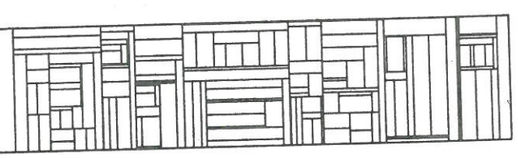
DELTA
Project Report No. 3

DESCRIPTION OF A MODEL OF A SINGLE HELIX POMATIA BRAIN NEURON AND AN ASSOCIATED NEUROPHYSIOLOGICAL EXPERIMENT

by
Morten Kyng
&
Birger Møller Pedersen

DAIMI PB-42
December 1974

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06-12 83 55



PREFACE

The work reported in this paper is a part of the DELTA Project (see ref. (1) and (2)). The DELTA Project has as one of its objectives to develop language tools for description of complex, interacting and dynamic systems. The first DELTA language proposal will appear as a publication from the Norwegian Computing Center this autumn.

For a short introduction to the basic concepts of the DELTA language we refer to the appendix.

In this report the authors use the DELTA language (as of September 1974) to describe a biological system in interaction with an experimental set-up. The system selected consists of a single brain neuron of Helix Pomatia (the edible snail) and a neurophysiological experimental equipment.

The description comprises the knowledge available and relevant to the hypotheses to be tested in the experiment, some hypotheses as to the nature of feedback mechanisms present in this type of neuron and the aspects of the instrumental set-up which directly interact with the neuron during the experiment.

The description is to be used for three purposes:

1. as a tool for communicating to other neurophysiologists the nature of the model and the results gained by the experiment.

2. as a precise formulation of the neuron model and the experiment, to be used by a computer specialist in the formulation of a computer simulation model.

3. as a case study in testing the usefulness of the first DELTA language proposal as a tool for the description of this type of biophysical models.

The experiments have been designed and conducted by Helge Rasmussen under the direction of professor I. Engberg, both of them from Institute of Physiology, University of Aarhus.

The computer simulation program is to be programmed by Mogens Nielsen, lecturer at Department of Computer Science, University of Aarhus.

The report is the result of a close cooperation between Helge Rasmussen, Mogens Nielsen and the authors, with Kristen Nygaard (one of the authors of the DELTA language) as a consultant.

The DELTA reports being issued from various institutions are provided with a common, consecutive DELTA Project Report (or Working Note) numbering. This report is published as DELTA roject eport No. 3 and as DAIMI publication No. 42.

Aarhus, December 1974

Morten Kyng &
Birger Møller Pedersen

CONTENTS

1. The system and its main components

The presentation in this chapter is aimed at giving the reader an understanding of the main properties of the system. Details are postponed till Chapter 2.

The system described covers the whole physical experiment examining one single neuron. We confine our model to one neuron and its immediate environment, and do not describe this environment in terms of neurons. Firstly because the experiment is designed to investigate the behavior of one single neuron. Secondly because otherwise we would be forced either to use an extremely simple model for the single "environment neurons" or to restrict the description of the environment to cover two or three neurons and their connections. The system consists of the following main components (figure 1.1):

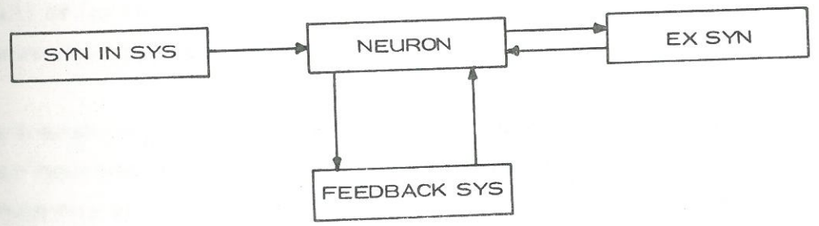


Figure 1.1
The main components of the system

SYN IN SYS:

The synaptic input system. This system represents the input which the neuron, independent of its own activity, gets from other neurons in the brain of the snail.

NEURON:

The single neuron under examination. It works on input from the synaptic input system and the feedback system, and dependent on these and properties of the neuron itself, it emits impulses at intervals - these impulses are called firings.

FEEDBACK SYS:

The feedback system, which comprises the feedback the neuron receives as a result of its firings.

EX SYS:

The experimental system, which comprises the system connected to the neuron during the experiment.

It should be remarked that if the neuron receives synaptic input as a result of its firings, through loops of other neurons, these loops are represented by FEEDBACK SYS.

The transmission of impulses from the neuron under investigation to other neurons is without significance in the experiment, unless these neurons are part of the feedback system. If this is the case the interaction is described by the interaction between the NEURON and the FEEDBACK SYS objects, otherwise the impulse transmission is not covered by the system description.

1.1 SYN IN SYS - synaptic input system

When a neuron fires, an impulse travels along a narrow (possibly branched) extension of the cell, named an axon. The end points of the axon are the regions known as the synapses. Each synapse impinges onto a single neuron. The impulse is transmitted to the neuron through the synapse (fig. 1.2.).

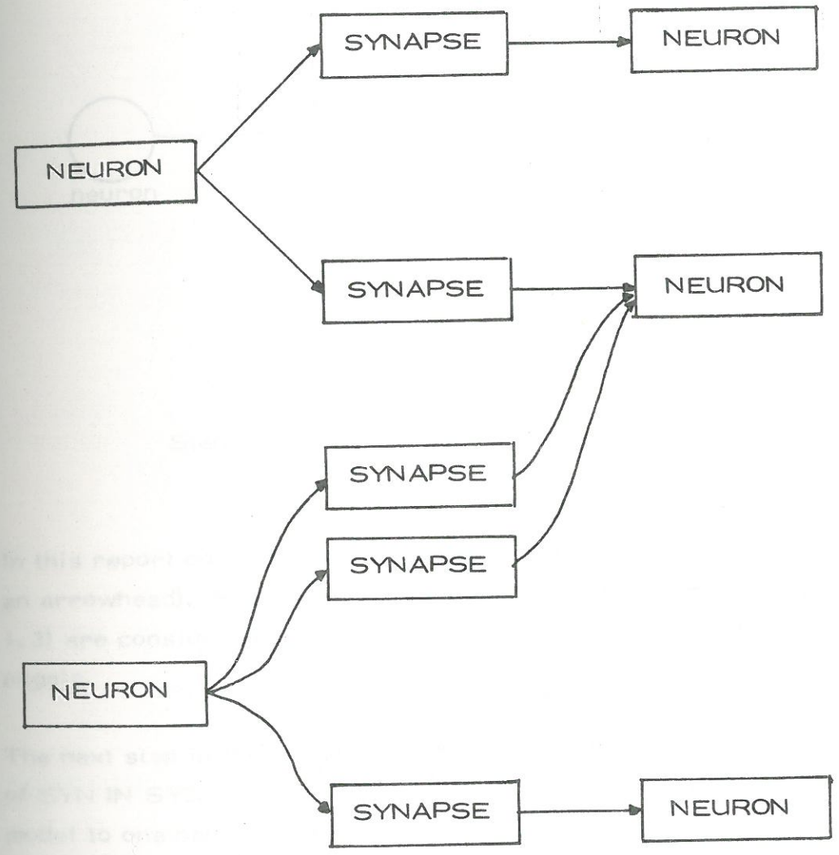


Figure 1.2
Connections between NEURONs through SYNAPSEs

In neurophysiology the standard way of drawing diagrams of neuron networks is illustrated by figure 1.3, where the neuron on the left transmits to the one on the right.

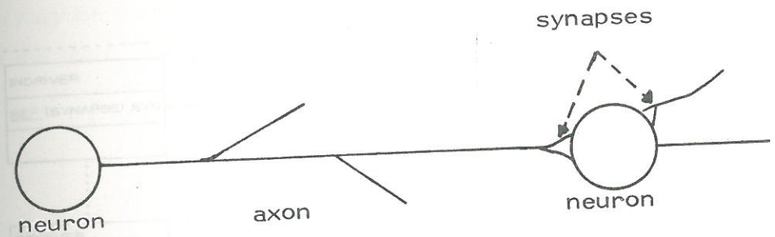


Figure 1.3
Standard diagram of neuron network

In this report connections are drawn by directed lines (indicated by an arrowhead). Synapses (indicated by the symbol ⟝ in figure 1.3) are considered as system components and represented by rectangels.

The next step in the model development consists of the splitting up of SYN IN SYS into subsystems (figure 1.4). Since we confine our model to one neuron and its immediate environment, we have decided to represent the input part of the environment by a set of input drivers (represented in the model by INDRIVER objects) firing impulses into the neuron through synapses (represented in the model by SYNAPSE objects):

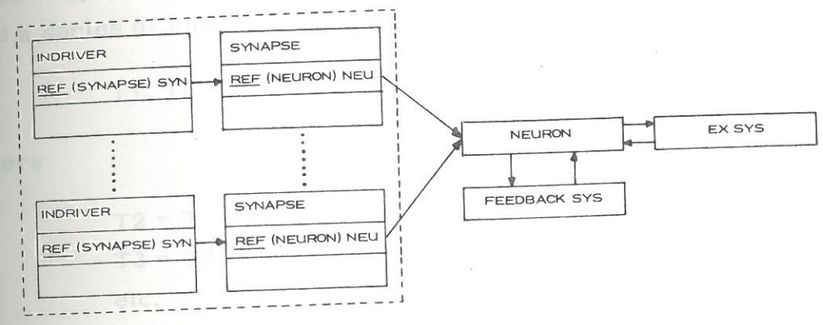


Figure 1.4
Splitting in SYN IN SYS

INDRIVER:

The INDRIVER objects of our model describes the behavior of the neurons of the total snail-brain system which are producing input to our limited single-neuron experiment system.

The purpose of introducing the INDRIVER objects thus is to feed the SYNAPSE objects with streams of impulses, simulating the environment of the single-neuron system.

Since all impulses to one synapse are similar in their characteristics, each stream may be characterized by a time series representing the times of arrival of impulses

T1, T2, T3, ...

which may be transformed to a series consisting of a starting time and a series of intervals

T1, I1, I2, I3, ...

where

T2 = T1 + I1,
T3 = T2 + I2,
etc.

The intervals I1, I2, ... may be regarded as a sequence of values of a stochastic variable which we may name

INTERVAL.

