**Specification Description Language (SDL)**

SDL improves on the structured analysis method in that, while structured analysis focuses only on the data in a system and the operations on that data, SDL focuses *both on the data in a system and on the behaviour of the system.*

Development of SDL began in 1972 and has been expanded on by the CCITT since then. The latest version of it is SDL-92. It is a recognised international standard and is constantly growing in popularity. It can be used in all real-time and interactive systems, but is of particular interest in the telecoms field as it was developed *specifically* as a tool for modelling telecommunications systems.

The target object for specification in SDL is called a *system*. An SDL system is a formal model that represents significant properties of some, existing or planned, real life application system. The property that is of major interest is behaviour, i.e. what the system does and under what circumstances. Everything that is not part of the system is called the *environment*. Systems may be either *open* - interact with the environment - or *closed*. 
Before we begin to explore the use of SDL to model systems, it is necessary to make a few distinctions in the terminology that will be used. As in object oriented design, the concept of *instances* is of relevance in SDL.

A *specification* is a structural representation of a system in SDL. A specification consists of a *system, blocks, channels, processes* and *signal routes*. All of these things contribute towards describing the *structure* of the system. However, when we have developed a complete specification, we must create instances of these items in order to evaluate how they work. A parallel may be drawn with source code and an executable program. The specification is like the source code, which must be "compiled" and have various attributes or parameters supplied to it in order to create an executable program or system instance. Therefore, when designing a system in SDL, we refer to systems, blocks, processes etc., but when describing operation, we refer to *system instances, block instances, process instances* etc.

Note that for any one system specification, we can have a large number of system instances by setting different parameters for each. In other words, we can run a simulation on the system specification many times with different input parameters and its operation and output will vary each time.
**An Introduction to SDL Concepts**

*Processes and Process Instances*

The behaviour of a system instance is constituted by the combined behaviour of a number of process instances in the system instance. A *process* is a state machine that works independently of and at the same time as other processes. The process instances communicate asynchronously by sending discrete messages called *signals* (signal instances) over *signal routes* between processes. A process instance reacts to external stimuli (in the form of signals) in accordance with its specification. A process instance may also send signals to and receive signals from the environment - this represents interactions between a system and the outside world. This is shown in the diagram below - note the symbols used to represent a system and its processes.

![Diagram of system and processes](image)

Each process instance has a unique address. A signal always carries the address of both the sending and the receiving processes, in addition to any necessary values or parameters.
Therefore, the receiving process instance always knows the address of the sending process instance.

A process has a memory of its own for the storage of its variables. A variable is completely owned by a process - processes cannot alter the values of variables belonging to other processes. In fact, a special signal request is required in order to allow a process to even *view* a variable belonging to another process.

A process instance has an infinite *First-in-first-out (FIFO) queue* at its input, in which incoming signals are placed. A process is either in a *state* (waiting for some occurrence) or performing a *transition* between two states. A transition is initiated by a signal in the input queue. When a signal has initiated a transition, it is removed from the input queue and is *consumed*. During a transition, variables can be altered, decisions made, new process instances created, signals sent to other processes etc.

**Blocks and Block Instances**

All functions of a system instance are carried out by processes. However, each process only contains a small amount of functionality to represent a particular operation. Therefore, even a simple, straightforward system is made up of a large number of interacting processes. If a system specification remained this unstructured, it would greatly complicate the task of developing a
system model and make it very difficult to actually understand what the specification is supposed to do.

Therefore the concept of the block was developed and the rule made that process instances must always be contained within a block instance. This makes it possible to assemble processes involved in the same task within the same block structure for ease of development and understanding. It also provides the hierarchical structure to the system. Note that it is possible to split the system specification into a large number of layers by partitioning blocks into lower-level blocks and channels.

Therefore, there are two types of block instance:
1. The leaf-block instance, which contain only process instances.
2. Partitioned block instances, which contain only block instances and channel instances.

