signals defined as priority are consumed before others (in order of their arrival)

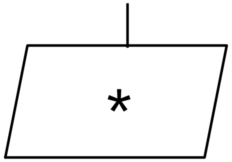


- **Internal Input/Outputs signals are used for signals sent/received within a same FSM or SW component**
- **There is no formal definition when they should be used.**

Shorthands - All Other Input/Save

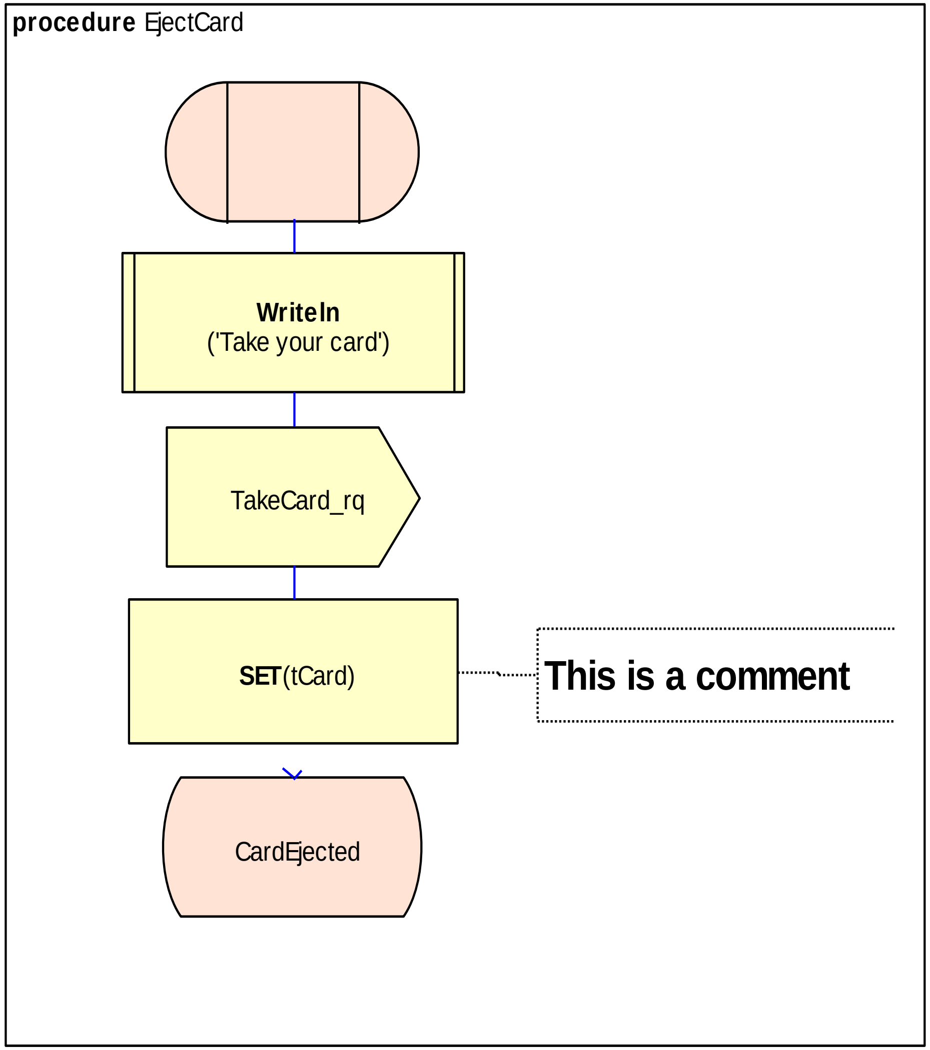


- **The input with an asterisk covers all possible input signals which are not explicitly defined for this state in other input or save constructs**

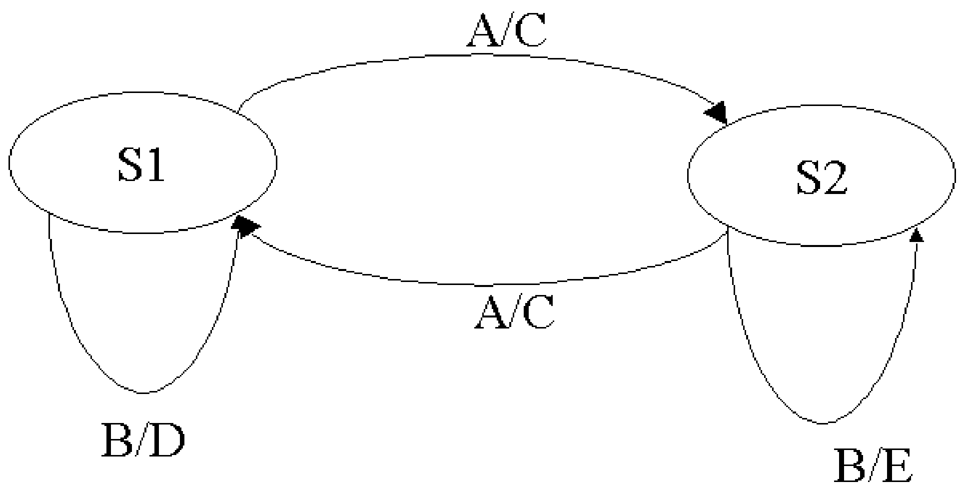


- The Save with an asterisk covers all possible signals which are not explicitly defined for this state in other input or save constructs

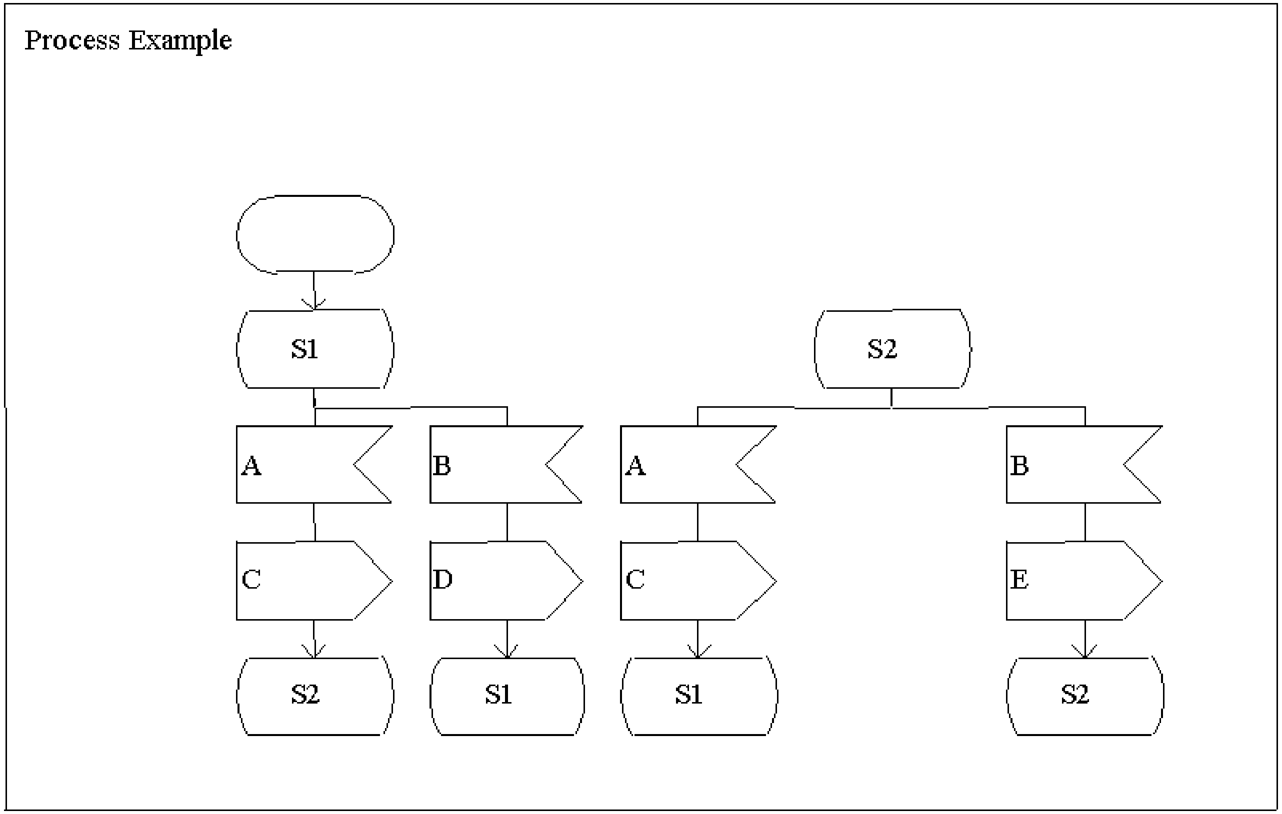
Comment Example



One Very Simple FSM (VS-FSM)