Since we do not know the distribution functions of the INTERVAL variables (one for each INDRIVER), they will be introduced as VIRTUAL REAL FUNCTION s. (REAL FUNCTION implies that the function, when applied, produces a real number and VIRTUAL implies that the exact nature of the function is not known or not decided upon on the level of generality used in this description).

In the following description of the INDRIVER objects:

- SYN is a name (reference) used to denote the SYNAPSE to which THIS INDRIVER object is connected.

- T NEXT is a variable whose current value always is the scheduled time of arrival of the next impulse taking on successively the values T1, T2, T3, ...

- RECEIVE IMPULSE (the actions performed when receiving an impulse) is a task procedure which is a part of the action repertoire of a SYNAPSE object.

SYN.RECEIVE IMPULSE implies that the RECEIVE IMPULSE procedure of the SYNAPSE object named SYN shall be executed. (SYN's RECEIVE IMPULSE task is to be executed, the genitive notation "'s" being substituted by ".").

```
CLASS INDRIVER:
  OBJECT BEGIN
    REF(SYNAPSE) SYN;
    REAL T NEXT;
    VIRTUAL  REAL FUNCTION INTERVAL;
    TASK BEGIN
      REPEAT
        (* ENFORCE SYN. RECEIVE IMPULSE UPON SYN;
           T NEXT:=TIME + INTERVAL;
           WHILE TIME < T NEXT IMPOSE {REST} *);
    END TASK;
  END OBJECT
```

This declaration describes a class of objects named INDRIVER. Each object in this class has a reference variable SYN, a real variable T NEXT, a function named INTERVAL and an action pattern consisting of a working cycle (REPEAT (* ..... *)) of three actions. An INDRIVER object

- interrupts its SYNAPSE object by enforcing the SYNAPSE object to execute the procedure RECEIVE IMPULSE
- determines the next time at which the SYNAPSE object shall be interrupted and
- rests until then.

The actions of the working cycle are then repeated.

SYNAPSE:

A SYNAPSE object rests until it receives an impulse from its INDRIVER.

The SYNAPSE then starts to transmit to the NEURON. The decay caused by the transmission is for two reasons without any significance in the experiment, and is therefore omitted in the model. Firstly, the delay is so small that it is impossible to measure it with the equipment used. Secondly, its effect is only the addition of a constant increment to each impulse arrival time, and the time interval pattern is not changed.

The strength of an impulse is constant for a given SYNAPSE and may be characterized by the value of a real variable, STRENGTH. Each SYNAPSE has its own individual value of STRENGTH.

When the interrupt RECEIVE IMPULSE has been carried out, the SYNAPSE object resumes its ongoing time-consuming "rest action", being in the state of inactivity:

CONTINUE {REST}

```
CLASS SYNAPSE:
  OBJECT BEGIN
    REF(NEURON) NEU;
    REAL STRENGTH;
    VIRTUAL  TASK(NEURON) PROCEDURE IMPULSE;
    COMMENT The task procedure IMPULSE changes the
            NEURON's ability to fire. The size of the
            change depends upon the value of the
            STRENGTH of THIS SYNAPSE and the
            internal state of the NEURON;
    TASK PROCEDURE RECEIVE IMPULSE:
      TASK BEGIN
        ENFORCE IMPULSE PUT IMPACT:=STRENGTH
                UPON NEU;
      END TASK;
    TASK BEGIN
      CONTINUE {REST};
    END TASK;
  END OBJECT
```

A formal description of the procedure IMPULSE will be given when the NEURON has been described in detail.

The IMPULSE task, described by the procedure, is executed by the NEURON named NEU and has to make use of the value of STRENGTH of the SYNAPSE. This is achieved by the SYNAPSE's assigning the value of STRENGTH to another real variable, IMPACT, available within the IMPULSE task (the description element PUT IMPACT:= STRENGTH).

While a SYNAPSE object rests, i.e. executing

```
CONTINUE {REST}
```

RECEIVE IMPULSE tasks from its INDRIVER may interrupt the SYNAPSE. We do not associate any resistance priority to this time-consuming "rest action". It has then the lowest priority value, NO,

and will be interrupted by all interrupting tasks independent of their power (priority value). Then we need not explicitly give the interrupt

ENFORCE SYN. RECEIVE IMPULSE UPON SYN

in the INDRIVER object a power to make sure that it will interrupt the time-consuming "rest action". That is, the interrupting task also has the priority value NO.

1.2 NEURON

The state of excitability of a neuron is characterized by the membrane potential (i.e. the potential difference between the inside and the outside of the membrane) and the firing threshold. The membrane potential is usually negative.
The neuron is activated when the membrane potential becomes larger than the firing threshold. The activation results in a firing, which consist of the sending of a short impulse along its axon, which connects it to other neurons through synapses. Since the experimental recording equipment records the impulse as a sharp peak (a "spike") on the display used, a firing is usually referred to as a spike.

The internal state of a neuron immediately after a spike is not well understood. However, no new firing will take place during a time interval which is named the absolute refractory period. The absolute refractory period gradually develops into the relative refractory period. In the beginning of this period a very large input may initiate a spike. As time passes by, the input required is diminished, until the normal processes within the neuron eventually initiate a new firing, even without any external input.

Since the potential outside the neuron is constant during the experiment the membrane potential, POT, may be considered as a sum of two components,

POT = increasing value of the background potential (BP)
+ effects of external inputs.

The external inputs may come from either the SYN IN SYS, the FEEDBACK SYS or the EX SYS.

The background potential, BP will be increasing towards a constant value, the background potential limit (BP LIMIT). (The background potential may be considered as a combination of the results of two processes: an afterhyperpolarization and a pacemaker or prepotential process. These two processes will not be discussed further).

The firing threshold, FT, will be decaying towards a constant value, the firing threshold limit (FT LIMIT).

The ongoing internal processes may be illustrated by figure 1.5., where the value of the real variable T LAST is the time of the last firing:

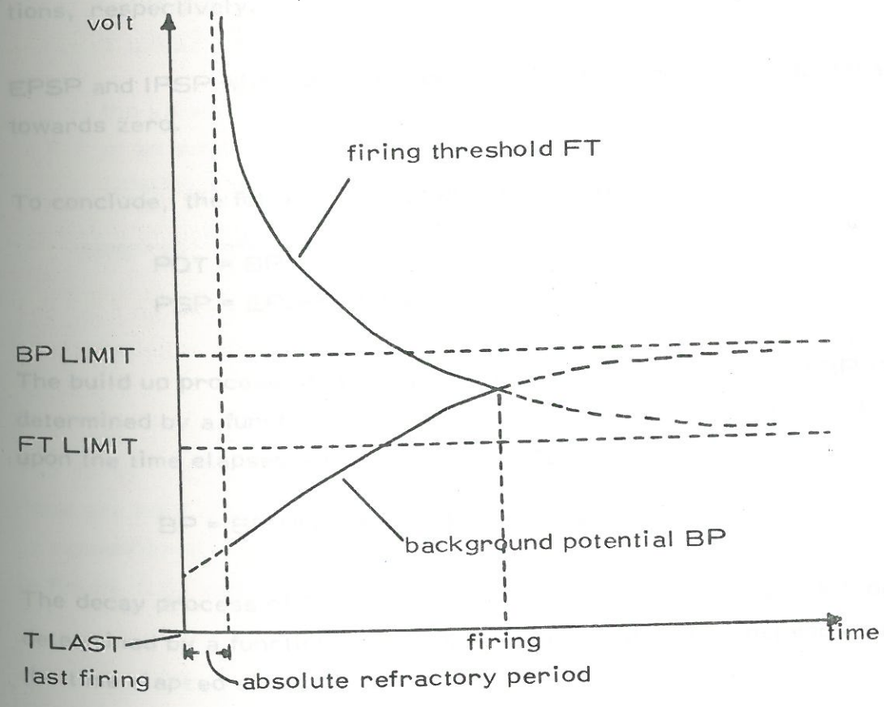


Figure 1.5
Ongoing internal processes of the NEURON

It follows that if a NEURON does not receive any external stimuli and if BP LIMIT>FT LIMIT, the result will be that the NEURON fires at a regular rate.

The external synaptic input may be split up in two components:

effects of external synaptic input =
postsynaptic potential = effects of impulses from excitatory synapses
- effects of impulses from inhibitory synapses.

The postsynaptic potential will be named PSP, the contribution of the excitatory synapses will be named excitatory postsynaptic potential (EPSP) and the contribution of the inhibitory synapses will be named inhibitory postsynaptic potential (IPSP). EPSP and IPSP are the total sums of the excitatory and inhibitory impulse contributions, respectively.

EPSP and IPSP will, when no additional inputs are received, decay towards zero.

To conclude, the following equations are valid:

POT = BP + PSP
PSP = EPSP - IPSP.

The build up process of BP is described by letting the value of BP be determined by a function, BP BUILDUP whose value is dependent upon the time elapsed since the last firing:

BP = BP BUILDUP(TIME - T LAST).

The decay process of FT is described by letting the value of FT be determined by a function, FT DECAY whose value also depends on the time elapsed since the last firing:

FT = FT DECAY(TIME - T LAST).

The decay process of FT and the build up process of BP are only interrupted by the firing of the NEURON. The values of FT and BP after a firing and the values of FT LIMIT and BP LIMIT are parts of the definitions of the functions.

The ongoing decay processes of EPSP and IPSP are interrupted by the firing of the NEURON and by the reception of exicitatory and inhibitory impulses, respectively. The decay process of EPSP is described by the function EPSP DECAY and that of IPSP by IPSP DECAY.

The value of EPSP (IPSP) is determined by the value of EPSP DECAY (IPSP DECAY) whose value depends upon

- the time elapsed since the last firing, TIME - T LAST
- the time elapsed since the last reception of an excitatory (inhibitory) impulse, TIME - T LAST EXC (TIME - T LAST INH)
- the value of EPSP (IPSP) at the time of the last reception of an excitatory (inhibitory) impulse, EPSP LAST (IPSP LAST):

EPSP = EPSP DECAY(TIME - T LAST,
TIME - T LAST EXC, EPSP LAST)

IPSP = IPSP DECAY(TIME - T LAST,
TIME - T LAST INH, IPSP LAST).