When specifying a block within a system, it is possible to create a combined block specification. This is a specification for a block which contains both process specifications (corresponding to a leaf-block instance) and lower-level block specifications (corresponding to a partitioned block instance). However, as a block instance may only contain either processes or sub-blocks, a choice must be made as to which one applies when a system instance is created. Therefore, one system specification may
cover a number of system instances which differ not only in the parameters supplied, but also in actual structure.

Therefore, a specification consists of a system, combined blocks, partitioned blocks, leaf-blocks and processes. However a simulation generated from the specification may only be resolved into:

- one system instance containing one or more block instances which are connected to each other and to the boundary of the system (the environment) by *channel instances*.
- Partitioned block instances which contain one or more block instances which are connected to each other and to the boundary of the block by channel instances.
- At the lowest level of the block hierarchy are the leaf-block instances. These contain one or more process instances which are connected to each other and to the boundary of the leaf-block instance by signal route instances. A channel may be pictured as containing multiple signal routes.
EXERCISE

Take the above representation of an SDL specification and show what it would look like as a layered hierarchical model. Note that the model has a tree structure with the processes at the end (lowest level) of each branch.

Representation Forms

Most formal languages only have a textual representation, while SDL also has a graphical representation, which contributes substantially to its user friendliness. In the graphical
representation, a graphical syntax is use to give an overview, complemented by a textual syntax for those concepts which cannot be described using graphical symbols. The graphical representation allows the user of SDL to specify a system using a set of symbols which cover different levels of detail, from a broad overview down to detailed design level.

The textual representation only makes use of textual syntax. The graphical and the textual representations have a common subset of textual syntax and therefore overlap each other.

**Abstract Data Types**

Note also that in SDL, the *abstract data type* approach has been chosen. This means that data types or *sorts* (e.g. integers, boolean etc.) are defined independently of implementation and only in terms of their properties. As an example, the sort *integer*, as defined in SDL, is implementation-independent and therefore does not have any restriction in size, but corresponds to the mathematical integer concept (i.e. all whole numbers between $-\infty$ and $+\infty$). The set of operations allowed on the sort are the same as usual (e.g. addition, multiplication, mod, div etc.). However, the most important factor of taking this approach is that users are allowed to define their own sorts with any kind of operations acting on them. This concept will be addressed in more detail at a later stage.
Basic SDL

Basic SDL refers to a central portion of the language, which is sufficient for most SDL specifications and covers all the detail required as part of this course. Here, only one level of blocks will be covered - block partitioning is only used in advanced system modelling and is outside the range of this course.

SDL provides two methods of describing a system - a textual method and a graphical method. The *textual syntax* is described using the so-called BNF (Backus-Naur Form), whereas the *graphical syntax* is described using a modified BNF. The textual syntax strongly resembles Pascal, which is the preferred language when comparing SDL with programming languages.

Before directly addressing the SDL concepts, some general mechanisms for specifying hierarchical systems will be introduced. This is to aid readers in understanding how a specification is divided into manageable pieces.

Here, as was stated previously, it is important to differentiate between *specifications*, *types* and *instances*. For example, a signal type has a certain name and a certain set of formal parameters. A signal instance is an actual signal that is
transmitted or received, and has a set of actual parameters, which are values. A signal specification would define the signal type. Whenever the qualification is omitted, for example by using the term signal alone, it is the instance that is meant.

Hierarchical Specifications

An SDL specification contains, as do many other languages, a hierarchy of elements or components that are arranged in a tree-like structure. This results in the system specification having a number of levels of hierarchy or abstraction. The components may now either be described directly where they are identified or their description removed from the context and placed somewhere else. In the first case, different abstraction levels are shown at once, whereas in the second case, the different levels of abstraction have been separated.

The traditional kind of specification, called a local specification, is shown below. There, nested specifications are shown, with lower-order specifications contained within the next higher order of specification. This can be compared to a circuit diagram that is completely contained in a single sheet.
Although the local kind of specification is certainly suitable from the toolmaker’s point of view and for very small specifications, it poses the following problems for the human user:

- it provides no overview,
- it provides no separation of abstraction levels,
- there is too much information in one place,
- it is difficult to map documents to the specifications.

