SEKI-PROJEKT

SEKI MEMO

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

Programming in a Distributed Environment : a Collection of CSSA Examples

Hans Voss

Memo SEKI-82-01

Programming in a Distributed Environment :

A Collection of CSSA Examples

Hans Voss

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
West Germany

Abstract :

Several CSSA scripts as solutions of mostly well known problems in the area of parallel activities are documented. All but two examples are compared with "equivalent" programs which are based on the concept of monitors. One example gives a distributed solution of the N-queens problem. Besides the purpose of demonstrating the use of CSSA concepts, the last example deserves particular attention. Here, CSSA is used for simulating and testing the logic of the "window mechanism" which is a representative for the class of communication protocols in computer networks.

Contents :

0. Introduction

This paper is a documentation of my first programming experiences with CSSA. The study of the examples offers an opportunity to deepen the understanding of CSSA concepts and the corresponding programming methodology. This is even more true, because a comprehensive reference manual and application guide is currently being prepared.

Perhaps with the exception of the alarm clock solution in Section 5, the decision for the ordering of the examples was based on increasing complexity. Therefore, a CSSA greenhorn is advised to profit from the sequence given. For the advanced reader a deviating reading order or picking up one specific example should cause no problems.

In addition to short comments on the well known synchronization problems discussed in Sections 2 to 5, "equivalent" solutions for the same problems using monitors are presented. For sake of coherent and unmisleading reference all these monitor solutions are programmed using syntax from the programming language CLU [Lis, Sny, Atk, Sch 77] and are taken from [Bloom 79]. Without intending a rigorous comparison between the monitor approach and the CSSA approach, some hints at specific differences are given.

Due to personal time constraints I did not succeed in making this paper self contained. I rather decided to make the following assumptions:

1. The reader should have some basic knowledge about the CSSA concepts and programming constructs.
   If this is not the case, he should first read [FRV 81] or the more concise and complete publication [FRV 82].

2. A full understanding of Sections 2.3, 3.3, 4.1.3 and 5.3 is only possible, if the reader is familiar with the monitor concept. No further comments at the syntax and semantics of the monitor programs are included in this paper.

All the CSSA examples have been compiled and executed by the BMS-CSSA-Simulation System, which was designed and implemented by the students C. Beilken, F. Mattern and M. Spenke (BMS). Without their great efforts and personal engagement programming in CSSA would have remained a paper and pencil exercise for an indefinite amount of time.
In the simulation system, translation from a CSSA source script into executable code is a two step process. First, the CSSA source is compiled into a SIMULA program, which in the second step is embedded in a SIMULA multiprocessor simulation system.
All but one example have been executed with the default processor configuration depicted at the beginning of the first session protocol (fig. 2). For the window mechanism example described in Section 7, a problem oriented configuration was designed by simply changing the default parameters of the simulation system.

In all comments , lines of the source programs are referred to by

enclosing their numbers in parantheses.

1. One Slot Buffer

1.1. Problem Specification

A one slot buffer is an object which may contain exactly one information element. A new information can be put into the buffer only via a **write operation** and later be inspected by a **read operation.**
The synchronization problem is specified by the following restrictions:

1. write and read operations must take place in alternating order.

2. The first operation to be executed must be a write operation.

1.2 CSSA Solution

A CSSA solution for the one slot buffer problem is straightforward (fig.1). The buffer is represented by a string variable BUFFER.
In two parallel facets READ and WRITE the only operations RD and WR are defined respectively. Each execution of an RD operation causes a change to facet WRITE, such that a WR operation must be executed next. The same is true vice versa. The initial statement (25) guarantees the first operation to be a WR operation.

1.3 General Remarks

The history information of alternating reads and writes can be very well expressed by use of the facetting mechanism. The CSSA solution appears to be simpler than any monitor solution, since there at least one extra (boolean) variable storing this history information is needed. However, monitors do not have facets. But in my opinion, besides their practical importance the facet structure contributes much to a clear documentation of the CSSA solution.

2. Bounded Buffer

2.1 Problem Specification

A bounded buffer can be regarded as a generalization of the one slot buffer. In contrast to the one slot buffer it can store more than one element up to a fixed maximum number.
Users of the buffer can store and inspect information via two operations called **insert** and **remove**.
Like a one slot buffer agent, a bounded buffer agent has to solve the following synchronization problems:

1. New information may only be inserted into a buffer element iff

   1.1 either this element never has been used for insertion before, or
   1.2 the last information having been inserted into this element has already been removed.