The decay of EPSP and IPSP is towards zero and the decay functions are both exponential, as illustrated by figure 1.6. (It should be remembered that the IPSP DECAY, which is given positive values in figure 1.6, contributes negatively to PSP).

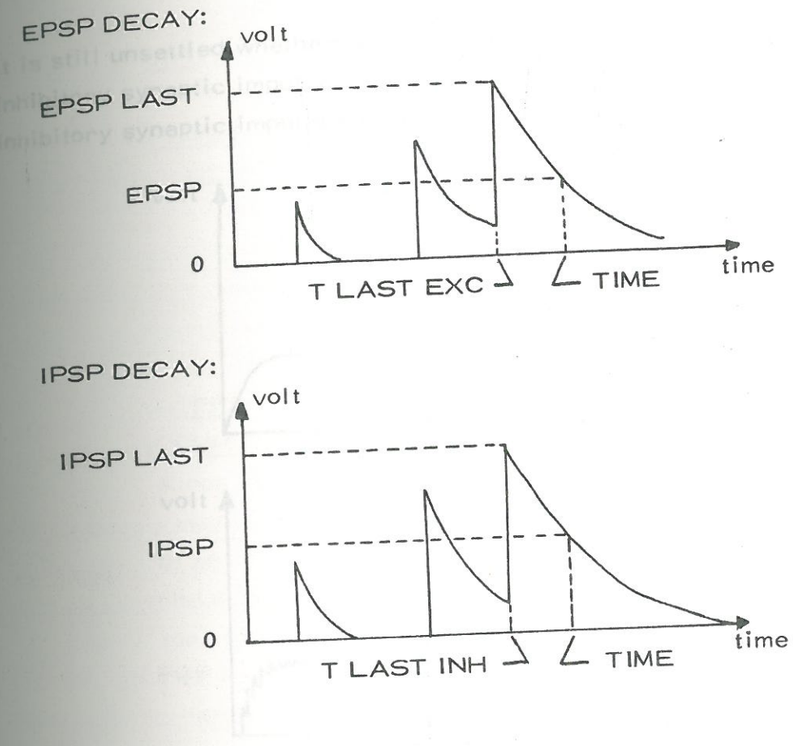


Figure 1.6
Decay of postsynaptic potential

The time constants for EPSP DECAY and IPSP DECAY are, in the neurons under investigation, different.

There is an increase of the values of the time constants during both the absolute and the relative refractory period. This causes a faster decay of impulses when they are received shortly after a firing.

There are possibly two kinds of time constants of IPSP DECAY. One is of the same magnitude as the time constant for the EPSP DECAY, the other is considerably larger, characterizing the decay of an inhibition of long duration (ild).

It is still unsettled whether an ild is due to one large elementary inhibitory synaptic impulse or due to a summation of several normal inhibitory synaptic impulses (figure 1.7):

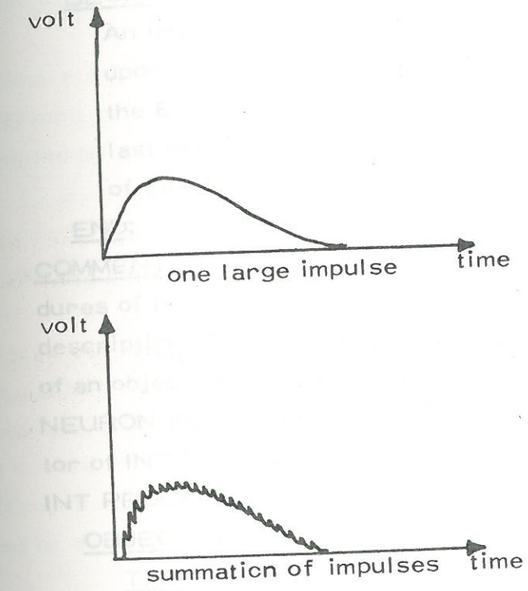


Figure 1.7
Inhibition of long duration

As the exact nature of ild is without direct significance in the experiment we choose for simplicity the latter mechanism: Summation of inhibitory synaptic impulses. It will thus be a property of the input drivers that they may produce series of inhibitory synaptic impulses similar in effect to an ild.

In the following description of the neuron the decay functions are introduced as virtual quantities. This is done because the exact nature of the functions is not known and because we want the description to be valid regardless of which of the various possibilities that are selected for mination later on.

```
NEURON:
  OBJECT BEGIN
    REAL POT, PSP, T LAST, T LAST EXC, T LAST INH,
         EPSP LAST, IPSP LAST;
    VIRTUAL REAL FUNCTION FT DECAY, BP BUILDUP,
                          EPSP DECAY, IPSP DECAY;

    PROCEDURE FIRE:
      BEGIN
        An impulse is generated. This enforces interrupts
        upon the FEEDBACK SYS and the ELECTRODE of
        the EX SYS (see later). T LAST (the time of the
        last previous firing) is given the value of the time
        of this firing (the current value of TIME);
      END;
    COMMENT Thedeclarations of the variables and proce-
    dures of the NEURON are given above. Below follows a
    description of its ongoing internal processes by means
    of an object descriptor. Throughout the existence of the
    NEURON the relations described by the property descrip-
    tor of INT PROCESS are imposed upon the NEURON;
    INT PROCESS:
      OBJECT BEGIN
        TASK BEGIN
          CONTINUE {PSP=EPSP DECAY - IPSP DECAY,
                    POT=PSP + BP BUILDUP};

        END TASK;
      END OBJECT;
    COMMENT The total activity of the neuron thus consists of
    the ongoing internal processes modulated by the additional
    actions of the working cycle and the actions enforced by
    the interrupts stemming from the reception of impulses;

    TASK BEGIN
      REPEAT
        (* WHILE POT <FT DECAY IMPOSE {REST};
          FIRE;
          WHILE absolute refraction obtains
            IMPOSE {restoration of spike initiation mechanism}*
    END TASK;
  END OBJECT
```

When POT > FT DECAY the NEURON fires, and this is done by executing the procedure FIRE. Since both BP BUILDUP, FT DECAY, EPSP DECAY and IPSP DECAY are functions of TIME - T LAST, the execution of the procedure FIRE implies that these functions are forced into a new cycle, starting from TIME - T LAST = 0.

It should be stressed that the duration of a firing is so small that we have described it as an instantantaneous action in our model.

The two time consuming actions in the NEURON object both have resistance priority value NO. When executing these actions it will thus be interrupted by every interrupting task.

Excitatory and Inhibitory Synapses

As mentioned above the neuron may receive two kinds of input from synapses, called excitatory and inhibitory impulses. A synapse is either excitatory or inhibitory, and this will be described by introducing two subclasses of SYNAPSE with two different IMPULSE procedures. (Only these procedures are declared and no further properties are added to those already described in CLASS SYNAPSE).

In the excitatory synapses IMPULSE affects EPSP by changing the value of the real variable EPSP LAST, and in the inhibitory synapses IMPULSE affects IPSP by changing the value of IPSP LAST. In both cases the size of the change depends on the value of IMPACT and of the value of POT. This dependency is described by the functions EXC and INH. These functions are declared as VIRTUALs as were the decay functions of the NEURON.

IMPULSE also has to reset the value of T LAST EXC (T LAST INH) to the value of TIME at the moment the exitatory (inhibitory) impulse arrives.

```
CLASS EXC SYNAPSE:
   SYNAPSE OBJECT BEGIN
                VIRTUAL REAL FUNCTION EXC;
                TASK (NEURON) PROCEDURE IMPULSE:
                   TASK BEGIN
                      REAL IMPACT;
                      NEU. EPSP LAST:=NEU. EPSP DECAY +
                                        EXC(IMPACT, NEU. POT);
                      NEU. T LAST EXC:=TIME;
                   END TASK;
                END OBJECT


CLASS INH SYNAPSE:
   SYNAPSE OBJECT BEGIN
                VIRTUAL REAL FUNCTION INH;
                TASK(NEURON) PROCEDURE IMPULSE:
                   TASK BEGIN
                      REAL IMPACT;
                      NEU. IPSP LAST:=NEU. IPSP DECAY +
                                        INH(IMPACT, NEU. POT);
                      NEU. T LAST INH:=TIME;
                   END TASK;
                END OBJECT
```

3. 3 FEEDBACK SYS - the feedback system

The feedback system consists of those mechanisms which make the output of the neuron dependent on previous firings.

The nature of feedback effects in the neurons under investigation is not well understood. The effects may be given various interpretations as e. g. the input to the neuron through a loop of other neurons or the result of processes within the neuron itself.

One purpose of the computer simulation program to be developed is to observe the effects of the various assumptions on the feedback-mechanism and compare these effects with those observed in the neurophysiological experiment.

In our description no definite choice is made, except that the effects upon the NEURON are introduced by the actions of a FEEDBACK SYS object. It is left open to the reader to regard the neuron as a union of a NEURON and a FEEDBACK SYS object, or as a NEURON object interacting with outside system components (e.g. neuron loops) represented by a FEEDBACK SYS object.

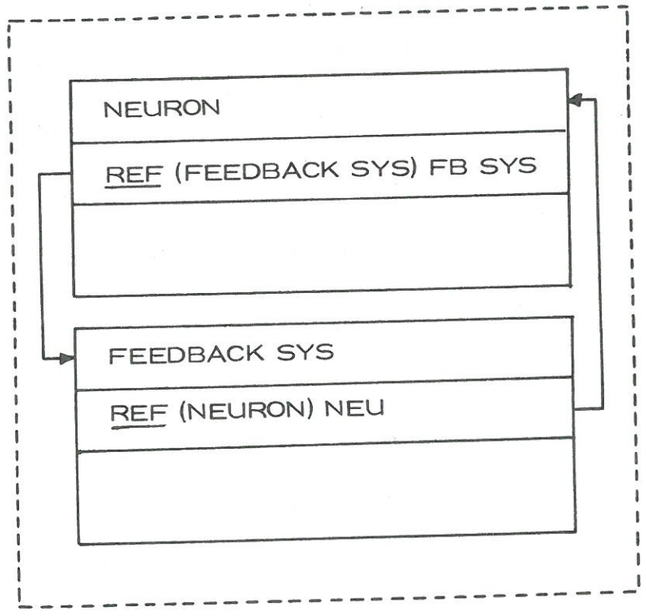


Figure 1.8
NEURON and FEEDBACK SYS
possibly representing a single neuron

The feedback effects are dependent upon earlier firings. This implies that the FEEDBACK SYS object must contain mechanisms which may reproduce these effects, regardless of which physical interpretation is chosen.