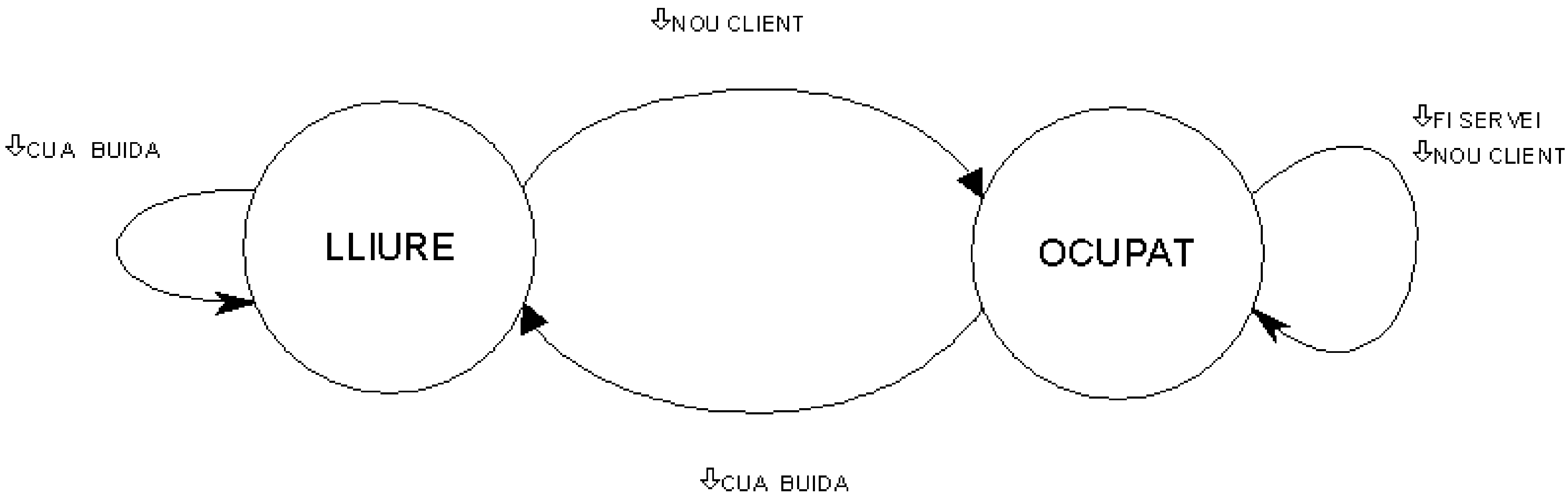


VS-FSM Process Diagram

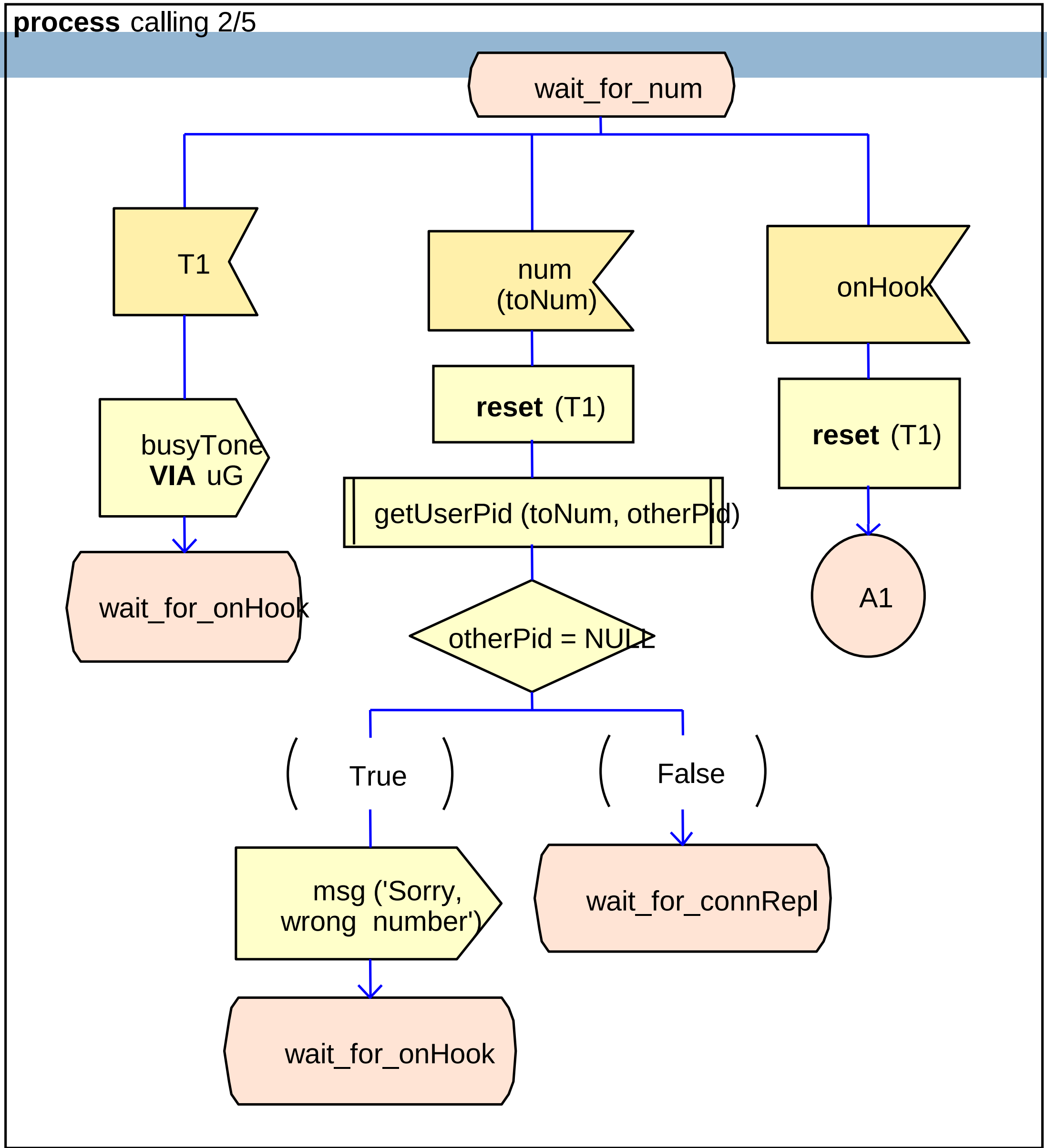


MM1 example

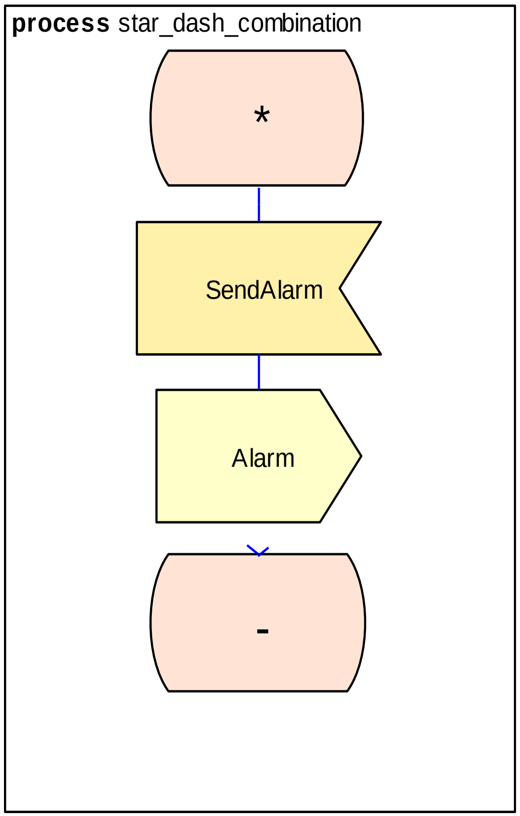
- Server states



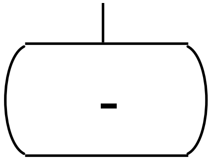
Process Diagram Example



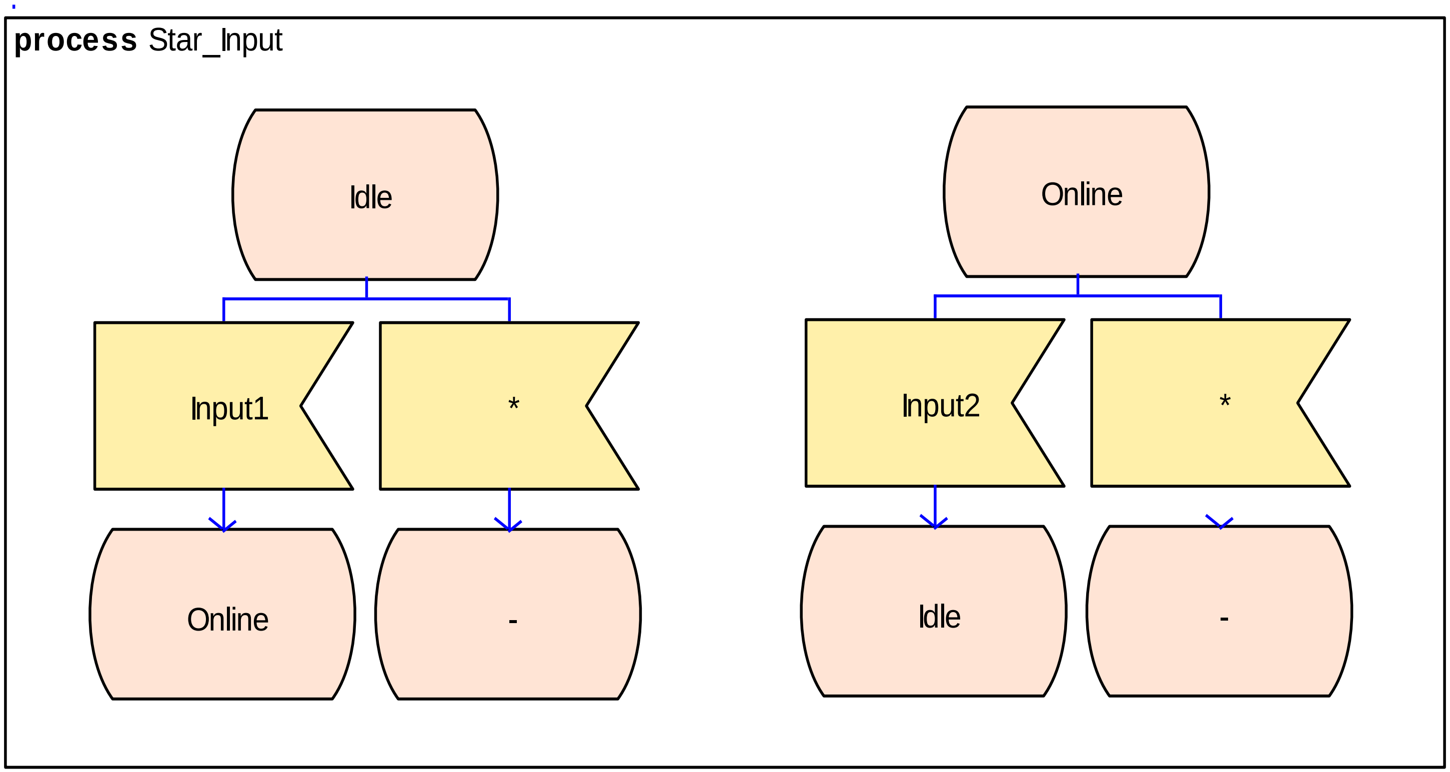
Shorthands - Same State



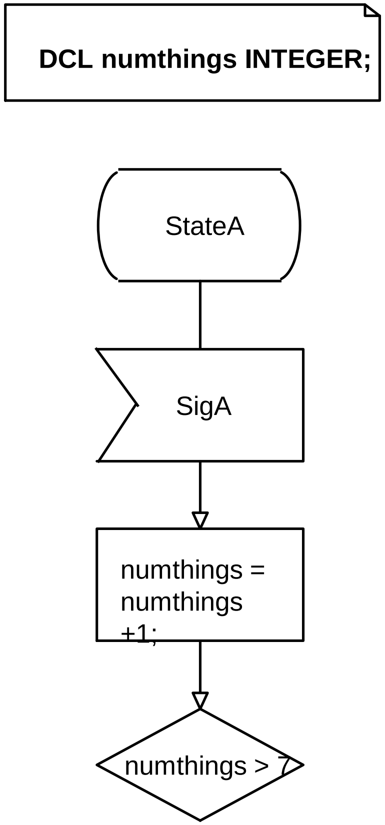
- When next state is same as current state the “dash” symbol may be used instead of state name.
- This is particularly useful in combination with * (any state)