2. A buffer element may only be removed iff information has been written into this element before, and after this insertion no other remove operation for this element has occured.

So, each buffer element can be interpreted as a one slot buffer. A bounded buffer with maximum element number 1 should exhibit the same behaviour as a one slot buffer.

2.2 CSSA Solution

In each creation message for a bounded buffer agent the buffer size MAX (line 1) must be included (cf. fig. 3). According to the problem specification, at each time the buffer agent is in one of three possible states:

1. In facet BUFFER_EMPTY, there is no buffer element which can be removed (operation REM).
   This is the case in the initial state (40) or later, when all inserted elements have been removed and after these removals no new elements have been inserted. Therefore, the only messages which can be processed in this facet are insert messages (operation INS).

2. When exactly MAX insertion messages have been processed without an intervening remove operation, the buffer agent's current facet becomes BUFFER_FULL.
   In this state the enabling of an INS message would destroy the information in a buffer element, which has not yet been inspected by a REM operation. Since this would violate restriction 1.2 of the problem specification, only REM

operations are allowed in this state (18).

3. In facet NO_CONSTRAINT there exists both

   3.1 a buffer element into which information has been inserted but not removed, and
   3.2 a buffer element which either never has been used for insertion or which already has been inspected and after inspection no further insertion has taken place.

   Because of 3.1 at least one REM message can be enabled, whereas 3.2 allows the processing of at least one INS message.

In order to decide which of these facets is to be chosen next, two integer variables INS_COUNT and REM_COUNT are incremented modulo MAX in each INS and REM operation respectively (27,34).
At each time, INS_COUNT is the index of the next buffer element to be used for insertion, and REM_COUNT yields the index for the next REM operation. If REM_COUNT becomes equal to INS_COUNT, the next current facet must either be BUFFER_FULL or BUFFER_EMPTY, depending on what operation was executed last (28,35).

A general comment on the program structure will close this section. Besides the natural disjunction of facets the bounded buffer solution demonstrates how the programmer profits from the possibility to define operations at the top level of a script (3,25-37). The advantage is that he has to write down the operations only once, including them in a facet where needed (14,18,22). In this specific example, the programmer has to put up with a slight disadvantage: as he can't decide in which facet the operations will be executed at any time, each enabled message causes a new facet instantiation to be set up. I.e., a new instantiation of facet NO_CONSTRAINT will be set up, although possibly the agent could have stayed in this very facet. Certainly, this little inefficiency could have been avoided by introducing a further variable remembering the current facet. For facets BUFFER_EMPTY and BUFFER_FULL, there is no such problem, because each operation execution in one of these facets must cause a change to another facet.

2.3 Comparison with the Monitor Solution

In script BOUNDED_BUFFER no event conditions (buffer.nonfull, buffer.nonempty, c.f. fig. 5) are needed. Instead of starting the execution of an arbitrary operation, and then testing if the started operation may continue, always only allowed operations are selected and executed without interruption.

This strategy rules out a more general difference. As a monitor

programmer has no influence on the sequence of started operations, often one of the first actions of a monitor procedure is to check if its execution may continue. In contrast, CSSA operations are indivisible. Once started, they must be executed without interruption. Hence, a CSSA programmer is forced to extract the synchronization code from the "operations" and put it into the facet structure and possibly into assertions governing the selection of messages in the mailbox.