The FEEDBACK SYS object thus has to contain some information about the streams of impulses from the NEURON.

Since all impulses from the neuron are similar in their characteristics, the stream of impulses may be characterized by the time of the last impulse and the series of intervals between earlier firings (the interspike intervals).

FEEDBACK SYS is only capable of holding a limited amount of information about the impulse stream of the NEURON. This information is held in what we may call the feedback memory.

The effects of FEEDBACK SYS are obtained by enforcing the task procedure FEEDBACK upon the NEURON. The exact nature of this procedure will not be discussed in this report. The FEEDBACK procedures effects on the NEURON are left open by introducing the procedure as a virtual quantity. It should be remarked that if we interpret the feedback effect as the result of processes within the neuron itself the task procedure FEEDBACK will possibly affect FT DECAY and BP BUILDUP.

The next time at which the NEURON will be interrupted by FEEDBACK is denoted T NEXT FB. T NEXT FB is determined by the contents of feedback memory. It is adjusted whenever the contents are changed or when a FEEDBACK interrupt has been enforced upon the NEURON.

```
FEEDBACK SYS:
  OBJECT BEGIN
    COMMENT Feedback memory is a data structure capable
            of holding a limited set of real values repre-
            senting times of arrival of impulses from the
            NEURON;
    REF (NEURON) NEU;
    REAL T NEXT FB;
    TASK PROCEDURE RECEIVE IMPULSE:
      TASK BEGIN
        adjust feedback memory and T NEXT FB;
      END TASK;
    COMMENT The procedure RECEIVE IMPULSE is
            executed when receiving an impulse from
            the NEURON;
    VIRTUAL TASK (NEURON) PROCEDURE FEEDBACK;
    TASK BEGIN
      REPEAT
        (*WHILE TIME < T NEXT FB IMPOSE {REST};
        ENFORCE FEEDBACK UPON NEU;
        adjust T NEXT FB*);
    END TASK;
  END OBJECT
```

The time consuming action in FEEDBACK SYS has resistance priority NO. This implies that the FEEDBACK SYS object will be interrupted by an impulse from the NEURON when it is executing

```
WHILE TIME < T NEXT FB IMPOSE {REST}
```

1.4 EX SYS - the experimental system

The next step in the model development is the decomposition of EX SYS into the experimenter and the experimental equipment consisting of an electrode and recording instruments collecting results (figure 1.9):

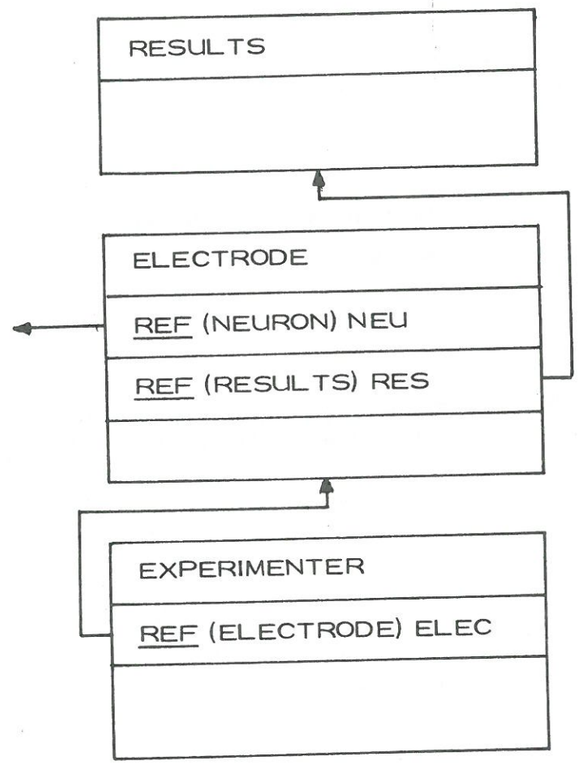


Figure 1.9
Splitting of EX SYS

At the beginning of an experiment the electrode (represented in the model by an ELECTRODE object) is impaled into the neuron by the experimenter (represented in the model by an EXPERIMENTER object). During the experiment the electrode both transmits the spikes of the neuron to the result collecting equipment and is used by the experimenter to change the potential of the neuron.

When the electrode is not acting upon the neuron it rests until it is interrupted by a firing of the neuron. It then transmits an impulse to the result-collecting recording instruments (represented in the model by a RESULTS object). The times of the firings are recorded, and statistics of the firing pattern are prepared (e. g. serial correlation coefficients).

In the following description of the ELECTRODE, RECEIVE FIRING is a task procedure enforced upon the ELECTRODE object by the NEURON when it fires.

When the ELECTRODE executes that procedure, this in turn enforces the task procedure RECEIVE FIRING of the RESULTS object to be executed by RESULTS.

(It should be remarked that in this first description of the ELECTRODE we do not include the possibility of the ELECTRODE to act upon the NEURON).

```
ELECTRODE:
   OBJECT BEGIN
      REF(RESULTS) RES;
      REF(NEURON) NEU;
      TASK PROCEDURE RECEIVE FIRING:
         TASK BEGIN
            ENFORCE RES. RECEIVE FIRING UPON RES;
         END TASK;
      TASK BEGIN
         CONTINUE {REST};
      END TASK;
   END OBJECT;
```

```
RESULTS:
   OBJECT BEGIN
      TASK PROCEDURE RECEIVE FIRING:
         TASK BEGIN
            update statistics;
         END TASK;
      TASK BEGIN
         CONTINUE {REST};
      END TASK;
   END OBJECT
```

The time consuming "rest actions" in the ELECTRODE object and the RESULTS object both have resistance priority NO and will thus be interrupted by every interrupting task.

During the experiment the main task of the experimenter is to manipulate the potential of the neuron through the electrode. He has two possibilities:

1. To change the level of the potential for a longer period of time (and then reset it to the former level again); this is called to polarize the neuron, (hyperpolarize if the level of the potential is lowered - depolarize if the level is raised).

   By hyperpolarizing the neuron it is hoped to obtain information about the nature of the feedback mechanisms. This is possible due to certain properties of the neuron and the feedback system:

   - if the lowering of the potential is sufficiently large the neuron will not fire during the hyperpolarization
   - the feedback at a given moment only depends upon the firing pattern of the neuron in a preceding limited period of time.

This implies that if the hyperpolarization is sufficiently large and is held for a sufficiently long period, the feedback effect on the neuron immediately after a hyperpolarization is a result of a period of time with no firings. The change in the firing pattern, as firings again affect the feedback to the neuron after a hyperpolarization, will thus possibly give valuable information about the feedback mechanisms.

2. To send impulses very similar in effect to excitatory and inhibitory synaptic impulses.

In the model we describe the fact that the EXPERIMENTER through the ELECTRODE is able to manipulate the potential, POT, of the NEURON by adding an electrode contribution, ELEC CONTRIBUTION, to POT:

POT=BP + BP BUILDUP + ELEC CONTRIBUTION

The size of the electrode contribution to the potential depends upon the strength with which the electrode is made to act upon the neuron, ELEC STRENGTH. The EXPERIMENTER controls the size of ELEC CONTRIBUTION by changing the value of ELEC STRENGTH of the ELECTRODE. The effects of the EXPERIMENTER's instantaneous, discontinous changes of the value of ELEC STRENGTH are continous changes of the value of ELEC CONTRIBUTION (Figure 1.10):

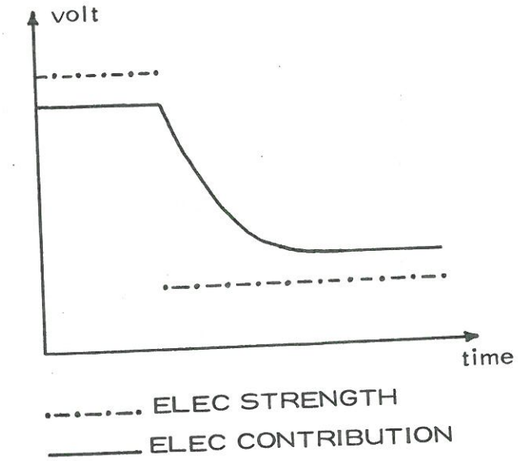


Figure 1.10
The effect of change of ELEC STRENGTH
upon ELEC CONTRIBUTION

ELEC CONTRIBUTION is besides ELEC STRENGTH, a function of the value of ELEC CONTRIBUTION immediately before the last change of ELEC STRENGTH, called ELEC CON LAST, and of the time elapsed since the last change of ELEC STRENGTH, TIME - T LAST CHANGE. This implies that

the electrode contribution to the potential
= ELEC CONTRIBUTION(ELEC. ELEC STRENGTH,
ELEC CON LAST,
TIME - T LAST CHANGE).

We have in the ELECTRODE object introduced a real variable, ELEC STRENGTH, representing the strength with which the electrode may act upon the neuron, and we have supplemented the description

of the NEURON by ELEC CONTRIBUTION. The following description of the NEURON contains only the modifications caused by the introduction of ELEC CONTRIBUTION:

```
NEURON:
   OBJECT BEGIN
      :
      VIRTUAL REAL FUNCTION ELEC CONTRIBUTION;
      :
      INT PROCESS:
         OBJECT BEGIN
            TASK BEGIN
               CONTINUE {PSP=EPSP DECAY - IPSP DECAY,
                         POT=PSP + BP BUILDUP
                             + ELEC CONTRIBUTION};
            END TASK;
         END OBJECT;
         :
   END OBJECT
```

ELEC CONTRIBUTION and ELEC STRENGTH are usually both equal to zero, (that is when the NEURON is not effected by the EXPERIMENTER through the ELECTRODE).