In SDL, these problems have been solved by introducing the *remote specification* construct. A remote specification is a specification that has been removed from its defining context to
gain overview. It is similar to calling and defining a procedure, but is “called” from exactly one place (the defining context) using a reference. In other words, there is a one-to-one correspondence between the reference and the remote specification. This can be compared to a circuit diagram that is presented as a block diagram, where the blocks are described on separate pages as circuits with electronic components.
By using remote specifications, the graphical and textual representation forms can be mixed at the discretion of the user. In these notes:

- Examples will mainly use the remote kind of specification, and
- We will be focusing on the graphical syntax.

System Specification

Everything that is to be specified is part of the system specification in SDL. The system delineates the part of the universe that is to be specified. The use of remote specification makes it possible to look at different hierarchy levels separately. These reflect different levels of abstraction within an SDL specification.

In this section, only the top level of the hierarchy or abstraction of an SDL specification is covered. At this level, a very abstract view of the system is shown. This aids in giving a first overview of the specification without getting into unnecessary details.

A top-level system specification describes a set of blocks that interact with each other and the environment of the system via channels. This interaction is performed by means of signals that may have additional information in the form of parameter values.
Within a channel, signals experience a nondeterministic delay. This implies that a specification cannot influence the delay of signals in a channel, and that no assumptions may be made about the delay. However, the ordering of signals is always preserved.

The environment of the system is the rest of the universe and is expected to act in an SDL-like fashion as far as interactions with the system are concerned and to obey the constraints imposed by the system specification. Other than this we know nothing of the environment of an SDL system.

Although it is eventually necessary to specify the behaviour of an SDL system, this is done at system level. Rather, this is deduced indirectly from the behaviour of the blocks constituting the system and their communication with each other and the environment via channels. Even though the dynamic description is hidden at this stage, a good specification does give a lot of information regarding the system at this level.

**Rules**

- A system specification must contain at least one block specification.
- Predefined data types, called predefined sorts in SDL, are considered to be defined at system level.
• All channels, signals and sorts used at system level must also be specified at this level.
• Declarations at system level also apply to the environment.

An SDL system may be a closed system, as can be inferred from the first rule above. An open SDL system must, of course, be able to receive signals from or send signals to the environment.

A system type has a name for the purpose of identification. This is the system name, which is part of the system heading in the graphical representation. When using the graphical representation, the system specification, called the system diagram is as shown in the diagram below. Note that the system heading is on the top left corner of the frame symbol.

![Diagram of a system specification]

Syntax of a system specification
The system specification contains two distinct parts. These are the *declarations area* and the *block interaction area*.

**The System Declarations Area**

The declarations part contains the declaration of:

- new sorts (data types),
- system level declarations (variables, definitions etc.)
- names and data types of signals,
- *signal lists* - a number of signals may be placed in a list and assigned a list name.

The system declarations area can be split into more than one subarea, which may be placed in different locations within the system diagram.

**The Block Interaction Area**

The block interaction area shown in the diagram specifies the blocks and their interactions via channels. Channels may also extend up to the system boundary (frame symbol) when they connect a block to the environment of the system. Note that channels must be described completely at this level. The information which must be provided includes

- the name of the channel,
- whether the channel is uni- or bidirectional,
• the signals sent over it in each direction (either a signal list or the names of all signals must be provided for each direction).

The graphical representation of a channel and its signals is as follows:

```
[<signal list>]
<channel name>
```

Using the remote type of specification, the block and its connections to other block must be shown at system level without showing what is within the block. The block name here is a reference for the actual (remote) block specification. The remote block diagram is anywhere outside the system diagram.