3. Protected Buffer

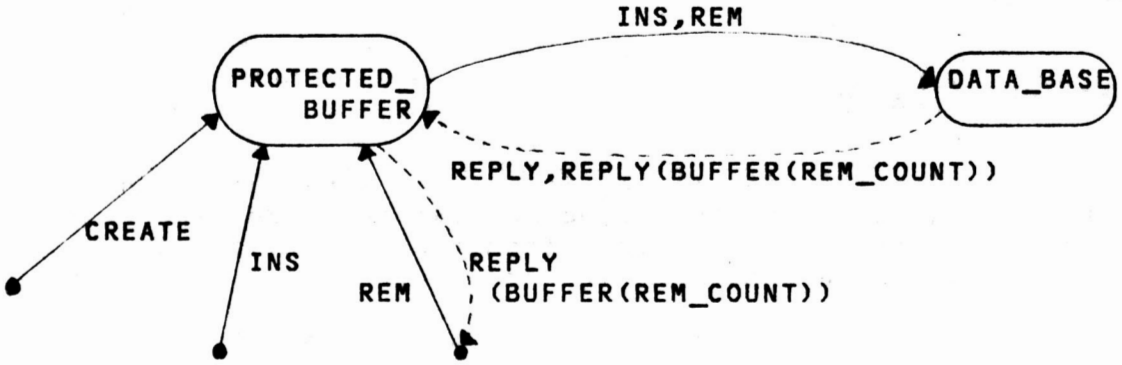
3.1 Problem Specification

In the bounded buffer solution of section 2. the same agent implemented both the **buffer** and the **synchronization code** for the buffer. In other words, the synchronization code visualized in the special facet structure and pure buffer access functions have been mixed up.
As is discussed in more detail in [Bloom 79], modifiability, modularity and understandibility will be enhanced when the buffer resource object itself is no longer part of the module which implements its synchronization scheme. Defined in such separated modules, the implementation of the buffer may be changed without modifying its synchronization code. Conversely, the synchronization scheme for the buffer can be altered without changing the buffer implementation.

3.2 CSSA Solution

In fig. 6, the synchronization code is defined by script PROTECTED_BUFFER, whereas the definition of the buffer with the buffer access functions is given in script DATA_BASE (6-35).
An agent of type DATA_BASE exactly executes the operations he is ordered to by its protector agent (an agent of type PROTECTED_BUFFER). Due to the inherent assumption in the CSSA model of computation that messages may pass each other, the protector-resource relationship must be "strongly coupled", guaranteeing that protector orders are followed in the sequence given. This can be achieved by having each execution of an INS or REM operation by the protector agent only finished after receiving a reply (78,91) indicating that the corresponding order to the resource agent (76,89) has been followed.

Script DATA_BASE does not seem to fit very well to the requirements R_COUNT are never needed in this environment. In spite of this inconvenience, the database is included in its present structure, because the same script can be used for the readers_writers solutions in the following section.



Users of the buffer communicate only with an agent of type PROTECTED_BUFFER who delegates requests to an agent of type DATA_BASE. It's impossible for a user to gain direct access to the real resource agent.

3.3 Comparison with the Monitor Solution

The two solutions differ with respect to the interpretation of where the borderline between the synchronization code and the buffer code should be defined. In the CSSA solution, a buffer is simlpy a data structure which can store and retrieve information provided an unique key is presented. In the monitor solution (fig. 8), a buffer still has much similarity with a bounded buffer in that it provides two functions **full** and **empty**. As a consequence, the protected buffer monitor no more has to take care of the counters REM_COUNT and INS_COUNT. In my opinion, these counters are essential elements of the synchronization code and not of the buffer definition.
Not willling to go into the particulars of these arguments, I only state that with the monitor concept and the CSSA concept corresponding solutions can be given for both interpretations. Certainly, no lack of expressiveness in either of the two concepts could be derived from such a discussion.

4. Readers Writers Problems

The readers_writers problem is particularly interesting because with only slight modifications of the general problem (cf. section 4.1.1) various versions with different synchronization aspects can be formulated. In section 4.1 the readers_writers problem with readers priority is discussed in more detail. Section 4.2 gives hints at solutions of other versions. Two results are especially noteworthy: whereas the monitor concept cannot succeed in giving a solution for the general readers_writers problem, no satisfying solution for the readers_writers problem with pure first_in_first_out scheduling can be given in CSSA.

4.1 Readers Writers with Readers Priority

4.1.1 Problem Specification

Given a database with read and write access functions, the **general readers_writers problem** can be specified by the following constraints:

(1) write operations are mutually excluded with other write and read operations.
(2) read operations may execute in parallel.

Special versions of the general readers_writers problem are defined by adding one or more further restrictions. By stating

(3) read requests have priority over write requests

we get a specification for the readers_writers problem with readers priority.

4.1.2 CSSA Solution

In fig. 9, script DATA_BASE (3-22) implements the read and write access functions. The use of the third operation RD_ACKN (23-26) will become clear soon. Note, that this is the same database as was used for the protected buffer solution. In the same sense as described in 3.1, the protection module RDER_PRIO defines the synchronization code specific to the readers priority restriction. An instantiation of one particular database can only be achieved by sending a CREATE message including the requested size of the database to an agent of type RDER_PRIO (44-47). Read and write requests can only be adressed to this protector agent because no acquaintance to the database can be communicated to the outside world.

The functionning of the whole system is best described by showing that restrictions (1)-(3) from 4.1 are fulfilled.
To this aim, I first want to interpret the rather intuitive notions in these assertions according to the present concrete environment.

- A read or write **request** occurs when a RD or WR message arrives in the mailbox of the protector agent.