A polarization is obtained by changing the value of ELEC STRENGTH of the ELECTRODE. The EXPERIMENTER may do this by enforcing the task procedure ELEC STRENGTH CHANGE upon the ELECTRODE object. ELEC STRENGTH CHANGE is a task procedure of the ELECTRODE and it

- assigns the current value of ELEC CONTRIBUTION of the NEURON to ELEC CON LAST
- assigns the current value of TIME to T LAST CHANGE (the variable holding the time of the last change of ELEC STRENGTH)

- changes the value of ELEC STRENGTH to the value of SIZE (the value of SIZE is set by the EXPERIMENTER when enforcing ELEC STRENGTH CHANGE upon the ELECTRODE).

```
ELECTRODE:
   OBJECT BEGIN
      REF(NEURON) NEU;
      REAL ELEC STRENGTH;
      :
      TASK PROCEDURE ELEC STRENGTH CHANGE:
         TASK BEGIN
            REAL SIZE;
            NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
            NEU. T LAST CHANGE:=TIME;
            ELEC STRENGTH:=SIZE;
         END TASK;
      :
   END OBJECT
```

The experimenter determines the strength and the duration of a polarization. When determining the strength he may take into account experience from earlier polarizations (e. g. if there have been firings during a hyperpolarization the experimenter probably wants to enlarge the strength of the next hyperpolarizations). This implies that the EXPERIMENTER object must be able to access information in the RESULTS object. In our description this is reflected by declaring a reference variable, RES, in the EXPERIMENTER object, referring to the RESULTS object.

In the following description of the EXPERIMENTER

- POLAR STRENGTH is a function which determines the strength of the polarizations. Since it is without direct significance in our model how the EXPERIMENTER determines the quantity POLAR STRENGTH it is introduced as a VIRTUAL FUNCTION.

- POLARIZE is a procedure which when executed interrupts the ELECTRODE object by an ELEC STRENGTH CHANGE task with SIZE (of the procedure ELEC STRENGTH CHANGE) equal to the value of POLAR STRENGTH. That is, the value of ELEC STRENGTH of the ELECTRODE is changed to the value of POLAR STRENGTH.

```
EXPERIMENTER:
   OBJECT BEGIN
      REF(ELECTRODE) ELEC;
      REF(RESULTS) RES;
      VIRTUAL REAL FUNCTION POLAR STRENGTH;
      PROCEDURE POLARIZE:
         BEGIN
            ENFORCE ELEC.ELEC STRENGTH CHANGE
                    PUT SIZE:=POLAR STRENGTH
                    UPON ELEC;
         END;
      TASK BEGIN
         WHILE experiment going on DO
                 (* interact with the NEURON through the
                 ELECTRODE by using the procedure
                 POLARIZE and observe the effects on
                 the RESULTS object*);
      END TASK;
   END OBJECT
```

The EXPERIMENTER initiates a polarization by executing the procedure POLARIZE. It is terminated by an execution of POLARIZE with a value of POLAR STRENGTH which resets the potential to the former level. The duration of the polarization is determined by the working cycle of the EXPERIMENTER. Figure 1.11 shows a hyperpolarization.

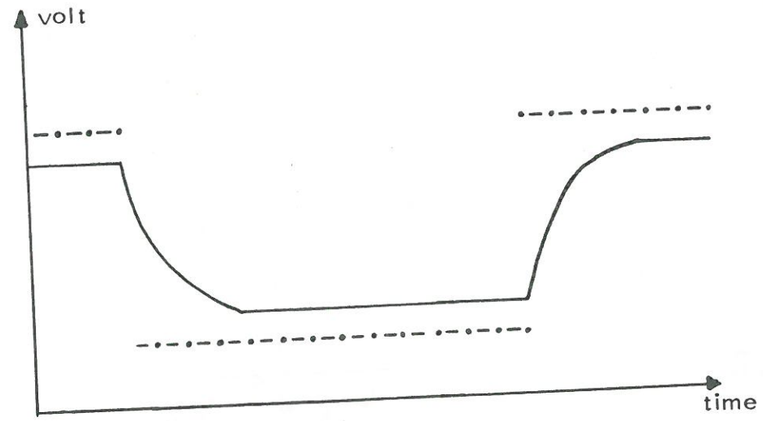


Figure 1.11
A hyperpolarization

The sending of an impulse resembling an exitatory or an inhibitory synaptic impulse to the neuron through the electrode is obtained by changing the value of ELEC STRENGTH for a very short period of time. The effect on ELEC CONTRIBUTION when changing the value of ELEC STRENGTH for a very short period is shown in figur 1.12. An impulse is characterized by

- SIZE, the size of the impulse, i.e. of the change of the value of ELEC STRENGTH
- T IMP, the time at which the impulse is initiated
- DURATION, the duration of the impulse.

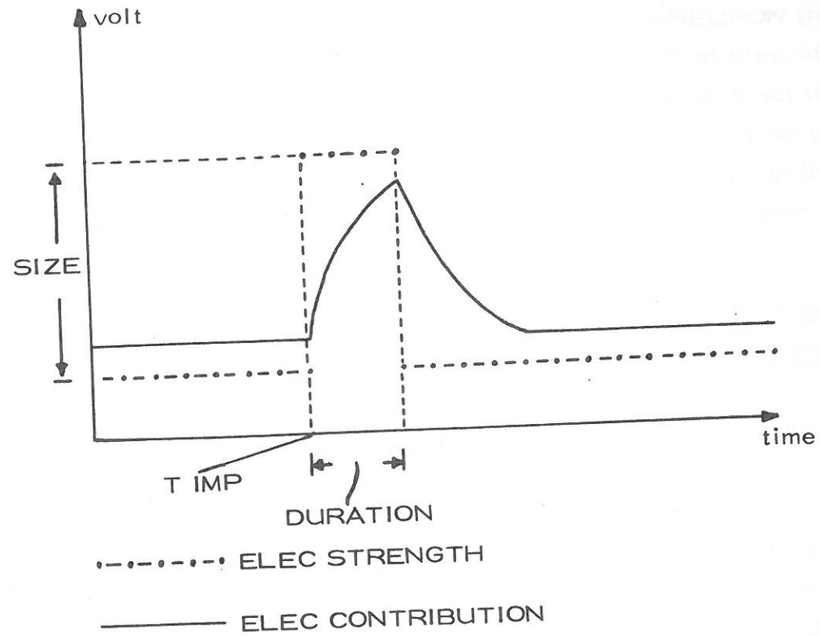


Figure 1.12

The effect of a short change of ELEC STRENGTH upon ELEC CONTRIBUTION. (In this case resembling the effect of an excitatory synaptic impulse).

The following statements describe the main actions of the ELECTRODE when sending impulses as mentioned above.

```
ELEC STRENGTH:=ELEC STRENGTH + SIZE;
WHILE TIME < T IMP + DURATION IMPOSE {REST};
ELEC STRENGTH:=ELEC STRENGTH - SIZE
```

For positive values of SIZE the effect will resemble that of an excitatory synaptic impulse and for negative values that of an inhibitory synaptic impulse.

If the NEURON fires while the ELECTRODE is executing the time consuming action stated above it shall be possible for the NEURON to interrupt the ELECTRODE.

On the other hand the sending of an impulse to the NEURON through the ELECTRODE is an indivisible action from the point of view of the EXPERIMENTER. This implies that he is not allowed to act upon ELEC STRENGTH during the ELECTRODE's execution of the time consuming action in question. This is prevented by declaring the priority HIGH in the ELECTRODE and then use it as resistance of the time consuming action:

WHILE TIME < T IMP + DURATION IMPOSE {REST}
RESISTANCE HIGH

The interrupts from the EXPERIMENTER, having power NO, is then unable to penetrate.

We declare the priority HIGH as RANKED; this implies that HIGH as power will penetrate HIGH as resistance. We thus enable the interrupts from the NEURON to penetrate by giving them the power HIGH.

The ELECTRODE sends an impulse resembling a synaptic impulse by executing the task procedure ELEC IMPULSE. In addition to the changing of the value of ELEC STRENGTH and the time consuming action it handles the necessary assignments to T IMP, T LAST CHANGE and ELEC CON LAST (as in the case of the task procedure ELEC STRENGTH CHANGE).

The EXPERIMENTER makes the ELECTRODE send an impulse by executing the procedure EXP IMPULSE, which enforces the task procedure ELEC IMPULSE upon the ELECTRODE object. The EXPERIMENTER determines the size of the impulse by the VIRTUAL REAL FUNCTION IMP SIZE and the duration of the impulse by the VIRTUAL REAL FUNCTION IMP DURATION. For positive values of IMP SIZE the impulses resemble excitatery synaptic impulses and for negative values of IMP SIZE they resemble inhibitory impulses. The functions IMP SIZE and IMP DURATION will not be discussed further.

The EXPERIMENTER's ability to send impulses through the ELECTRODE implies the following addition to the description of

- the ELECTRODE:

```
PRIORITY HIGH RANKED;
TASK PROCEDURE ELEC IMPULSE:
   TASK BEGIN
      REAL SIZE, T IMP, DURATION;
      NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
      NEU. T LAST CHANGE:=TIME;
      T IMP:=TIME;
      ELEC STRENGTH:=ELEC STRENGTH + SIZE;
      WHILE TIME < T IMP + DURATION IMPOSE {REST}
                                   RESISTANCE HIGH;
      NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
      NEU. T LAST CHANGE:=TIME;
      ELEC STRENGTH:=ELEC STRENGTH - SIZE;
   END TASK
```

- the EXPERIMENTER:

```
VIRTUAL REAL FUNCTION IMP SIZE,
                      IMP DURATION;
PROCEDURE EXP IMPULSE:
   BEGIN
      ENFORCE ELEC. ELEC IMPULSE
              PUT (*SIZE:=IMP SIZE;
                    DURATION:=IMP DURATION*)
              UPON ELEC;
   END
```

It should be observed that the procedures EXP IMPULSE and ELEC IMPULSE are redundant in the system description, in the sense that their effects could be obtained by use of the procedures POLARIZE. and ELEC STRENGTH CHANGE. In the physical experiment, however the experimenter has the possibility of pressing two different buttons: one for switching on and off a continous injection of a current (the sign and strength of which is determined by the experimenter), an-

other for injection of a current of very short duration (the sign, strength and duration determined by the experimenter). Those two buttons correspond to the procedures POLARIZE and EXP IMPULSE respectively.