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

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

Operators on Predefined Sorts

- 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

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 */
```

```
operator plusee;
    fpar a even, b even;
    returns res even;
    start;
        task res:=a+b;
    return;
end operator;
```

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

Data Types and Inheritance

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

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

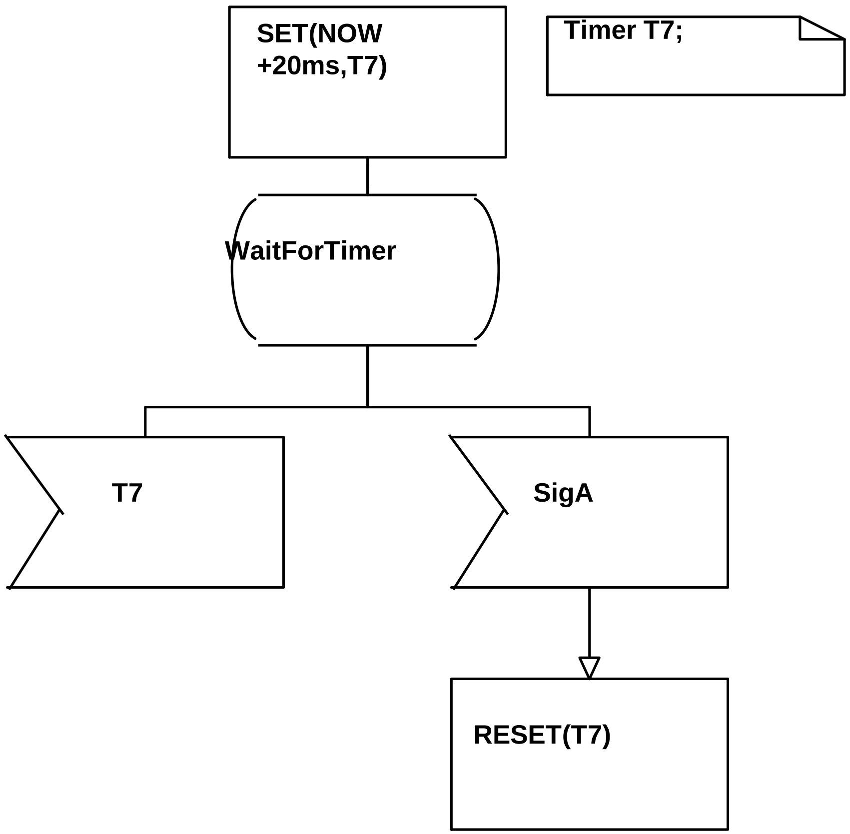
True, False are renamed to 1 & 0

Operators that are perserved

From this point new items are defined

- Most SDL protocol specifica describe data.
- Z.105 describes how SDL an used together.

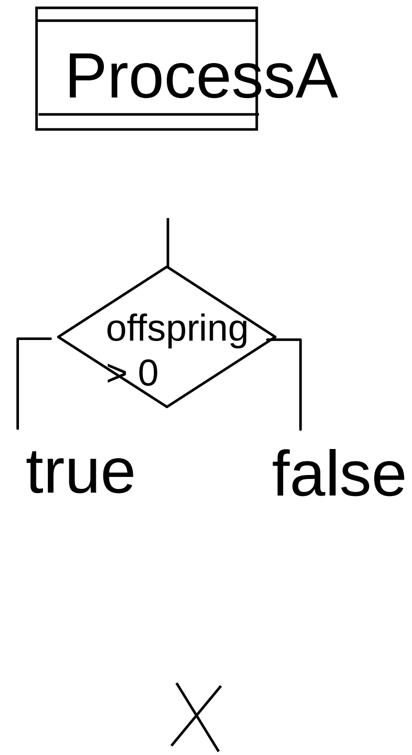
Specification of Timers in SDL



- 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.

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.

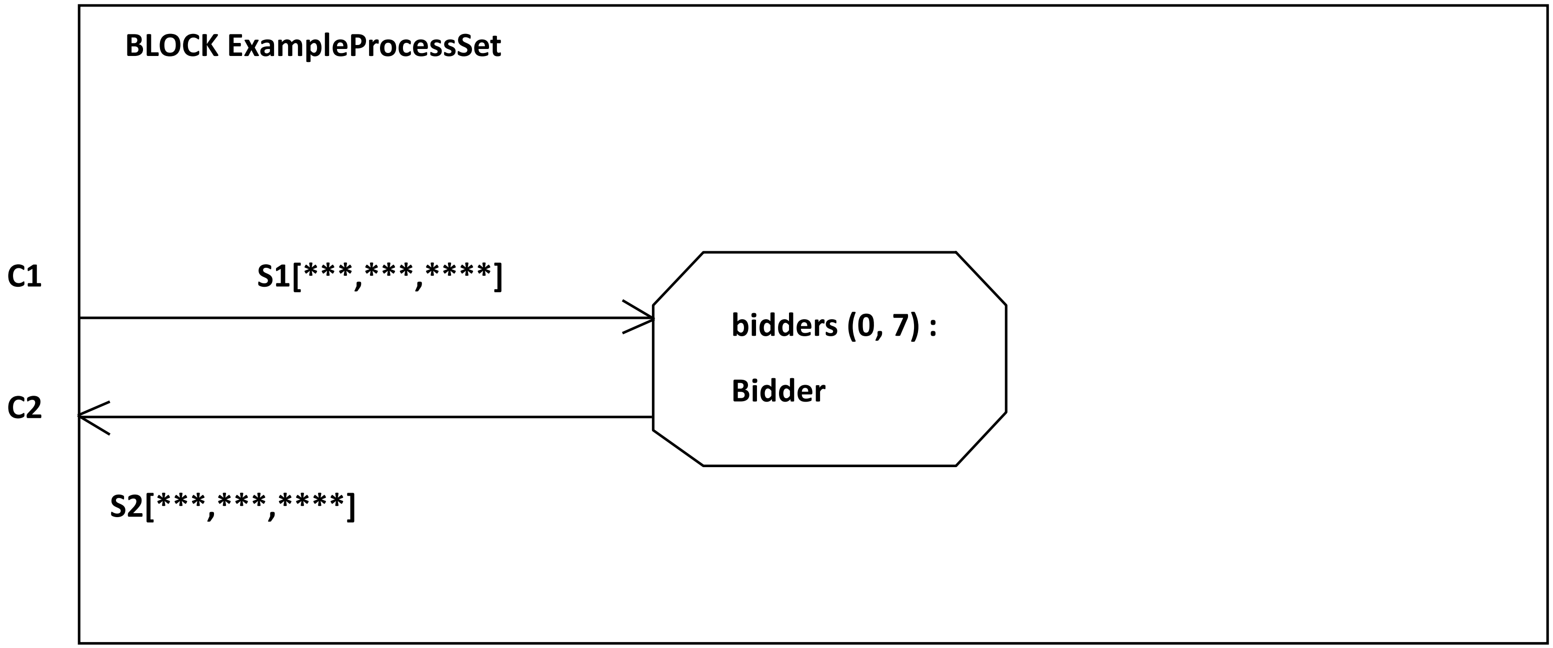


Dynamic Processes

- 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.

