The conceptual model

The conceptual model

1. How the customer explained it
2. How the Project Leader understood it
3. How the Analyst designed it
4. How the Programmer wrote it
5. How the Business Consultant described it
6. How the project was documented
7. What operations were installed
8. How the customer was billed
9. How it was supported
10. What the customer really needed
Hypotheses

- What is inside the model?
- Hypotheses
  - Systemic / Structural
  - Simplification

Formalism must be independent from the simulation tools.

The formalized model must allow some analysis.

- To determine relations between components.

Formalism must allow an easy transformation to the representations supported by the existing simulation frameworks.

- Simplify the implementation process.
- To evaluate alternatives.
Some aspects of the model can be no specified, without causing problems in the transformation to other representations. MODULARITY

The model must be defined in terms that no constrain its codification in a particular mechanism of simulation clock update.
Modularity

- The capacity to describe the behavior of each subsystem, independent from the other subsystems that compose the model.
  - Incremental design of the model.
  - Simplifies the verification and the validation of the model.
Assure the Modularity

1. A module cannot access directly to the state of other modules or components.

2. A module must own a set of ports (input/output) to allow the interaction with the other parts of the model.
Conceptual models

- Flow models.
- ODD.
- Queue networks.
- Petri nets
- Colored Petri nets.
- SDL
- DEVS
- Causal and Forrester diagrams.
Working with different formal languages

- Three of the main mechanisms for doing this:
  - Meta-formalism.
  - Common formalism.
  - Co-simulation.

Meta-formalism

- A formalism that incorporates the different formalisms of the various sub models that makes up the system.
A mechanism that converts all formalisms to a common formalism.

Transforming algorithms from:
- SDL $\leftrightarrow$ DEVS $\leftrightarrow$ Petri Nets…

Co-simulation

- Independent simulators that work together
- HLA: The **High Level Architecture (HLA)** is a general purpose architecture for distributed computer simulation systems. Using HLA, computer simulations can interact (that is, to communicate data, and to synchronize actions) to other computer simulations regardless of the computing platforms. The interaction between simulations is managed by a Run-Time Infrastructure (RTI).
Co-simulation with SDL
Other tools

Modeling for the Raspberry Pi video & example
Submarine nuclear reactor flow diagram
A standard protocol for describing individual-based and agent-based models


https://doi.org/10.1016/j.ecolmodel.2006.04.023
Queue networks

- **M|M|1**
  - Distribution of arrival time.
  - Distribution of service time.
  - The number of parallel servers.

By Maxtremus - Own work, CC0, [https://commons.wikimedia.org/w/index.php?curid=44460399](https://commons.wikimedia.org/w/index.php?curid=44460399)
Specification and Description Language
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.
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