2. Complete System Description

In the preceding chapters we have described the single components of the system and their interaction (figure 2. 1):

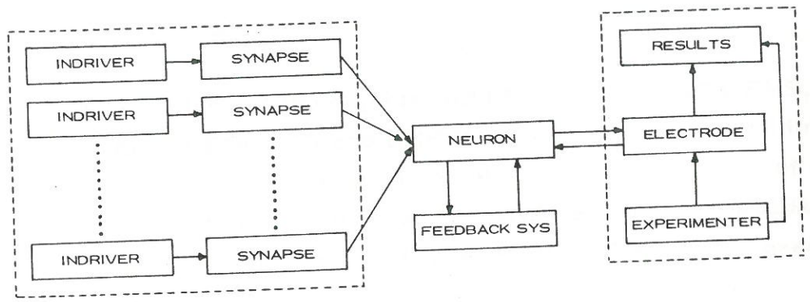


Figure 2. 1
The components of the total system

(Again directed lines represent reference variables and not necessarily information flow.)

In the following the complete system is described by the system object.

The system object contains

- objects describing the single components of the system and
- the actions required to establish the initial state of the experiment; these actions will appear as initializations in the system object.

```
SYSTEM BEGIN
   CLASS INDRIVER:
   .
   .
   .
   EXPERIMENTER:
   .
   .
   .
   TASK BEGIN
      initializations;
      WHILE experiment going on IMPOSE {REST};
   END TASK;
END SYSTEM
```

By the declarations of the objects NEURON, FEEDBACK SYS, ELECTRODE, RESULTS and EXPERIMENTER, these are introduced in the system. The initializations establish the environment of the NEURON by introducing a number of SYNAPSE and INDRIVER objects. When the initializations are made the system object begins executing the "rest action":

```
WHILE experiment going on IMPOSE {REST}.
```

```
SYSTEM BEGIN
  CLASS INDRIVER:
    OBJECT BEGIN
      COMMENT An indriver object feeds the SYNAPSE
              object SYN to which it is connected with
              a stream of impulses simulating part of the
              environment of the single neuron system;
      REF (SYNAPSE) SYN;
      REAL T NEXT;
      VIRTUAL REAL FUNCTION INTERVAL;
      COMMENT The function INTERVAL determines
              the intervals between the sending of im-
              pulses to the SYNAPSE object denoted
              SYN;
      TASK BEGIN
        REPEAT
          (*ENFORCE SYN. RECEIVE IMPULSE
              UPON SYN;
            T NEXT:=TIME + INTERVAL;
            WHILE TIME < T NEXT IMPOSE {REST}      *);
      END TASK;
    END OBJECT INDRIVER;
```

```
CLASS SYNAPSE:
  OBJECT BEGIN
    COMMENT A SYNAPSE object rests until it receives
            an impulse from its INDRIVER. It then
            transmits to the NEURON by means of the
            procedure IMPULSE and resumes resting;
    REF (NEURON) NEU;
    REAL STRENGTH;
    VIRTUAL TASK (NEURON) PROCEDURE IMPULSE;
    COMMENT The task procedure IMPULSE changes the
            NEURON's ability to fire. The size of the
            change depends upon the value of the
            STRENGTH of THIS SYNAPSE and the
            internal state of the NEURON;
    TASK PROCEDURE RECEIVE IMPULSE:
      TASK BEGIN
        ENFORCE IMPULSE
          PUT IMPACT:=STRENGTH
          UPON NEU;
      END TASK;
    COMMENT The procedure RECEIVE IMPULSE is
            executed when receiving an impulse from
            the INDRIVER object;
      TASK BEGIN
        CONTINUE {REST};
      END TASK;
  END OBJECT SYNAPSE;
```

```
CLASS EXC SYNAPSE:
   SYNAPSE OBJECT BEGIN
            COMMENT In this subclass of the class
                    SYNAPSE we define the procedure
                    IMPULSE to have an excitatory
                    effect on the NEURON. The size
                    of the effect is determined by the
                    REAL FUNCTION EXC. The function
                    is declared VIRTUAL and not speci-
                    fied further;
          VIRTUAL REAL FUNCTION EXC;
          TASK (NEURON) PROCEDURE IMPULSE:
             TASK BEGIN
                REAL IMPACT;
                NEU. EPSP LAST:=NEU. EPSP DECAY +
                                 EXC(IMPACT, NEU. POT);
                NEU. T LAST EXC:=TIME;
             END TASK;
       END OBJECT EXC SYNAPSE;
```

```
CLASS INH SYNAPSE:
   SYNAPSE OBJECT BEGIN
                COMMENT In this subclass of the class
                        SYNAPSE the procedure IMPULSE
                        is defined to have an inhibitory
                        effect on the NEURON. The size
                        of the effect is described by the
                        VIRTUAL REAL FUNCTION INH
                        and not specified further;
      VIRTUAL REAL FUNCTION INH;
      TASK (NEURON) PROCEDURE IMPULSE:
         TASK BEGIN
            REAL IMPACT;
            NEU.IPSP LAST:=NEU.IPSP DECAY +
                                 INH(IMPACT,NEU.POT);

            NEU. T LAST INH:=TIME;
         END TASK;
   END OBJECT INH SYNAPSE;
```

```
NEURON:
  OBJECT BEGIN
    COMMENT The NEURON object describes a single
            neuron. It receives impulses from
            INDRIVERs (through SYNAPSEs), its
            FEEDBACK SYS and the ELECTRODE.
            When firing it emits impulses to the
            FEEDBACK SYS object and the
            ELECTRODE;
    REAL POT, PSP, T LAST, T LAST EXC, T LAST INH,
         EPSP LAST, IPSP LAST;
    REF (FEEDBACK SYS) FB SYS;
    REF (ELECTRODE) ELEC;
    VIRTUAL REAL FUNCTION FT DECAY, BP BUILDUP,
                          EPSP DECAY, IPSP DECAY,
                          ELEC CONTRIBUTION;

    PROCEDURE FIRE:
      BEGIN
        ENFORCE FB SYS. RECEIVE IMPULSE
          UPON FB SYS;
        ENFORCE ELEC. RECEIVE FIRING
          UPON ELEC POWER HIGH;
        T LAST:=TIME;
      END;
    COMMENT Below follows a description of the NEURON's
            ongoing internal processes by means of an
            object descriptor. Throughout the existence
            of the NEURON the relations described by
            the proberty descriptor of INT PROCESS
            are imposed upon the NEURON;
```

```
INT PROCESS:
  OBJECT BEGIN
    TASK BEGIN
      CONTINUE
        {PSP = EPSP DECAY
                - IPSP DECAY,
         POT = PSP + BP BUILDUP
                + ELEC CONTRIBUTION};
    END TASK;
  END OBJECT INT PROCESS;
COMMENT The total activity of the neuron thus consists
        of the ongoing internal processes modulated
        by the additional actions of the working cycle
        and the actions enforced by the interrupts
        stemming from the reception of impulses;
TASK BEGIN
  REPEAT
    (*WHILE POT < FT DECAY IMPOSE {REST};
      FIRE;
      WHILE absolute refraction obtains
      IMPOSE {restoration of spike initiation mechanism} *);
  END TASK;
END OBJECT NEURON;
```

```
FEEDBACK SYS:
  OBJECT BEGIN
    COMMENT The feedback system consists of those mechanisms
            which make the output of the neuron dependent upon
            previous firings. Feedback memory is a data
            structure capable of holding a limited set of real
            values representing times of arrival of impulses
            from the NEURON;
    REF (NEURON) NEU;
    REAL T NEXT FB;
    TASK PROCEDURE RECEIVE IMPULSE:
      TASK BEGIN
        adjust feedback memory and T NEXT FB;
      END TASK;
    COMMENT The procedure RECEIVE IMPULSE is executed
            when receiving an impulse from the NEURON;
    VIRTUAL TASK (NEURON) PROCEDURE FEEDBACK;
    COMMENT The task procedure FEEDBACK represents
            the effects of FEEDBACK SYS on the NEURON.
            It is not described further. In the working cycle
            below, the FEEDBACK SYS object rests until
            the next time at which to affect the NEURON
            (T NEXT FB), then it enforces the task procedure
            FEEDBACK upon the NEURON;
    TASK BEGIN
      REPEAT
        (*WHILE TIME < T NEXT FB IMPOSE {REST};
          ENFORCE FEEDBACK UPON NEU;
          adjust T NEXT FB   *);
    END TASK;
  END OBJECT FEEDBACK SYS;
```

```
RESULTS:
  OBJECT BEGIN
    COMMENT The RESULTS object represents the result
            collecting equipment connected to the
            ELECTRODE during the experiment. It rests
            until the ELECTRODE object transmits a
            firing of the NEURON to it;
    TASK PROCEDURE RECEIVE FIRING:
      TASK BEGIN
        update statistics;
      END TASK;
    TASK BEGIN
      CONTINUE {REST};
    END TASK;
  END OBJECT RESULTS;
```

```
ELECTRODE:
  OBJECT BEGIN
    COMMENT The ELECTRODE object represents the electrode
            impaled into the neuron during the experiment. It
            rests until it is interrupted either by a firing of
            the NEURON or by the EXPERIMENTER;
    PRIORITY HIGH RANKED;
    REF (RESULTS) RES;
    REF (NEURON) NEU;
    REAL ELEC STRENGTH;
    TASK PROCEDURE RECEIVE FIRING:
      TASK BEGIN
        ENFORCE RES. RECEIVE FIRING UPON RES;
      END TASK;
    COMMENT The task procedure RECEIVE FIRING is enforced
            upon the ELECTRODE by the NEURON when fi-
            ring. When executed by the ELECTRODE it trans-
            mits the firing of the NEURON to the RESULTS
            object;
  TASK PROCEDURE ELEC STRENGTH CHANGE:
    TASK BEGIN
      REAL SIZE;
      NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
      NEU. T LAST CHANGE:=TIME;
      ELEC STRENGTH:=SIZE;
    END TASK;
  COMMENT The task procedure ELEC STRENGTH CHANGE
          is enforced upon the ELECTRODE by the
          EXPERIMENTER to change the effect of the
          ELECTRODE on the potential of the NEURON;
```

```
TASK PROCEDURE ELEC IMPULSE:
  TASK BEGIN
    REAL SIZE, T IMP, DURATION;
    NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
    NEU. T LAST CHANGE:=TIME;
    T IMP:=TIME;
    ELEC STRENGTH:=ELEC STRENGTH + SIZE;
    WHILE TIME < T IMP + DURATION IMPOSE {REST}
                          RESISTANCE HIGH;
    NEU. ELEC CON LAST:=NEU. ELEC CONTRIBUTION;
    NEU. T LAST CHANGE:=TIME;
    ELEC STRENGTH:=ELEC STRENGTH - SIZE;
  END TASK;
COMMENT The procedure ELEC IMPULSE is executed when
        the ELECTRODE by the EXPERIMENTER is made
        to send an impulse to the NEURON similar in effect
        to a synaptic impulse;
TASK BEGIN
  CONTINUE {REST};
END TASK;
END OBJECT ELECTRODE;
```

```
EXPERIMENTER:
  OBJECT BEGIN
    REF (ELECTRODE) ELEC;
    REF (RESULTS) RES;
    VIRTUAL REAL FUNCTION POLAR STRENGTH;
    PROCEDURE POLARIZE:
      BEGIN
        ENFORCE ELEC. ELEC STRENGTH CHANGE
                PUT SIZE:=POLAR STRENGTH
                UPON ELEC;

      END;
    COMMENT POLARIZE (and POLAR STRENGTH) is used
            to polarize the NEURON.
            EXP IMPULSE (and IMP SIZE, IMP DURATION)
            is used to send short impulses to the NEURON
            similar in effect to synaptic impulses;
    VIRTUAL REAL FUNCTION IMP SIZE, IMP DURATION;
    PROCEDURE EXP IMPULSE:
      BEGIN
        ENFORCE ELEC. ELEC IMPULSE
                PUT (*SIZE:=IMP SIZE;
                      DURATION:=IMP DURATION  *)
                UPON ELEC;

      END;
    TASK BEGIN
      impale the ELECTRODE into the NEURON;
      WHILE experioment going on DO
        (*interact with the NEURON through the ELECTRODE
          by using the procedures POLARIZE and
          EXP IMPULSE and observe the effects on the
          RESULTS object  *);

    END TASK;
END OBJECT EXPERIMENTER;
```

```
COMMENT Below follows the inherent task of the system
        object consisting of initializations and a
        "rest action";
TASK BEGIN
   Set up the environment of the NEURON by introducing
   a number of INDRIVER and SYNAPSE objects;
   WHILE experiment going on IMPOSE {REST};
END TASK;
END SYSTEM
```