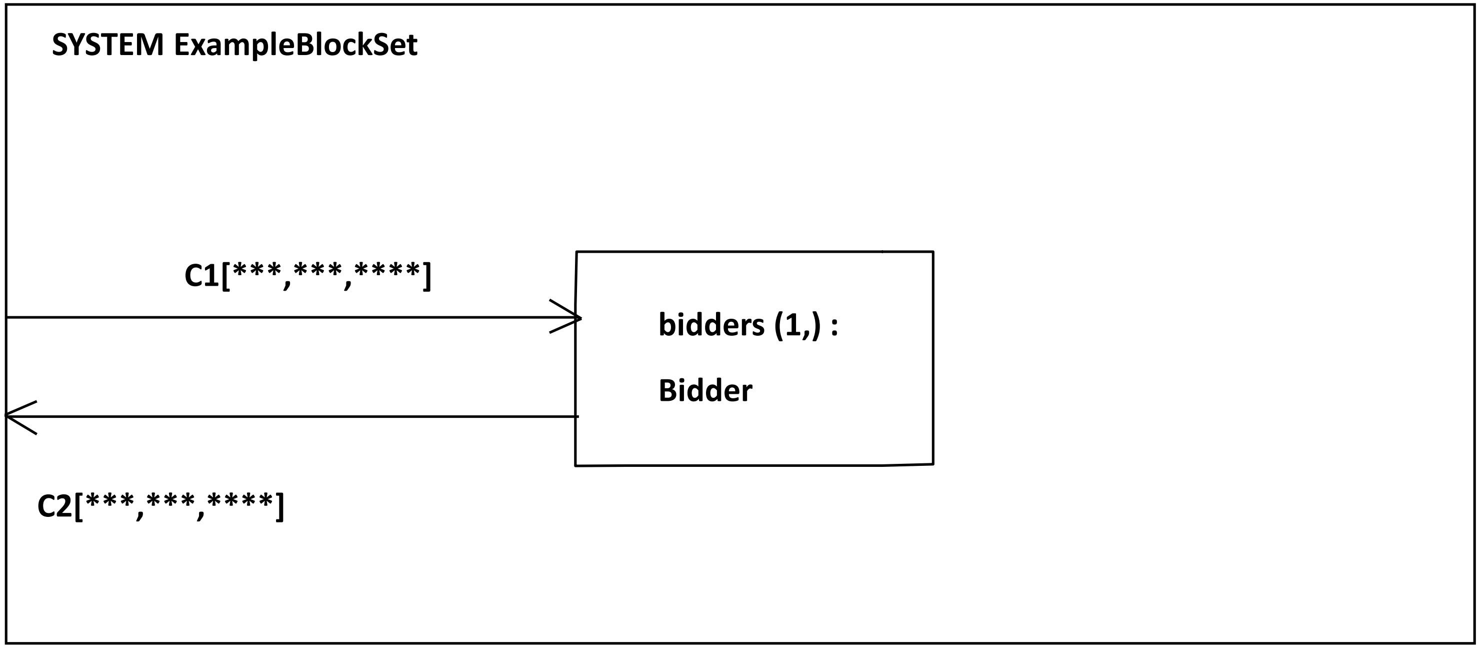
Process Sets

- 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



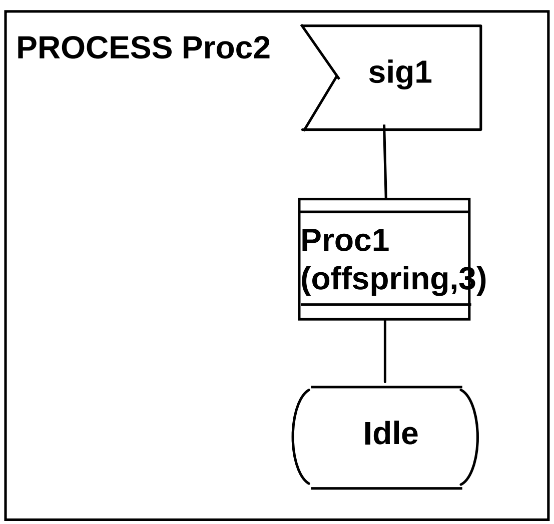
Block Sets

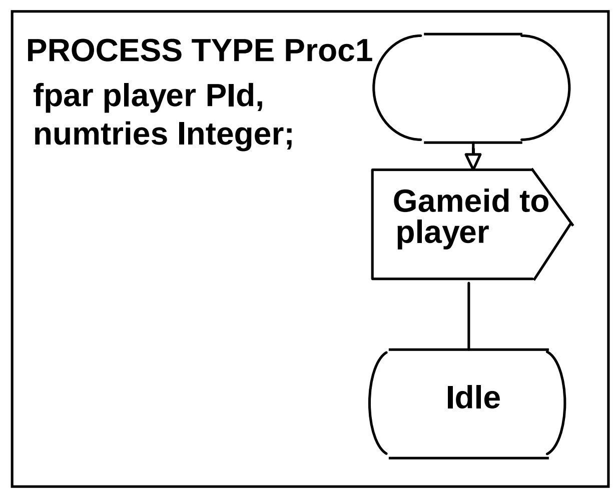
- The following Describes a set of Identical Blocks
- Initially there is one member of the set
- There is no limit to the number of members in the set



Formal Parameters

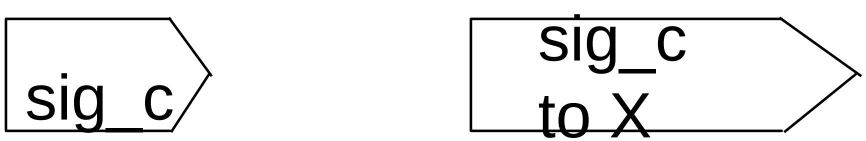
- Dynamic processes can have data passed into them at creation time using Formal Parameters
- Similar to C++ constructor





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).

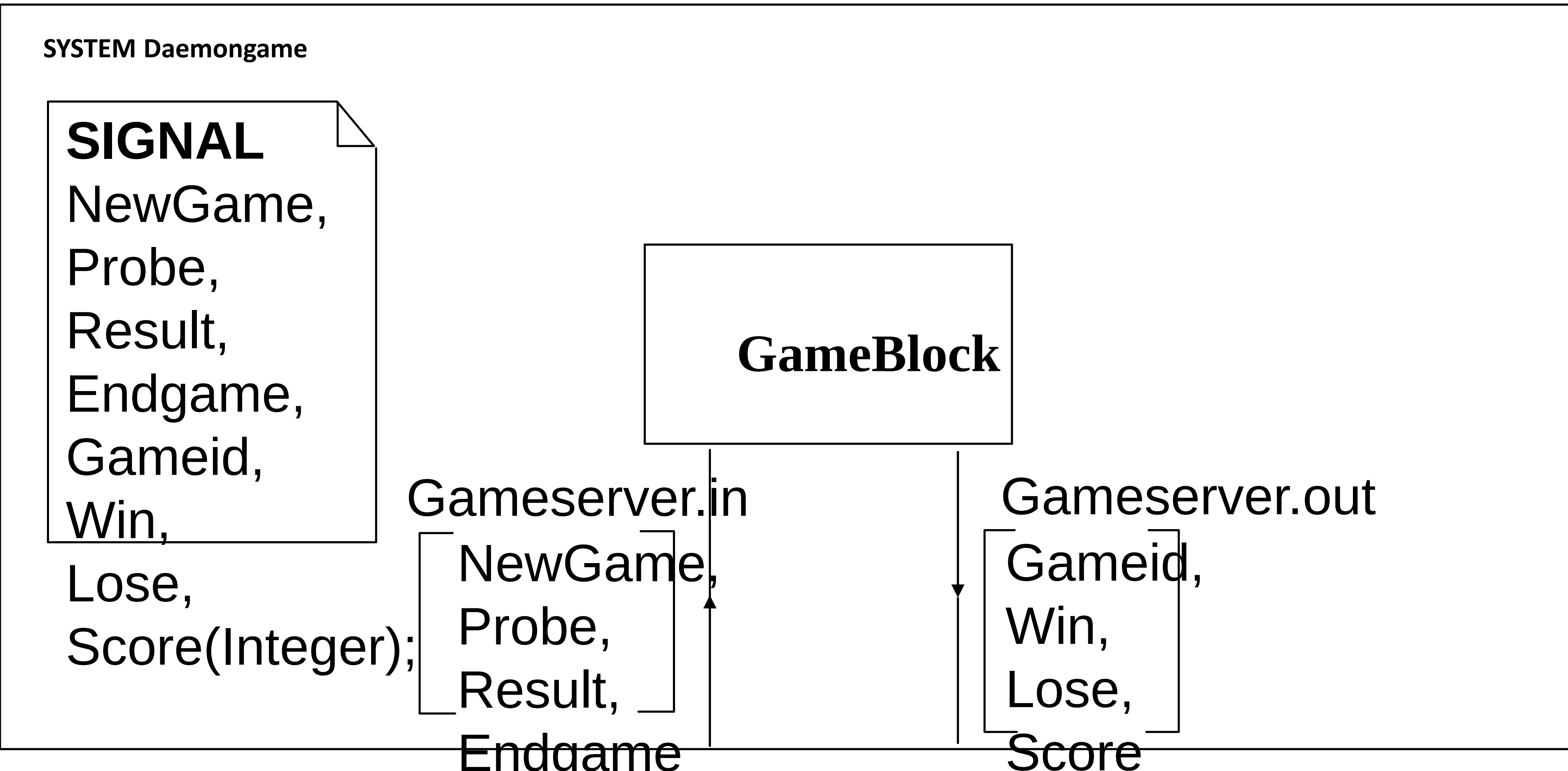
Examples

Daemon Game Example

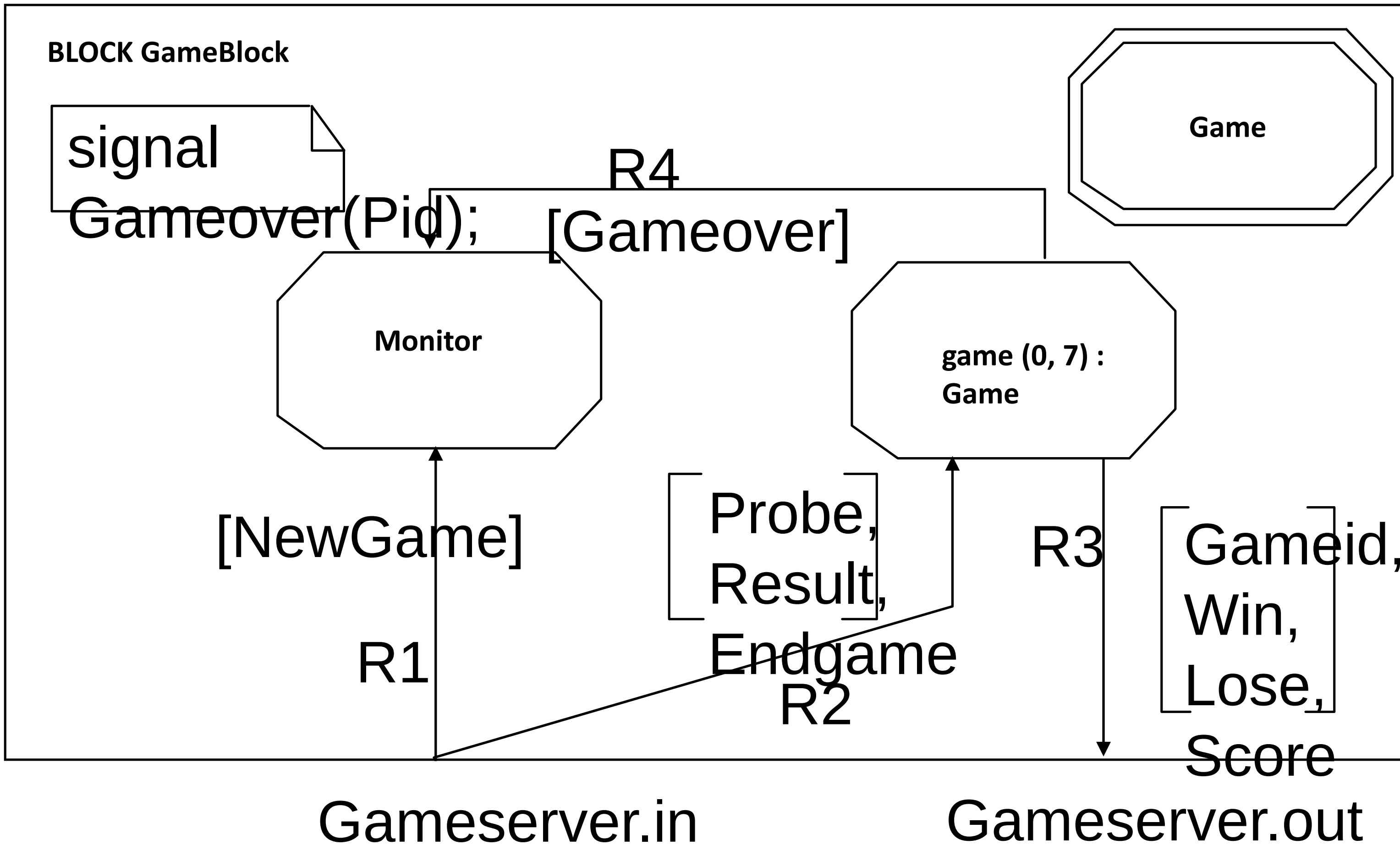
Daemon Game Example

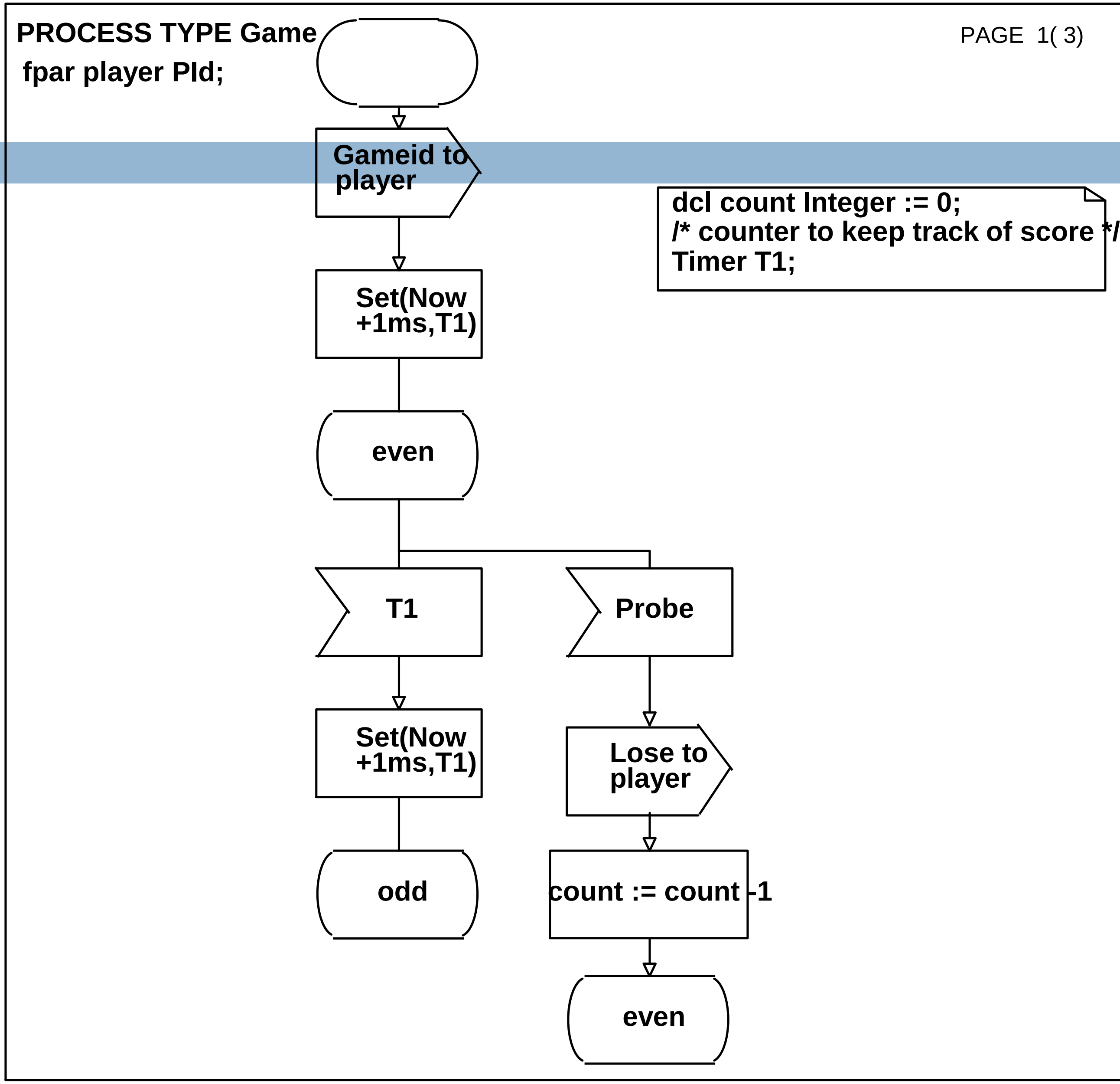
- The Z.100 standard partially defines an example of SDL in the form of a game called DaemonGame. A modified version is described here.
- The game consists of a quickly oscillating state machine, oscillating between odd and even.
- At random intervals the player queries the state machine.
- If the machine is in the odd state the player wins