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 unambiguous specification and description of the structure, behavior 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 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 Télégraphique et Téléphonique [CCITT]) as recommendation Z.100.
<table>
<thead>
<tr>
<th>Series</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Organization of the work of ITU-T</td>
</tr>
<tr>
<td>B</td>
<td>Means of expression: definitions, symbols, classification</td>
</tr>
<tr>
<td>C</td>
<td>General telecommunication statistics</td>
</tr>
<tr>
<td>D</td>
<td>General tariff principles</td>
</tr>
<tr>
<td>E</td>
<td>Overall network operation, telephone service, service operation and human factors</td>
</tr>
<tr>
<td>F</td>
<td>Non-telephone telecommunication services</td>
</tr>
<tr>
<td>G</td>
<td>Transmission systems and media, digital systems and networks</td>
</tr>
<tr>
<td>H</td>
<td>Audiovisual and multimedia systems</td>
</tr>
<tr>
<td>I</td>
<td>Integrated services digital network</td>
</tr>
<tr>
<td>J</td>
<td>Cable networks and transmission of television, sound programme and other multimedia signals</td>
</tr>
<tr>
<td>K</td>
<td>Protection against interference</td>
</tr>
<tr>
<td>L</td>
<td>Construction, installation and protection of cables and other elements of outside plant</td>
</tr>
<tr>
<td>M</td>
<td>Telecommunication management, including TMN and network maintenance</td>
</tr>
<tr>
<td>N</td>
<td>Maintenance: international sound programme and television transmission circuits</td>
</tr>
<tr>
<td>O</td>
<td>Specifications of measuring equipment</td>
</tr>
<tr>
<td>P</td>
<td>Telephone transmission quality, telephone installations, local line networks</td>
</tr>
<tr>
<td>Q</td>
<td>Switching and signalling</td>
</tr>
<tr>
<td>R</td>
<td>Telegraph transmission</td>
</tr>
<tr>
<td>S</td>
<td>Telegraph services terminal equipment</td>
</tr>
<tr>
<td>T</td>
<td>Terminals for telematic services</td>
</tr>
<tr>
<td>U</td>
<td>Telegraph switching</td>
</tr>
<tr>
<td>V</td>
<td>Data communication over the telephone network</td>
</tr>
<tr>
<td>X</td>
<td>Data networks, open system communications and security</td>
</tr>
<tr>
<td>Y</td>
<td>Global information infrastructure, Internet protocol aspects and next-generation networks</td>
</tr>
<tr>
<td>Z</td>
<td>Languages and general software aspects for telecommunication systems</td>
</tr>
</tbody>
</table>
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.1 modules;
- **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
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
  - 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.
P. Fonseca i Casas, J. Casanovas, X. Ferran

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


http://dx.doi.org/10.1016/j.simpat.2014.03.008
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
- Large number of processes 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 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 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](http://www.iec.org/online/tutorials/sdl/topic04.html)
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

Channels

Blocks

Frame symbol - boundary between system and environment

Signal Descriptions
in text symbol

Signal Lists

SIGNAL S1, S2, S3, S4, S5 ;
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
Decomposition Rules:
No Limit in number of Block levels
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

```plaintext
package defs
/* Signals between users * (internal) */
SIGNAL
    connReq,
    connFree,
    connBusy,
    connEstablish,
    connEnd;
/* Signals from a user (ENV) */
SIGNAL
    offHook,
    onHook,
    num (num_t);
```

Text Box Example
Signals

- Signals are the actual messages sent between entities

- Signals must be defined before they can be used:

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

  \[
  \{, \text{<signal name>} \ [(\text{<sort name>}\{,\text{<sort name>}\}^*)]\}\}^*;
  \]

Example:

\[
\text{SIGNAL}
\text{doc (CHARSTRING), conf, ind (MsgTyp), req (MsgTyp);}
\]
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 carried 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: `Status(True)`
A signal lists may be used as shorthand for a list of signal identifiers.

Example:

```c
system localExchange

/* 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;
```

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 carries
- 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.

\[ \begin{align*}
\text{C1} & \quad [S1,S2] & \quad \text{Uni-directional delaying Channel} \\
\text{[S1,S2]} & \quad \rightarrow & \quad \text{C2} & \quad [S3,S4] & \quad \text{Bi-directional delaying Channel}
\end{align*} \]
Non-Delivering Channels

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

- Uni-directional non-delivering Channel
- Bi-directional non-delivering 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

Signal Routes

Process
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 TYPE Game
fpar play Pld

odd

T1

Set(Now +1ms, T1)

even

Probe

Win to player

count := count +1

odd
Since SDL 92 reusable components may be defined as types and placed into libraries called packages. This allows the common type specifications to be used in more than 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 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;
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.

