1. IMPORTANT LANGUAGE FEATURES

1.1 Realtime Features

A program for on-line control or evaluation of a technical process has to react rapidly on spontaneously received information of the process or on timely results. Out of this reason it is not sufficient to arrange and go through the various divisions of the program sequentially, that means in timely unchanged sequence. It is of importance that the more or less complex automation problem has to be divided into problem-justified components of different states of urgency and that the program structure must be fitted to this problem structure. This causes the existence of independent program elements for sub-problems ready to be solved timely sequentially among other problems (e.g. procedures). However, there also arise independent program elements for sub-problems, which based on a timely not determined cause (e.g. disturbance in the process under control) have to be solved immediately parallel to other problems.

The execution of such a program element is called "task", for determination of urgency such tasks can be provided with priorities.

Concerning the definition and combination of tasks - with regard to the technical process - PEARL offers the following possibilities:

- Definition of tasks, e.g.
  
  SUPPLY: TASK PRIORITY 2;
  taskbody (definitions, statements)
  END;

- Start (activation), e.g.
  
  ACTIVATE SUPPLY;

- Termination, e.g.
  
  TERMINATE PRINTING;

- Suspension, e.g.
  
  SUSPEND STATISTICS;
According to demands of automation problems some of these statements can be scheduled for their (repeated) execution, e.g. scheduled for the case of time entrance, the end of a duration or the occurrence of an interrupt:

```plaintext
WHEN READY ACTIVATE SUPPLY;
```

(Meaning: Each time when the interrupt READY occurs, the task SUPPLY has to be activated.)

Schedules can also determine the timely periodical start:

```plaintext
AT 12:0:0 EVERY 30 MIN UNTIL 15:0:0 ACTIVATE PROTOCOL;
```

As far as certain actions do not interfere, different tasks execute their statements independently of each other. Sometimes however synchronization of two or more tasks is required, e.g. if a task periodically creates data for other tasks and puts them into a buffer. In this case the producer is not allowed to work faster than the consumer.

Synchronization problems of higher complexity arise, if a task must have exclusive access to a file (when writing), while others also participate simultaneously (when reading).

In order to solve such synchronization problems PEARL contains the synchronization primitives sema and bolt variables.