- If the machine is in the even state the player looses.

System Diagram

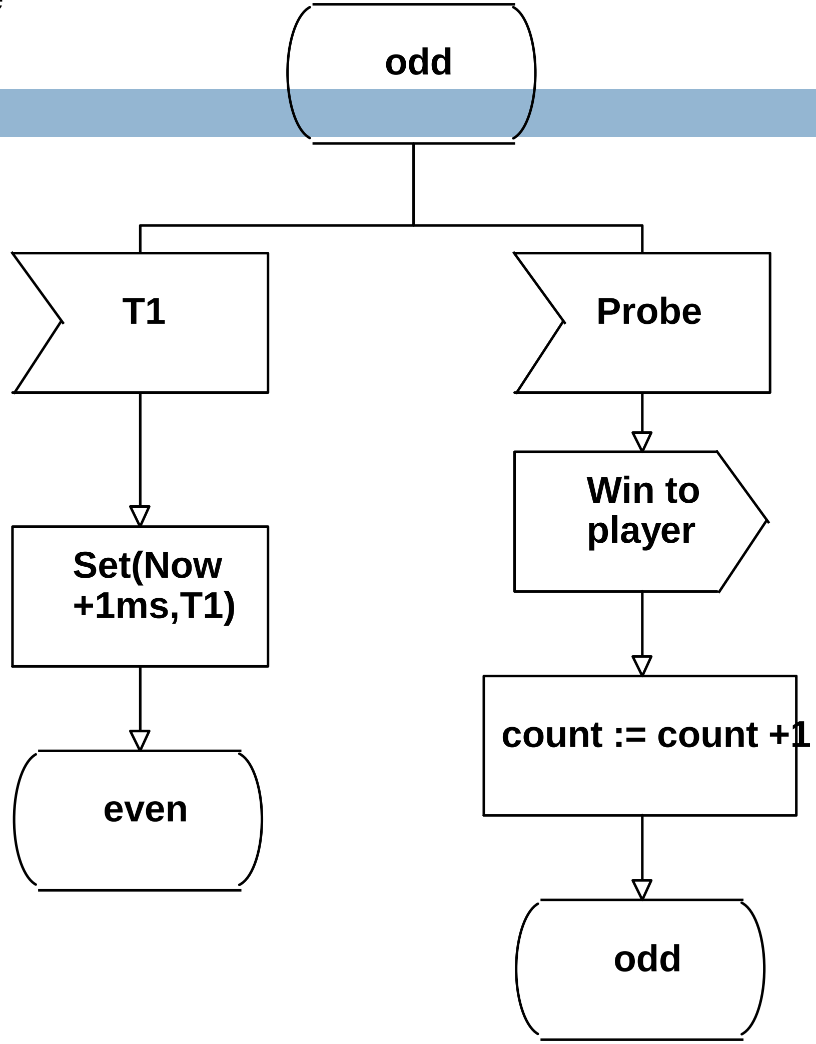


Block Diagram

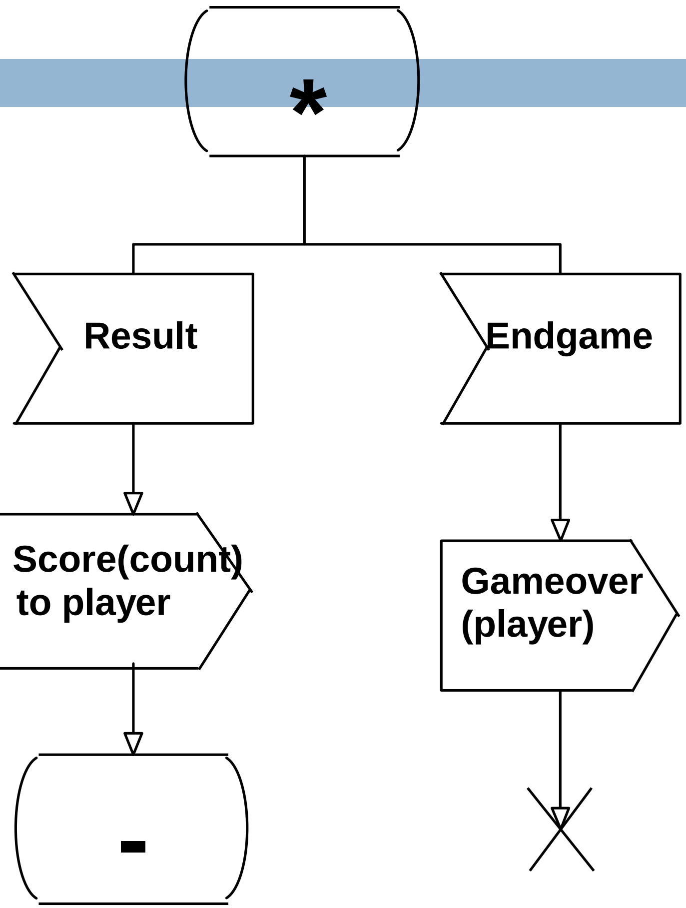



PROCESS TYPE Game
fpar player PId;
PAGE 1(3)
Gameid to player
dcl count Integer := 0;
/* counter to keep track of score */
Timer T1;
Set(Now +1ms,T1)
even
T1
Probe
Set(Now +1ms,T1)
Lose to player
odd
count := count -1
even

PROCESS TYPE Game



PROCESS TYPE Game



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.

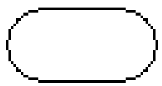
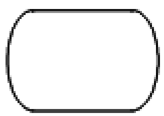
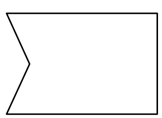
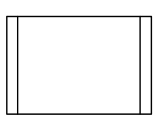
SDL for simulation

Using SDL to represent the behavior of the simulation model elements.

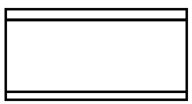
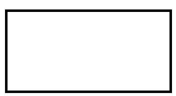
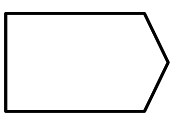
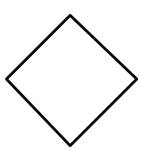
Preliminary comments

- No all the elements of the SDL formalism can be used in all the diagrams.
- The simulation engine manages all the delays:
  - The timers cannot be used inside the process diagrams.