Parameterised 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.
This model corresponds to an Automated Teller Machine (ATM). Banking transactions are performed by means of cash card. This ATM allows cash withdrawal only. Withdrawals must be authorized by the consortium, and in case of success, must be reported to the consortium. */
ATM Example - UI Block Diagram

- **UI**
  - Customer
  - Eco_UI
    - UI (1,1):
      - Eco_UI
    - cent
    - cust
  - card, entry, cashtaken, quit
  - display_wait, print, cash, eject, go_ATM, stop_ATM
package bank_lib

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

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};

signal entry (Charstring),
cashtaken,
quit,
r_accept (RespConso),
stop_tr;

signal display_wait (Charstring),
print (Charstring),
cash (Charstring),
eject,
tr_end,
q_accept (QuestConso),
wdrok (CashCard, Charstring);

signal card (CashCard),
go_ATM,
stop_ATM;

/* This implements a simplified banking transaction. */
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)
wait_for_connection

connectTone

reset (T1)

connectTone VIA uG

connected

Connected

onHook

reset (T1)

connEnd TO otherPid

idle

connEnd TO otherPid

set (NOW + T_10sec, T2)

wait_for_onHook

T1

busyTone VIA uG
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.
  - No effects on the semantics.
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
  - On Petri Nets this defines the initial marking!
**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)

```plaintext
PROCEDURE ProcB
fpar player Pld;
```

![Diagram of PROCEDURE ProcB]

- sigA
  - ProcB (SENDER)
  - stateC

- Gameid to player
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.
procedure EjectCard

Writeln
('Take your card')

TakeCard_rq

SET(tCard)

This is a comment

CardEjected
One Very Simple FSM (VS-FSM)
VS-FSM Process Diagram

Process Example

Diagram showing a state machine with states S1 and S2, transitions labeled A, B, C, D, E, and paths S1, S2.
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

```
DCL numthings INTEGER;
```

```
StateA

SigA

numthings = numthings + 1;
```

```
numthings > ?
```
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.
Time in SDL2010

FinishService1 VIA S1Ch

delay = 0;
priority = 0;
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.
Process Sets

- 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.
  - 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 ExampleProcessSet

C1  S1[***,***,****]  bidders (0, 7) :
     Bidder
    C2

S2[***,***,****]
```
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.

```
SYSTEM ExampleBlockSet

C1[***,***,****]   bidders (1,):
                  Bidder

C2[***,***,****]
```
Formal Parameters

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

```
PROCESS TYPE Proc1
fpar player PId, numtries Integer;
Gameid to player

Idle
```

```
PROCESS Proc2

sig1

Proc1 (offspring, 3)

Idle
```
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.
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).

  `sig_c via G3`

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

  `sig_c via all`
SDL simulation example

The chemical container.
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

Container

Sensor

channel

T_OK, T_HI, T_CR

T_MOD
States diagram for the sensor

This is not an SDL diagram, since is not complete!
BLOCK Sensor

channel

[T_MOD]

P_Sensor

T_OK, T_HI, T_CR
PROCESS Sensor

DCL
int temp=0;

TemperatureStatus(int temp)

MONITORING

MONITOR

TemperatureStatus(temp)

MONITOR

TemperatureStatus(temp)

T_MOD

MONITORING

T_OK

T_HI

T_CR

0

1

2

MONITORING

MONITORING

MONITORING

MONITORING

MONITORING

MONITORING

MONITORING

MONITORING

MONITORING
PROCEDURE TemperatureStatus

PROCEDURE TemperatureStatus(int temp)

    temp = uniform(0, 2);
Diagrama d'estat del contenidor

This is not an SDL diagram, since is not complete!
Block Container

T_MOD
channel
T_OK, T_HI, T_CR

P_CONTAINER
PROCESS P_Container

DCL
int qttBUsed=0;

qttBUsed++;
qttBUsed++;
PROCESS P_Container

- REACTION B
  - START_B
    - Modifying temperature
      - T_MOD
      - REACTION B
    - T_OK
      - NORMAL
    - T_CR
      - EXPLOSION
    - T_HI
      - START_B
      - qttBUsed++;
    - T_CR
      - EXPLOSION
      - START_B
      - REACTION B
  - T_OK
    - NORMAL
  - T_CR
    - EXPLOSION
Petri Net definition for the sensor
Questions?

http://necada.com/
http://project.necada.com/

Ph. D. Pau Fonseca i Casas
InLab FIB, Head of Environmental Simulation
pau@fib.upc.edu