A typical block interaction area would therefore look as follows (with the contents of the block shown elsewhere):
Unlike the systems declaration area, the block interaction area must form a coherent area on one page. If there is not enough space left on the page for both block and declaration areas or if the user chooses to do so, the system diagram can be split into a number of pages. In this case, the frame symbol is repeated with the same system heading on the top left corner.
EXAMPLE

Develop a top-level system specification for a vending machine with the following system concept description:

The vending machine contains three products - toffee, chocolate and chewing gum. It has two slots. One slot accepts money entered by the user. Only 10p, 50p and £1 coins are accepted. Change and any unacceptable coinage are returned to the user at the other slot. A series of buttons allow the user to select which product is desired. Another button allows the user to cancel the transaction. There is a display on the machine which serves a number of functions such as:

1. It displays the amount to be paid - when money is entered and accepted, this value decreases accordingly until all required money has been entered.

2. If the product is unavailable, "Empty" is written to the display.

3. If no change is available, the display tells the user to use exact change only.

4. If maintenance is required, the string "Out of Order" is written on the display.

Finally, there is a tray at which the requested product is dispensed once the money has been entered. If stocks are low or if the coin tray is full, maintenance is requested. The system remains out of order until manually reset by the maintenance technician.
The order of execution of a request is:

- The user presses the button for the desired product,
- Stocks of the required product are checked - if none are available, the user is informed and maintenance requested. Else,
- The machine decides if change is available - if not, a "Use Exact Change Only" message written to display,
- The required amount to be paid is written to the display,
- The user must then enter a coin within a given time frame. If they do not, the machine resets and the screen blanks. Else,
- A coin is entered, accepted, and the amount remaining to be paid is written to the display. If the coin is unacceptable, it is immediately returned and the display does not change.
- The above step is repeated until the required money has been entered,
- The relevant product is released,
- Any change is released,
- The machine resets.

At any stage, the user may choose to cancel the transaction by pressing the cancel button. When this happens, any money entered is returned to the user and the machine is reset.

The system specification for the vending machine is as follows:
As shown, the system is called `vending_machine` and contains two blocks `Dialogue` and `Ware_Mgr`. The block `Dialogue` is responsible for communication with the user and for the acceptance and management of coins. The other block, called `Ware_Mgr` is responsible for the toffee and chocolate. The channels `Inp_C` and `Pay` carry the signals for communication with the user, where the coins are also represented by signals. An
internal channel $Sync$ connects the two blocks, whereas the channel $Out_W$ outputs the desired items to the user, again modelled by signals. The signal lists show which signals are carried by the channels in a specified direction. For example, channel $Inp_C$ carries the signals $button$ and $undo$ from the user (in the environment) to the block $Dialogue$ and the signals $display1$, $display2$, $overpay$ and $empty$ in the opposite direction. The channel $Out_W$ carries signals only in one direction (to the environment).

This example shows that it is possible to gain a first insight into the behaviour of the system. Appropriately chosen signal, block and channel names give some indication of the functions carried out within the system, albeit informally. A good choice of names does not improve automatic processing of the specification, but certainly serves to aid human readers in getting an intuitive understanding of the system. However, to get a clear picture of the reaction of the system to certain signals, this level does not suffice. What is needed is an understanding of the blocks concerned, which is described in the next section.
For comparative purposes, the textual syntax of the system specification would look as follows:

```plaintext
system vending_machine;

signal
  coin_10, coin_50, coin_100, coin_x, button, undo,
  display1, display2, overpay, empty, status, complete,
  maint, exists, paid, coinerr, toffee, choc, gum;

block Dialogue referenced;
block Ware_Mgr referenced;
channel Inp_C
  from Dialogue to env
    with display1, display2, overpay, empty;
  from env to Dialogue
    with button, undo;
endchannel Inp_C;
channel Pay
  from env to Dialogue
    with coin_10, coin_50, coin_100, coin_x;
endchannel Pay;
channel Flush
  from Dialogue to env
```
with coin_10, coin_50, coin_100, coin_x;

endchannel Flush;

channel Sync

from Dialogue to Ware_Mgr

with exists, paid, coinerr;

from Ware_Mgr to Dialogue

with status, complete, maint;

endchannel Sync;

channel Out_W

from Ware_Mgr to env

with toffee, choc, gum;

endchannel Out W;

endsystem vending_machine;