  - The channels cannot be delayed channels.

Process diagram useful elements for simulation

	**Start.** Allows defining the initial state of a process.
	**State.** A state element contains the name of a state. All diagrams start and end with **state** elements. One process can start with the **start** element.
	**Input.** These elements describe the kind of events that can be received depending on the state and the numbers of the ports that these events travel through. All branches of a specific **state** start with an **Input** element, since an object changes its state only after a new event is received.
	**Procedure call.** These elements perform actions that do not generate delays in the model (delays are modeled through the event processing time parameterization).

Process diagram useful elements for simulation

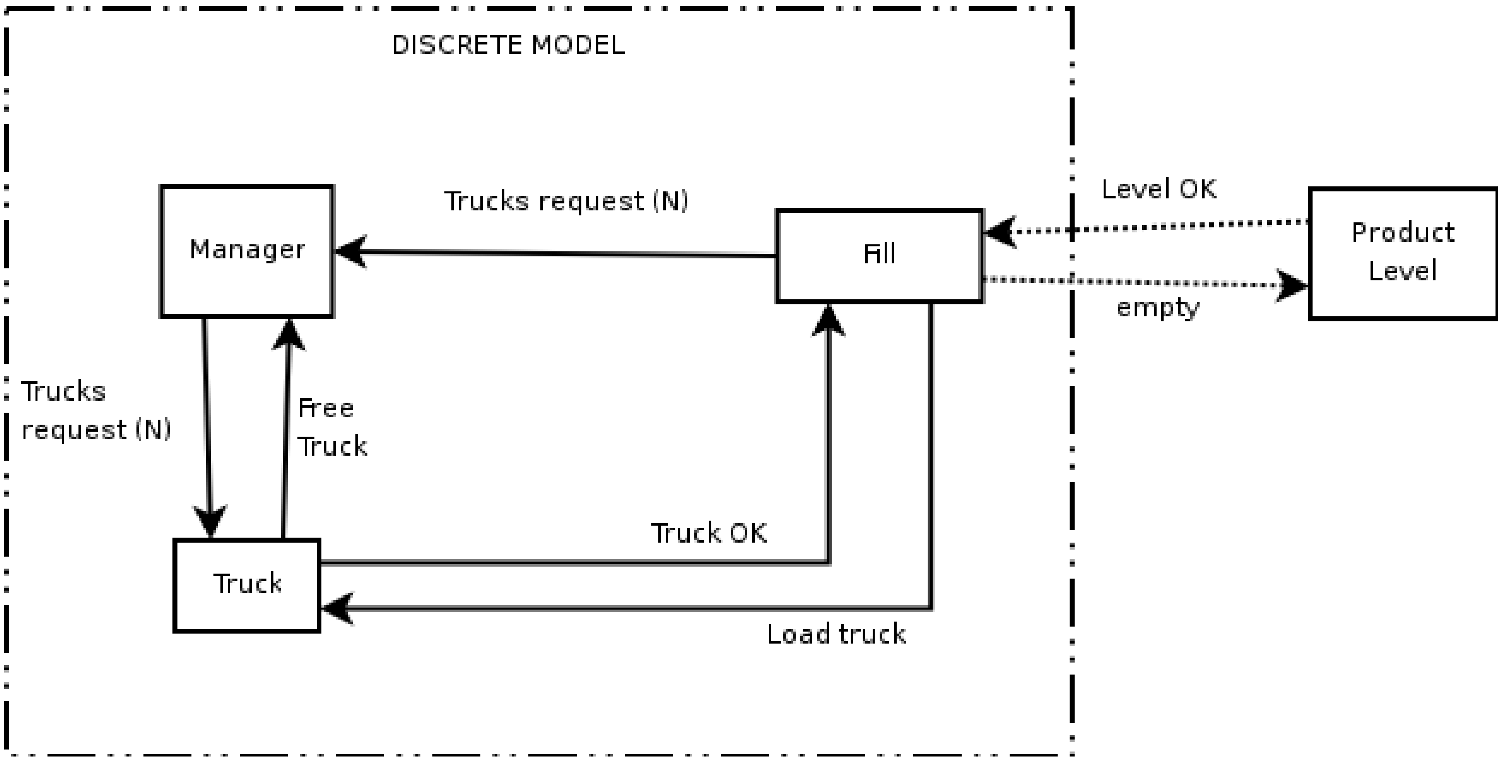
	**Create.** This element allows the creation of an object.
	**Task.** This element allows the definition of assignments, assignments attempts or the interpretation of informal texts.
	**Output.** These elements describe the kind of event to be sent and the port used. Other attributes of the event can also be detailed (priority, execution time, etc.).