Appendix: Basic Concepts of the DELTA Language

A description in the DELTA language consists of formal and informal elements. Formal elements are written in upper case (capital) letters while informal elements are written in lower case letters.

A set of reserved words having predefined meanings, the keywords, are written in underlined upper case letters. The keyword VIRTUAL used in the specification of a DELTA entity implies that the final definition of the entity belongs to a deeper level of description containing more details about the entity under consideration.

Objects

Systems are described in terms of components named objects interacting within the enclosing system object. The system object represents the system as a whole.

Objects are described by

- their data characteristics, in terms of variables, described by variable declarations,
- quantities being functions of other quantities, described by function declarations,
- their action repertoire, in terms of action rules, described by procedure declarations,
- their structure of internal objects, described by object descriptors,
- entities whose final definitions are not given, virtuals
- their inherent actions, described by a sequence of either instantaneous changes of state (e.g. of the values of some of the variables) or time consuming actions, representing continuous changes of state during time intervals. The inherent actions, regarded as a whole, are referred to as the inherent task of the object.

The underlined words above are described in the following.

The structural properties of an object are discribed by an object block enclosed by the brackets OBJECT BEGIN and END OBJECT:

```
OBJECT BEGIN
      variables;
      functions;
      procedures;
      object descriptors;
      virtuals;
      TASK BEGIN
            inherent task;
      END TASK;
END OBJECT
```

The various language elements appearing in the description (e.g. declarations, actions) are separated by semicolons ";".

A singular object is described by an object label, giving the title of the object, followed by an object block, e.g.

A : OBJECT BEGIN ..... END OBJECT

describes an object with the title A.

Since a number of objects may be of the same kind, i.e. having the same structural properties, descriptions of objects may be given as class declarations:

CLASS MAN : OBJECT BEGIN ..... END OBJECT

By this class declaration a class of objects is introduced, each having the title MAN.

Objects belonging to the same class (described by a common class declaration) share structural properties,

- number, types and names of variables
- virtuals
- functions
- action repertoire, described by procedure declarations
- inherent task

but differ by

each being an individual entity

and

at a certain moment usually

- having different values assigned

  to their variables (i.e. being in different states)

- being in different stages of execution of their inherent task

If two or more classes have common features, these features may be described in a separate class and then used as prefix upon declaration of the classes. Classes declared with a prefix are called subclasses of the prefix class.

Suppose that we want to declare the classes TRUCK and BUS. A common feature of trucks and busses is that both are provided with a license number. This is due to the fact that they are vehicles. But in addition a truck is characterized by its load and a bus by its seatings. This may be described by declaring the classes TRUCK and BUS as subclasses of the class VEHICLE:

```
CLASS VEHICLE :        OBJECT BEGIN
                           INTEGER LICENCE NO.;
                           :
                       END OBJECT;

CLASS TRUCK : VEHICLE OBJECT BEGIN
                           REAL LOAD ;
                           :
                       END OBJECT ;

CLASS BUS : VEHICLE OBJECT BEGIN
                           INTEGER SEATINGS ;
                           :
                       END OBJECT
```

The structural properties of the objects of a subclass are those of the objects of the prefix class extended with those of the object block. That is an object of the class TRUCK contains the variables LICENCE NO and LOAD, an object of the class BUS has the variables LICENCE NO and SEATINGS (fig. A.1.).

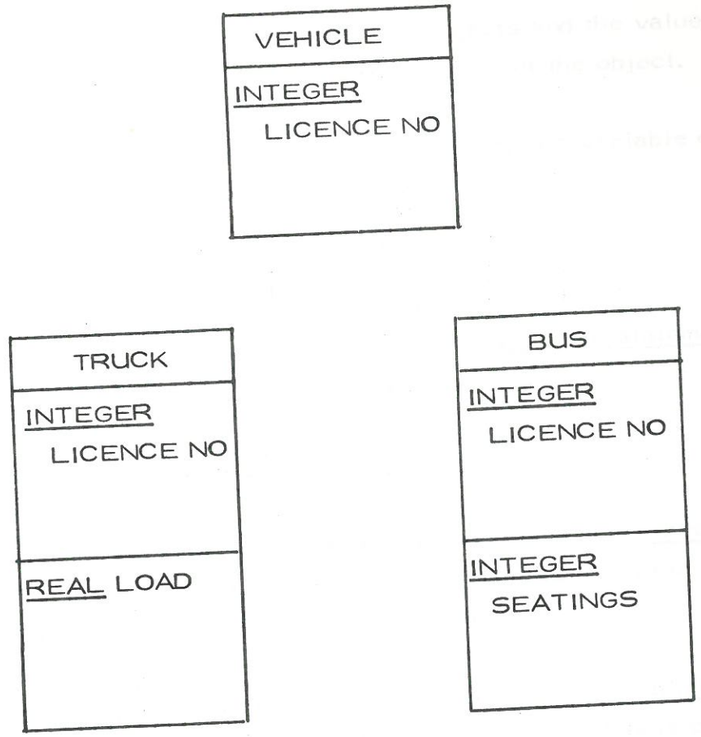


Figure A.1

Three objects, belonging to a class and two subclasses

A quantity introduced as VIRTUAL in the class VEHICLE may in TRUCK get its final definition. The same quantity may in BUS get another definition, e.g. a quantity describing the task of emptying a vehicle.

In the following each of the components of an object block is described in detail (except virtuals and object descriptors).

Variables

A variable is a quantity which consists of

- a name
- a value, e.g. 5.2 or -7
- a type, which defines the set of possible values of the variable, e.g. integer or real.

Variables are used to characterize objects and the values of the variables of an object describes the state of the object.

The type and name of a variable is fixed in the variable declaration, e.g.

```
REAL A, B, C
```

whereas the value may be changed by assignment statements. The effect of an execution of the assignment statement

```
C := A + B
```

is that the value of C is changed to the sum of the current values (the values at the time of execution of the statement) of A and B. The language element ":=" reads "becomes".

An important type of variables are the reference variables. Reference variables may have objects as values and thus denote objects. The set of possible values of a reference variable is restricted to a given class of objects or to a singular object. The class or the singular object is specified by the qualification part of the declaration of the variable. By

```
REF (MAN) ADAM
```

a reference variable ADAM is declared, qualified by the class MAN. ADAM may have as value objects belonging to the class MAN. The object which is the value of ADAM at a given moment is said to be referred to by ADAM. Phrased in another way: ADAM denotes the object. In fig. A.2 the reference is contained in an object of class A:

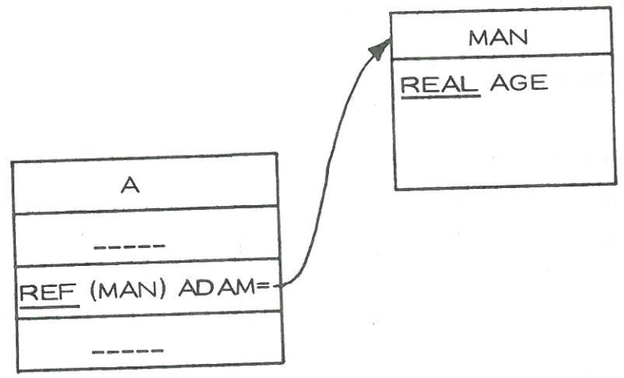


Figure A.2

The use of a directed line to

indicate the value of a reference variable

Access to ADAM's AGE is within the A object obtained by "ADAM.AGE" (the genitive notation "'s" being substituted by a dot, " . "). An object may reference itself by use of the keyword THIS followed by its title (usually its class name). In the object which ADAM refers to, THIS MAN denotes the same object as ADAM.

In the DELTA language it is possible to reference a singular object simply by means of its title. When this report was written the use of singular objects was not quite settled and therefore we reference singular objects only by the use of reference variables.