	**Decision.** These elements describe bifurcations. Their behavior depends on how the related question is answered.

SDL simulation examples

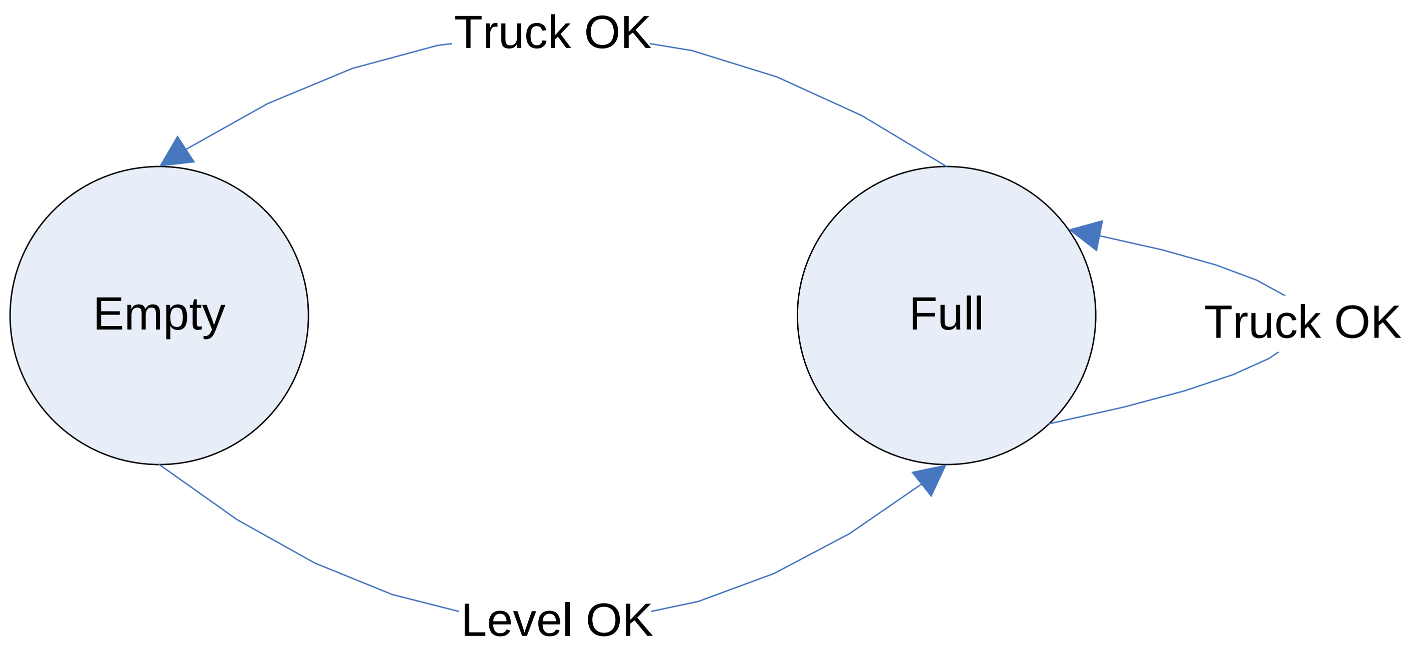
Chemical container.

The trucks fleet.

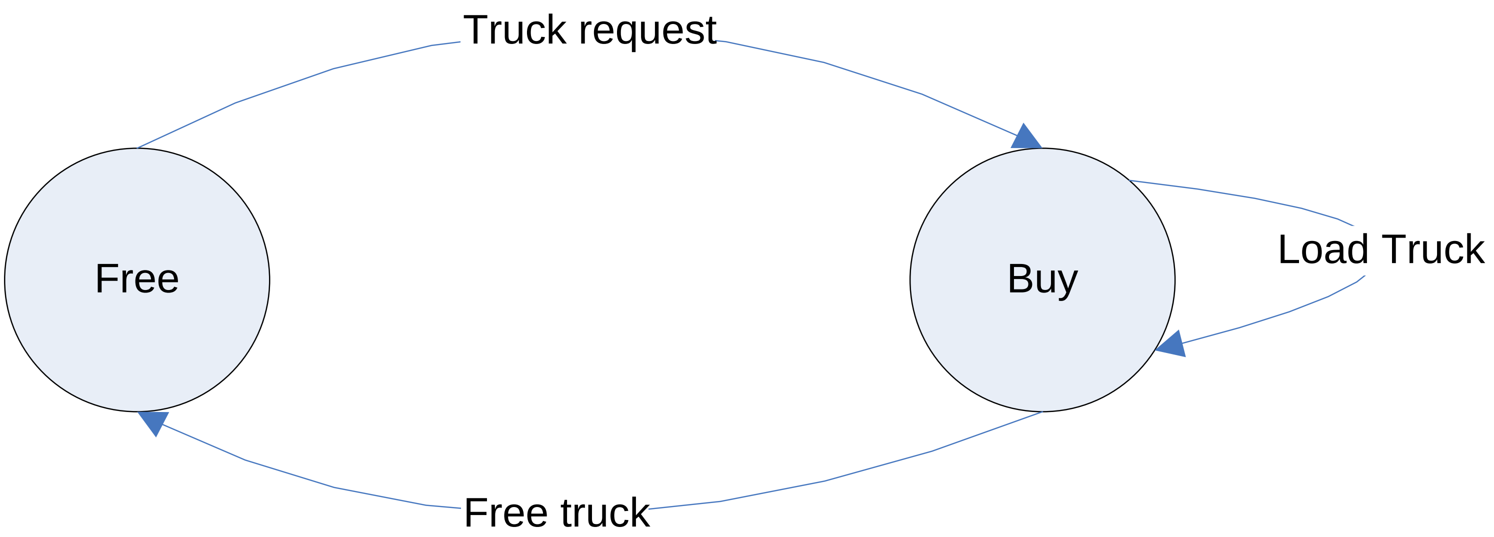
The trucks fleet



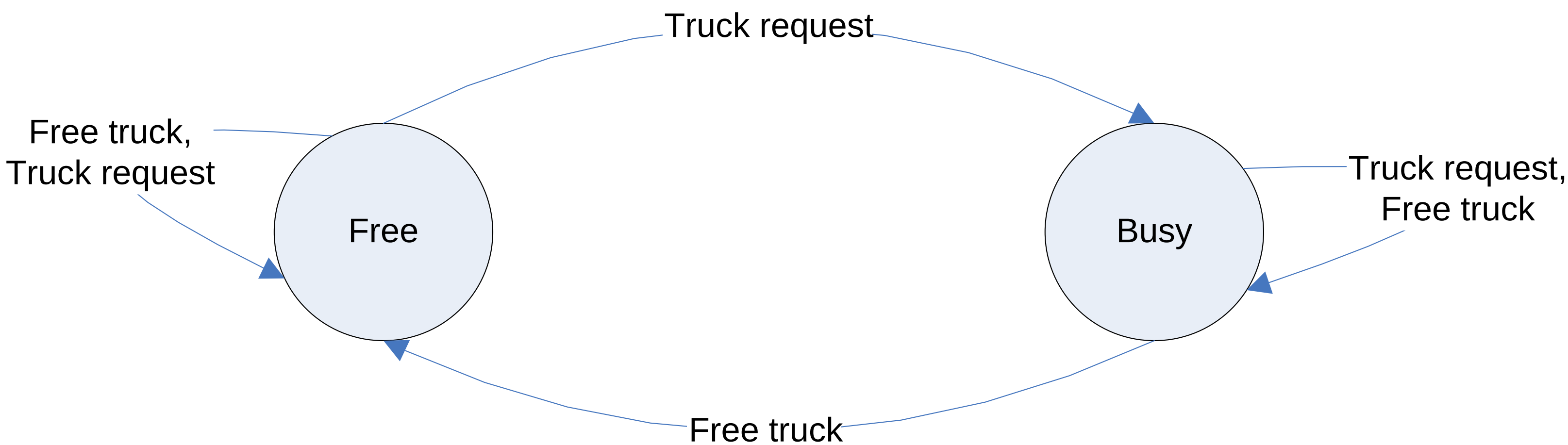
State diagram Fill element



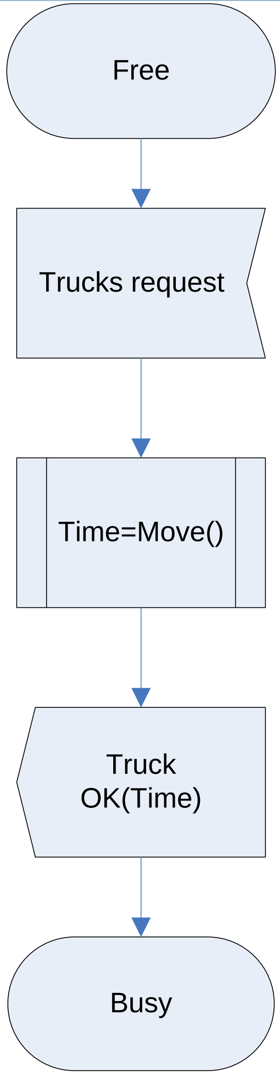
State diagram Truck element



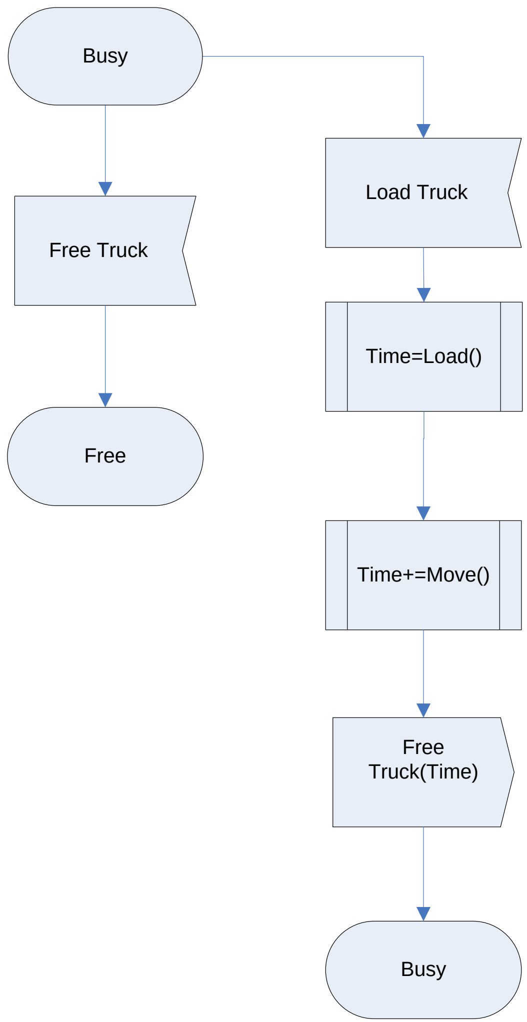
State diagram Manager element



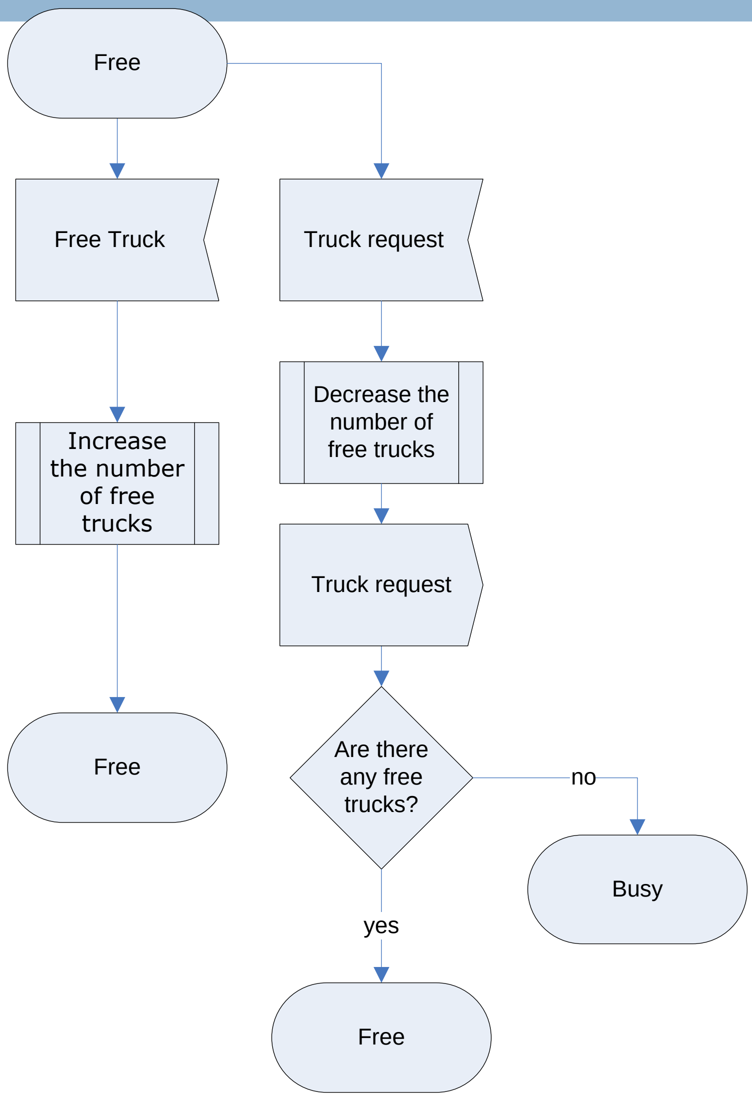
Process Truck element “free” state



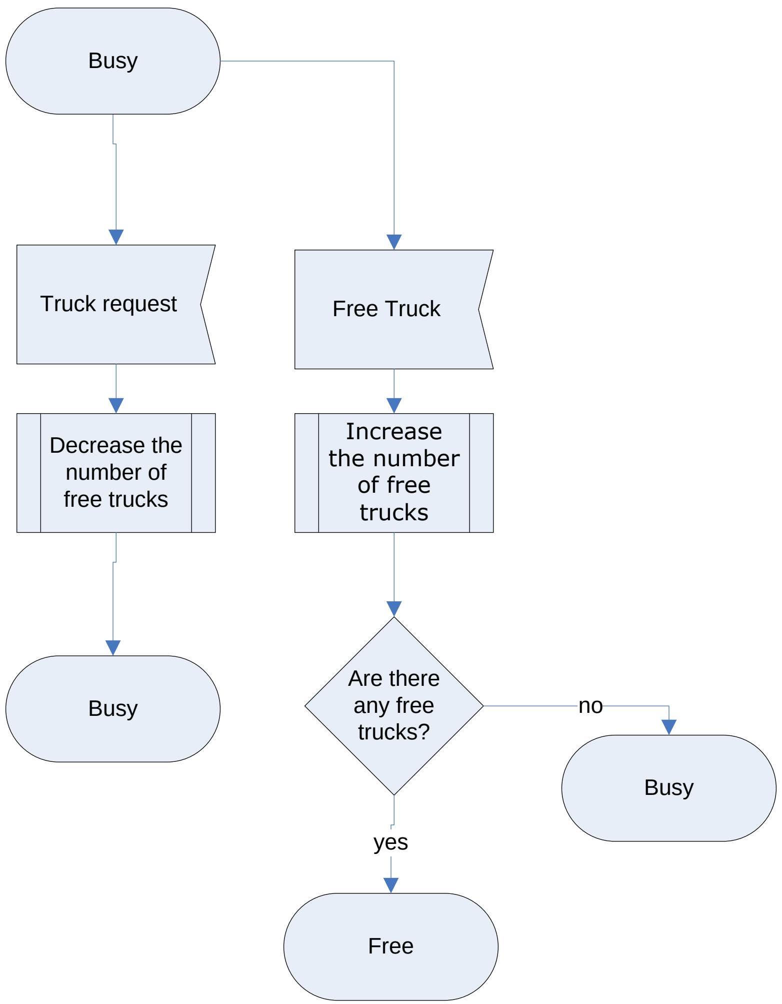
Process Truck element "busy" state.



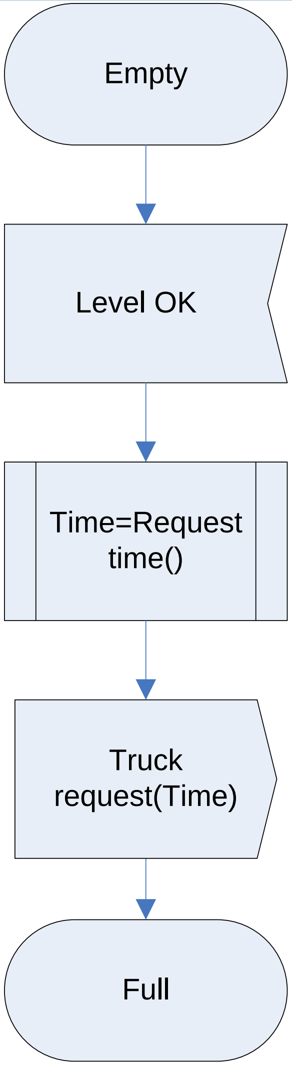
Process Manager element "free" state



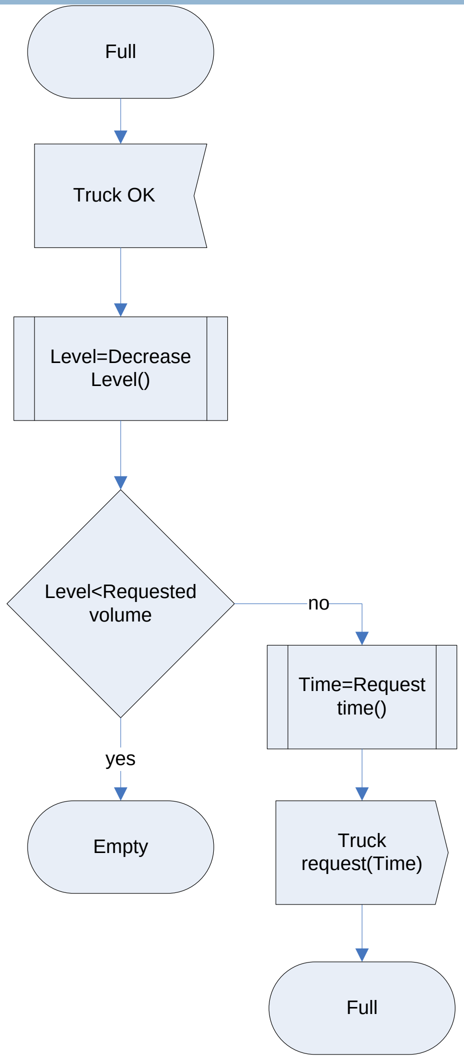
Process Manager element “busy” state



Process Fill element "empty" state



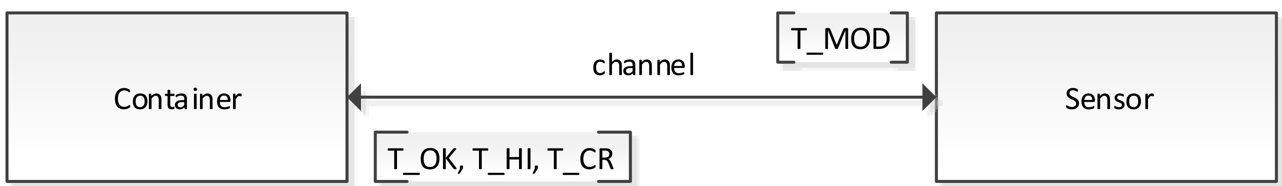
Process Fill element “full” state

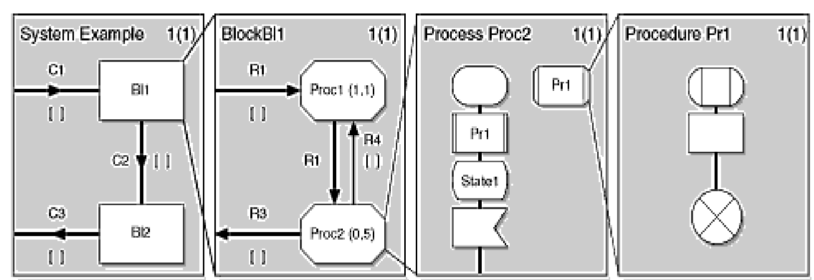


Chemical container example

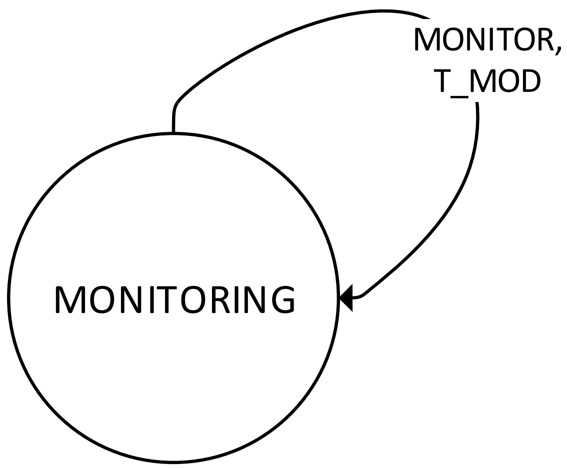
- A chemical container containing an inflammable product
- A sensor controls the temperature and shows if this is normal, high or critical.
- If temperature is high the doors of the room are closed and a reaction with a B product starts in order to reduce the temperature.
- If the temperature is critical a controlled explosion is initiated.

SYSTEM FireContainer

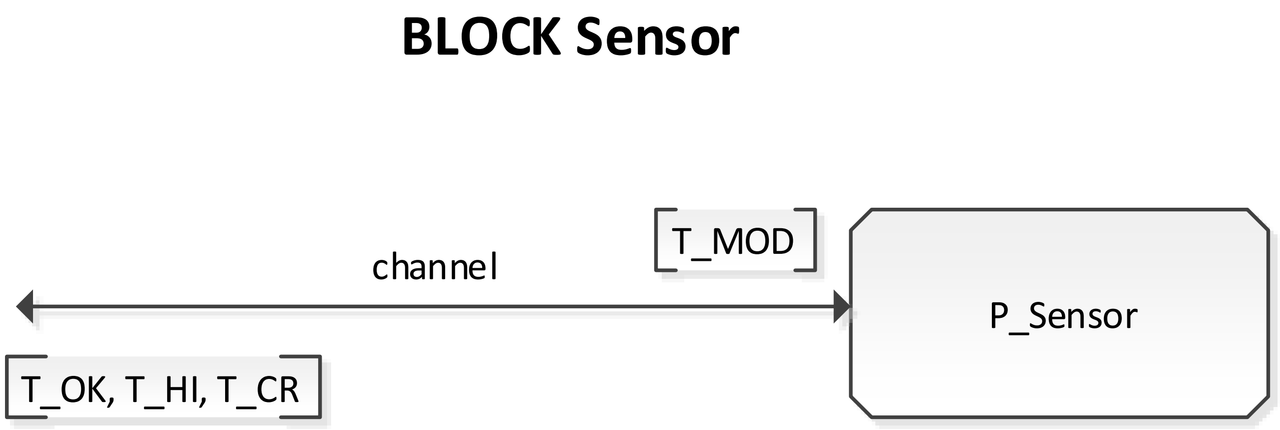




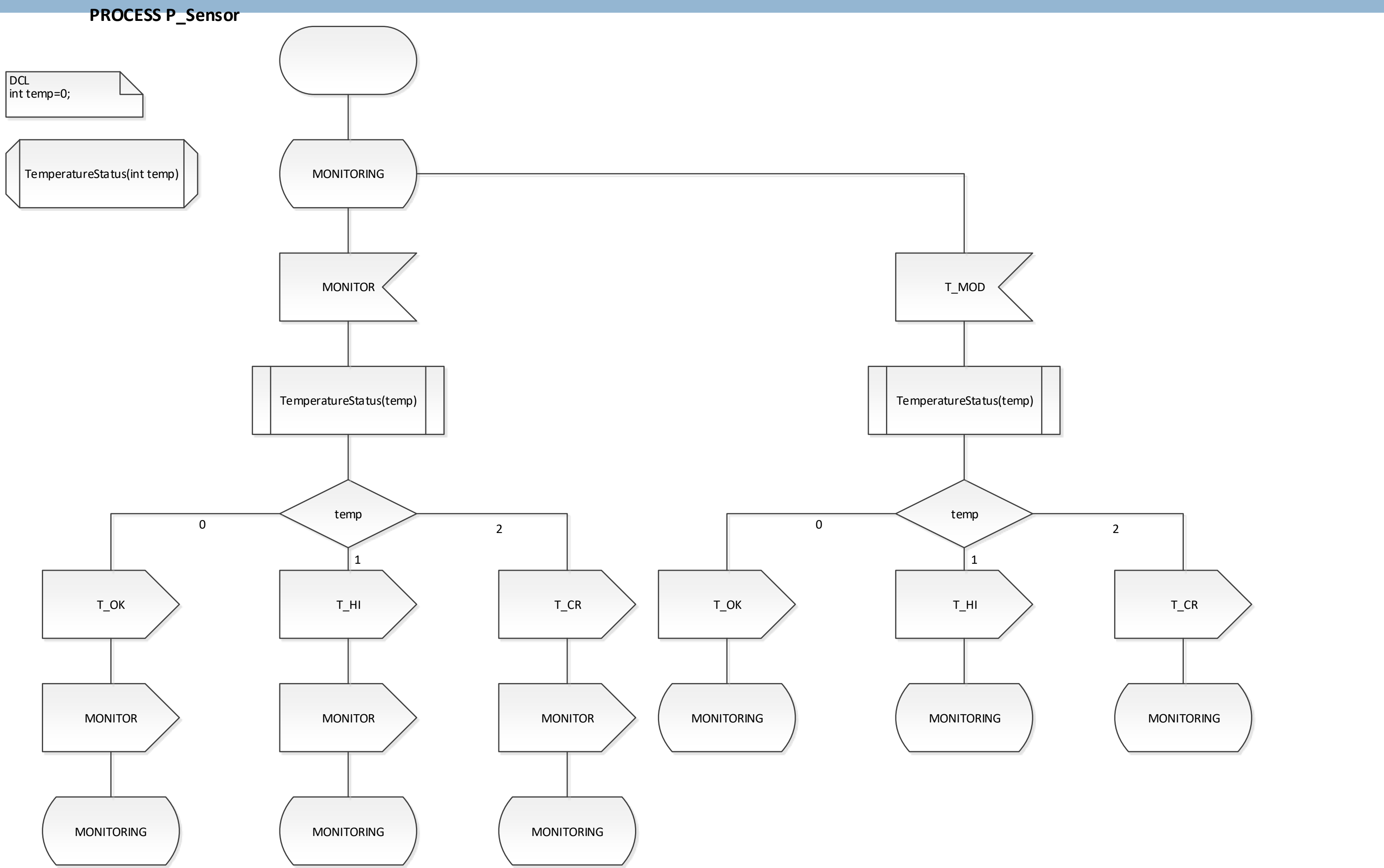
States diagram for the sensor



BLOCK Sensor



PROCESS Sensor



PROCEDURE TemperatureStatus

PROCEDURE TemperatureStatus(int temp)

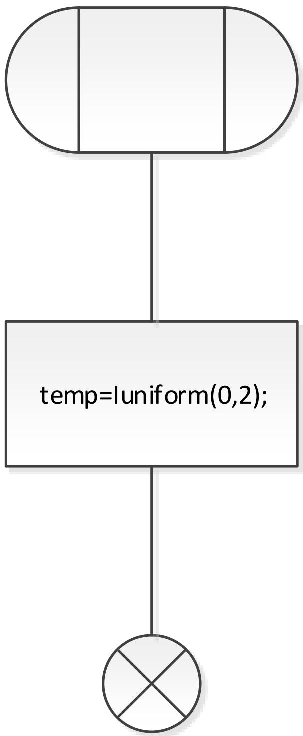
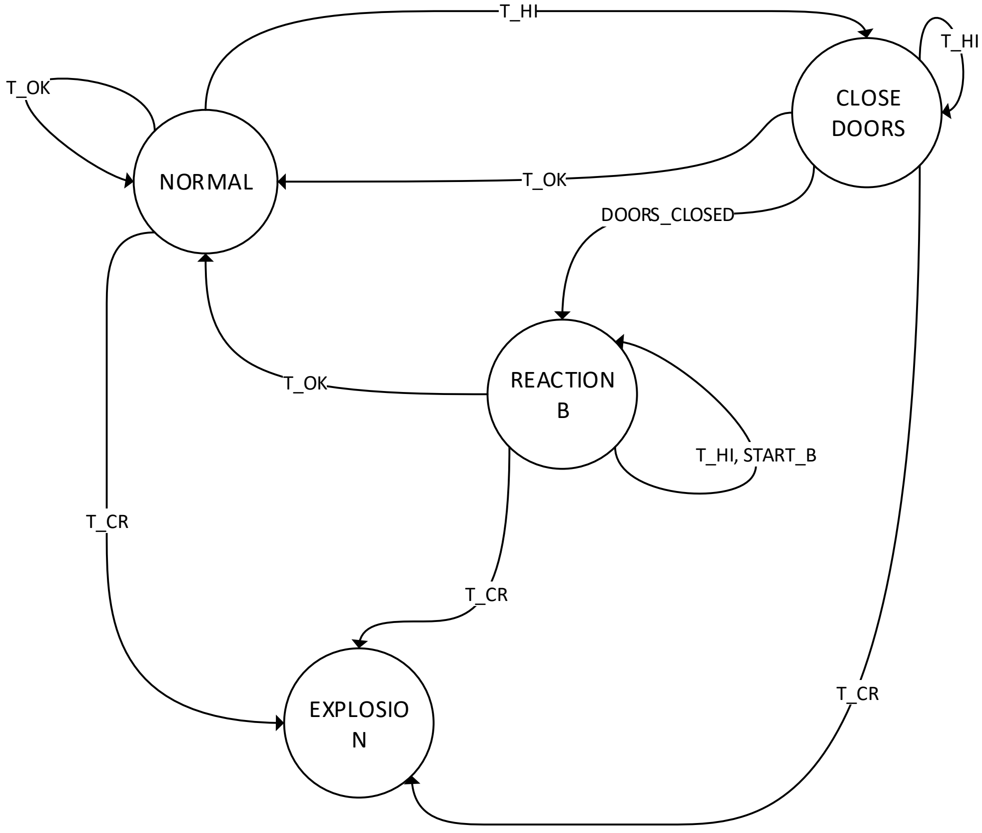
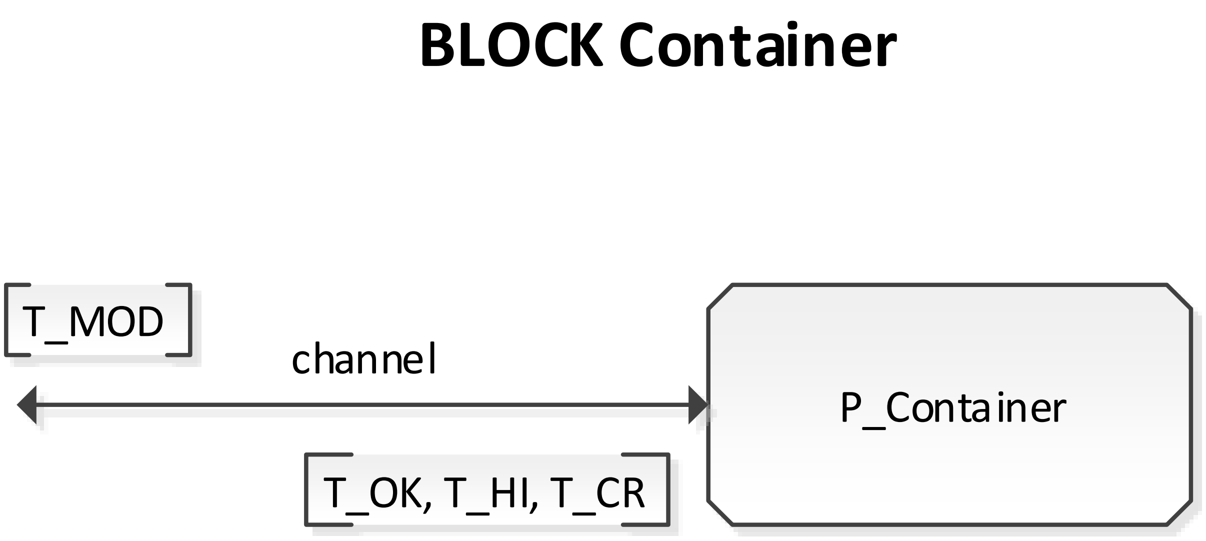


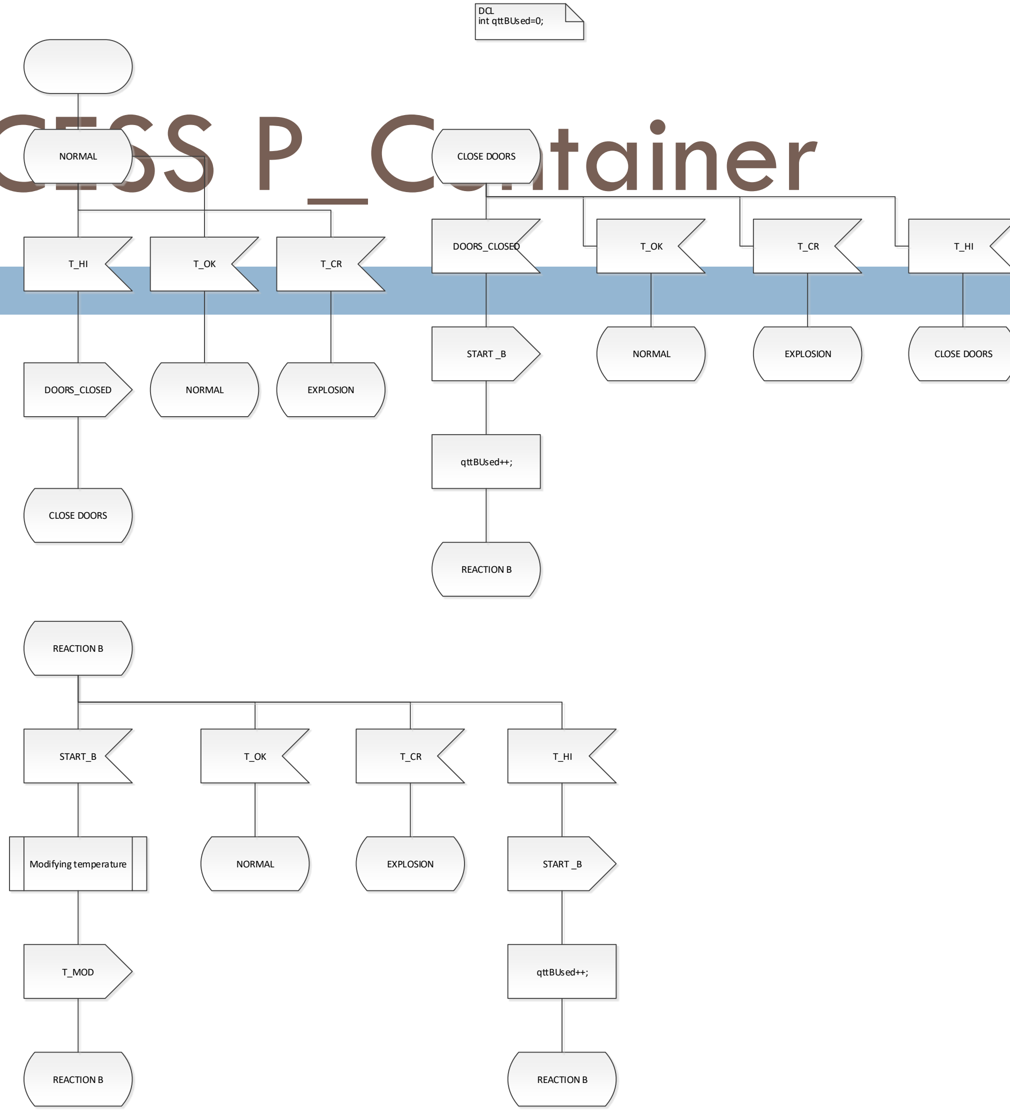
Diagrama d'estat del contenidor



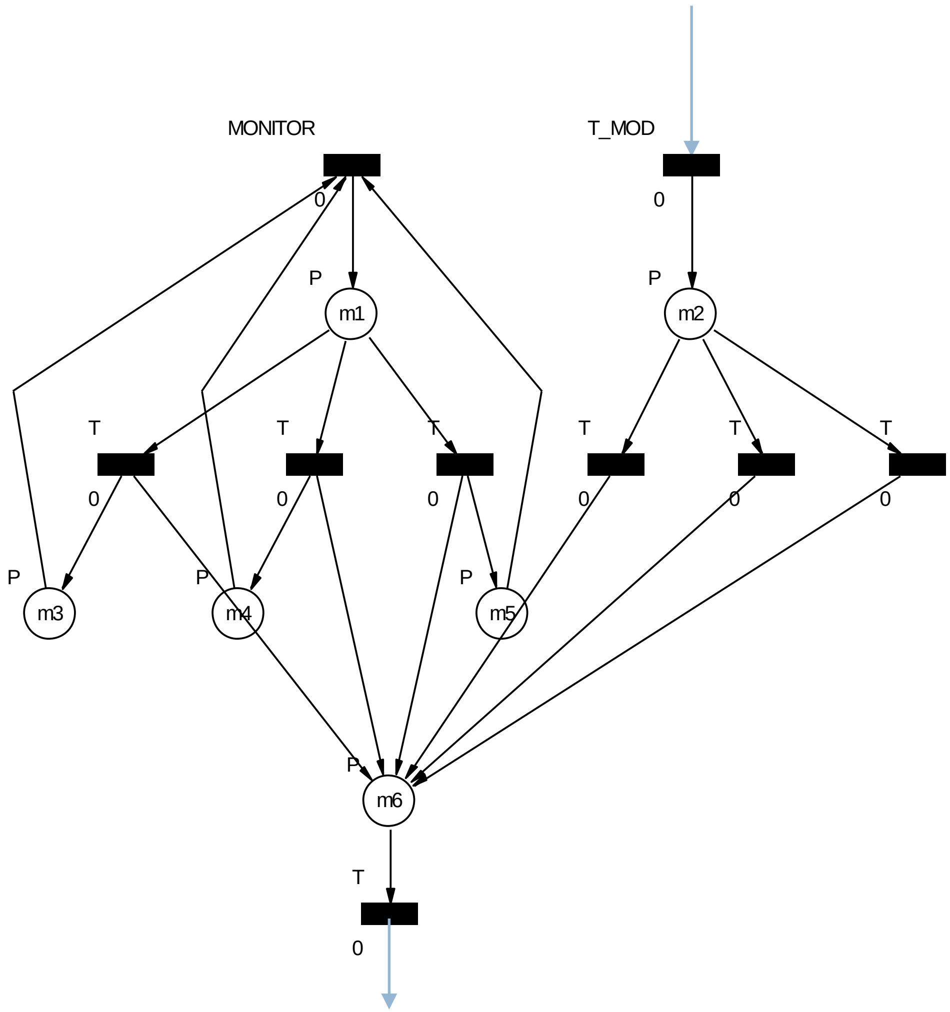
Block Container



PROCESS P_Container



Petri Net definition for the sensor



Comparative

	PetriNets	SDL	DEVS
Standard		X	
Number of rules	+	++	+++
Symetric	+++	+	++
Tools	+++	++	++
Easy to learn	+++	++	+
Code generation	++	+++	+++
Integration with 3rth tools	+	+++	++
Integration with other standars	++	+++	++