Functions

A function is a quantity whose value depends upon the values of other quantities. The running speed of a man may depend upon several quantities, e.g. his condition, and this may be described by the function

REAL FUNCTION RUNNING SPEED : BEGIN .... END

By this declaration a function of type REAL with the title RUNNING SPEED is described. The function block enclosed by BEGIN and END gives the definition of the function, that is gives a rule for determining the value of RUNNING SPEED from the values of other quantities (e.g.

CONDITION). If a function is declared as VIRTUAL the function block is omitted and the actual dependence on other quantities not specified:

```
VIRTUAL REAL FUNCTION RUNNING SPEED
```

By both of the above declarations the function is specified to be of type REAL. This means that RUNNING SPEED, when applied, yields a real number as result. A function may be applied whenever a variable of the same type could have been used, e.g.:

```
DISTANCE COVERED := RUNNING SPEED x (TIME-START TIME)
```

The quantities upon which the value of RUNNING SPEED depends may be given as parameters, e.g.

```
DISTANCE COVERED :=
    RUNNING SPEED (CONDITION, LENGTH OF LEGS,
        REASON FOR RUNNING) x (TIME - START TIME)
```

Procedures

A procedure declaration defines an action rule (pattern) which can be used within the system. The action rule may be executed by the object in which it is declared or by other objects according to rules specified below.

Procedures are of two kinds, instance procedures and task procedures.

An instance procedure declaration has the format:

```
PROCEDURE P :
    BEGIN
        Declaration of local quantities;
        Instantaneous actions;
    END
```

The procedure block defining the action rule is enclosed by the key-words BEGIN and END, and P is the title of the procedure. The procedure block contains declarations of quantities to be used in the procedure block and the sequence of instantaneous actions defining the action rule.

An object performs the actions of an instance procedure by a "call" of the procedure:

- in the object in which it is declared by

  ....; P ;....

- in an object referring to that object (by a reference variable or the title of the object if it is a singular object) by e.g.

  ....; AB.P ;....

  where AB is a reference variable denoting the object in which P is declared (Fig. A.3).

The procedure may be executed by no other objects.

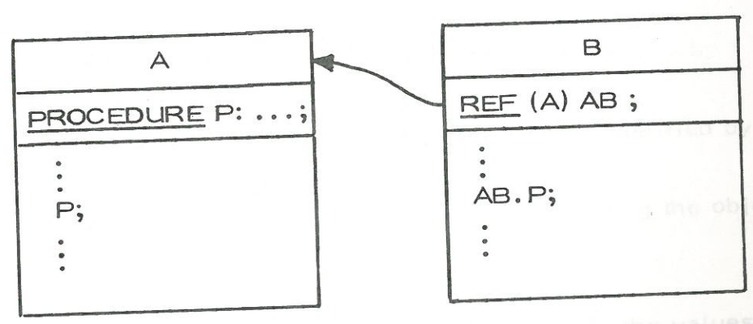


Figure A. 3
A "local call" (left) and a
"remote call" (right) of a procedure

All action rules containing time consuming actions are declared as task procedures. The set of objects which may perform the action rule specified by a task procedure is either

- one class of objects,
- one singular object

or

- the object in which it is declared.

In the first two cases the declaration of the task procedure contains the title of the class or the title of the singular object; the task procedure is said to be qualified by the class or the singular object. In the last of the three cases the qualification is omitted from the declaration.

The action rule is described by a task block which is enclosed by TASK BEGIN and END TASK, e.g.

```
TASK (MAN) PROCEDURE P :
    TASK BEGIN
        Declarations of local quantities;
        Actions;
    END TASK
```

This declaration describes an action rule which may be performed only by objects of the class MAN.
An object performs the actions of a task procedure by a "call" of the procedure:

- in the object in which it is declared by

  ....; ENFORCE P ;....

- in objects by which the task procedure is qualified by e.g.

  ....; ENFORCE ADAM.P ;....

  where ADAM is a reference variable denoting the object in which P is declared.

In the call of a procedure it is possible to specify the values of one or more of the local variables. If the task procedure P has a local variable declared by REAL A, we may write

....; ENFORCE P PUT A := 1.5;....

This implies that when the execution of the action rule begins the value of A will be 1.5.

The Inherent Task

The action pattern of the object is described by a task block. The task block is enclosed by TASK BEGIN and END TASK. The task block describes the actions of the object in terms of a sequence of statements.

When the actions are exhausted the object will remain inactive. It is then said to be terminated.

Statements

Instantaneous actions may be described by algorithmic statements, e.g. A := A + 1, the effect of an execution of this statement is that the value of A is increased by one. Time consuming actions are described by means of property descriptors. A property descriptor is a list of relations between quantities of the system, separated by commas (","), and inclosed by braces ({...}).

Time consuming actions are described by statements of the form

WHILE logical expression IMPOSE property descriptor

where the logical expression states the duration of the action. The statement

WHILE TIME < NEXT TIME IMPOSE {A = B + TIME, B = C}

has the effect that the fulfilment of the relations A = B + TIME and B = C is imposed upon the system as long as the value of TIME is less than the value of NEXT TIME. That is, as long as the logical expression is "true" the relations of the property descriptor are imposed upon the system. TIME is a variable available allthrough the system, at any moment giving the current time. TIME is continuously increasing and is the basis for description of dynamics in the system.

An object upon which a property descriptor is to be imposed as long as it exists, except for the periods when it executes interrupting tasks enforced upon it by other objects, is described by the inherent task

CONTINUE property descriptor

The statement is equivalent to the statement

WHILE TRUE IMPOSE property descriptor

When {REST} is used as a property descriptor this implies that the object remains passive as long as the property descriptor is imposed upon the object. An object which is to be passive as long as it exists, except when it executes interrupting tasks, is thus described by the inherent task

CONTINUE {REST}

Statements may be grouped together to compound statements by use of the statement parentheses (* ... *), and a statement may be executed repeatedly by writing

WHILE logical expression DO statement

The statement is executed repeatedly as long as the logical expression has the value "true".

The following example illustrates the use of statement parentheses and "WHILE ... DO ...":

An inherent task describing the performance of a symphony in five movements could be written

```
ENFORCE PLAY MOVEMENT PUT MOVEMENT NO:= 1;
WHILE pause IMPOSE {REST};
ENFORCE PLAY MOVEMENT PUT MOVEMENT NO:= 2;
    :
WHILE pause IMPOSE {REST};
ENFORCE PLAY MOVEMENT PUT MOVEMENT NO:= 5;
WHILE pause IMPOSE {REST};
WHILE audience not too tired IMPOSE {applause}
```

where PLAY MOVEMENT is a task procedure (time consuming) describing the performance of movement number MOVEMENT NO. ("pause", "audience not too tired" and "applause" are used as informal languages elements, and are consequently written in lower case letters.)

The task could also be written

```
NO:= 1;
WHILE NO ≤ 5 DO
   (*ENFORCE PLAY MOVEMENT PUT MOVEMENT NO:= NO;
     WHILE pause IMPOSE {REST}
     NO:= NO + 1   *);
WHILE audience not too tired IMPOSE {applause}
```

The description could be made more general by using a variable, NUMBER OF MOVEMENTS instead of the constant 5, that is

```
NO:= 1;
WHILE NO ≤ NUMBER OF MOVEMENTS DO
     (* ..... *);
WHILE audience not too tired IMPOSE {applause}
```

An object performing the same action repeatedly is described by the inherent task

REPEAT statement

This statement is equivalent to

WHILE TRUE DO statement

Interrupts

An object may enforce the execution of an action rule upon another object, i.e. interrupt the other object. An action rule (instantaneous or time consuming) enforced upon another object is declared as a task procedure, and the qualification of the task procedure (see section on procedures) defines the set of objects upon which the action rule may be enforced. An interrupt is generated by an interrupt statement, e.g.

ENFORCE P UPON A

where A is a reference variable denoting the object to be interrupted and P is the name of a task procedure.

Only time consuming actions may be interrupted. When an object tries to interrupt another object then

- either the interrupt penetrates immediately, that is the time consuming action is discontinued, and the object begins the execution of the task enforced by the interrupt. When the task is concluded the object resumes the interrupted time consuming action.
- or the time consuming action resists the interrupt and continues. The interrupt is placed in an interrupt queue. It may then later possibly penetrate and be executed (i.e. when the object embarks upon another time consuming action).

Whether an interrupt penetrates or not is decided upon by comparing the importance of the ongoing time consuming action with the importance of the interrupt. The comparison is made on the basis of priorities associated with the ongoing action and the interrupt. The priority value of the ongoing action is called its resistance (priority value). The priority value of the interrupt is called its power (priority value).

Time consuming actions and interrupts which are not explictly given priority values are regarded as having the priority value NO. Other priority values (than NO) may be introduced by a priority declaration, e.g.

```
PRIORITY P1, P2
```

which introduces the priority values P1 and P2.

An interrupt with power P1 or P2 penetrates a time consuming action with resistance NO, while a time consuming action with resistance P1 or P2 resists an interrupt with power NO. When declared as above

```
P1 resists P1 and P2
and P2 resists P1 and P2.
```

Other relations between priorities than the above may be stated explicitly in the declaration. This may be done in two ways:

- by use of the keyword PRECEDENCE followed by a list of one or more precedence relation pairs
- by use of the keyword RANKED.

In the declaration

PRIORITY P1 PRECEDENCE P1 < P1

the precedence relation pair P1 < P1 states that a time consuming action with resistance P1 will be interrupted by an interrupt with power P1.

The use of the keyword RANKED, e.g. in

PRIORITY P1, P2 RANKED

is equivalent to stating the precedence relation pairs

P1 < P1, P1 < P2, P2 < P2

The priority value of a statement is specified by the use of the keywords

RESISTANCE for time consuming actions, e.g.

CONTINUE {REST} RESISTANCE P1

and POWER for interrupt calls, e.g.

ENFORCE P UPON A POWER P2.

REFERENCES

[1] Nygaard, K. (1973):
"On the Use of an Extended SIMULA in System Description".
NCC-publication S-59.

[2] Fjellheim, Håndlykken, Nygaard (1974):
"Report from a Seminar on System Description at 'Skogen', Røros", NCC-publication S-65.

[3] Wallöe, Jansen, Nygaard (1969):
"A Computer Simulated Model of a Second Order Sensory Neurone", Kybernetik 6, 130-140 (1969).