- Read operations $RD_1, RD_2, \ldots, RD_j$ execute in **parallel** iff their requests have been accepted and executed by the protector agent, but no "corresponding" acknowledgement has been received at port R_ACKN in the protector's write operation WR.

Hence, from the protector's point of view, these read operations are either

- just being transmitted between the protector and the database,
- in the mailbox of the database.
- just being executed in the database,
- already executed in the database.

Before receiving an acknowledgement at port R_ACKN, the protector has no information about the state of the database. In this case, we also say that $RD_1, RD_2, \ldots, RD_i$ are in flight (between the protector and the database).

Hence, this definition of parallel reading only allows for read requests being nondeterministically executed in the data base. As the execution of operations is indivisible and sequential, the database agent can only accept and answer read requests one after another. The execution is nondeterministic, because no one can guarantee that the read operations are accepted by the database in the same order they were received in the protector's mailbox or sent away by the user agents.

The notion of "mutual exclusion" is explained more precisely as:

- **Mutual exclusion** between read and write requests and between write operations themselves is guaranteed iff no other operation ever can be in flight together with a write operation.

Proof of assertion (1) [mutual exclusion]:

(1) It's easily shown that one write operation excludes other write operations because it must be acknowledged [**wait** W_ACKN **do** ...,(80)] before the next write request may be accepted. Receiving this acknowledgement guarantees that the database has accepted the write operation. So, a next write operation never can't be in flight together with a preceding write operation.

(2) In order to show that "write excludes reads" we have to prove that a write operation never can't be in flight together with read operations. According to the acknowledgement protocol described in (1), only read operations have to be taken into account.

The proof is by induction:

(2.1) When a protector agent is created, it first must accept a CREATE message [**initial** CREATION,90] resulting in a change to facet WRRS (46). Because the idle operation has no effect when enabled before the first write operation [FIRST_OPER = **true** in line 85) the protector stays in facet WRRS until a write request will be accepted. So, the first write operation clearly is not in flight together with read operations.

(2.2) Assume the k'th write operation was not in flight together with other read operations.
As an effect of its execution the protector's RDER_COUNT was reset to 0 (78). Having received an acknowledgement at port R_ACKN (77), the protector is assured that in the database RCOUNT has been reset to 0 (25).
Then the completion of the protector's WR operation results in a change to facet RDERS (85), since FIRST_WR and FIRST_OPER have been set to false (72,73).
Now, two cases have to be considered:

(2.2.1) No read operation is requested:
Then the idle operation in facet RDERS immediately leads back to facet WRRS (59).

(2.2.1.1) If no write operation is requested, the protector will loop back to facet RDERS. These facet changes will continue until either a write operation (2.2.1.2) or a read operation (2.2.2) will be requested.

(2.2.1.2) If a write operation is requested it will execute without any read operations being in flight.

(2.2.2) A read operation RD_1 is requested.
RD_1 will be accepted and RDER_COUNT will be set to 1 (56). As long as further read operations RD_i ($i > 1$) are requested before the protector has finished execution of read operation RD_{i-1}, they will be accepted and sent to the database.

(2.2.2.1) $\forall i \in N$.
($i > 0$ => read operation RD_i has been requested before RD_{i-1} has finished execution)

In this case, an infinite number of read operations will be transmitted to the database. Therefore, no further write operation will ever be executed and "the (k+1)-st write operation is not in flight with read operations" holds trivially.
(Being polite assume here, that CSSA integer variables can store any natural number. So our program never will abort with integer overflow.)

(2.2.2.2) Assume RD_j finishes before RD_{j+1} has been requested. Then the idle operation will be executed causing a change to facet WRRS.

(2.2.2.2.1) A write operation is not requested.
Then the idle operation leads back to facet RDERS with RDER_COUNT = j.
When new read requests have arrived they will be executed either ad infinitum (case 2.2.2.1) or until RD_{j+k} finishes before RD_{j+k+1} is requested. In the latter case we have the same situation as in 2.2.2.2 with RDER_COUNT = j +

k.

(2.2.2.2.2) Terminating the induction, we assume that a write operation (the (k+1)-st write operation) is requested.
RDER_COUNT > 0 is the number of read operations being in flight. When the message RD_ACKN(RDER_COUNT) is sent to the database (76), it will only be accepted when RDER_COUNT read operations have been executed in the database (**assert** COUNT = RCOUNT, 23). Therefore, the receipt of an acknowledgement at port R_ACKN guarantees that all read operations have been executed. When the write operation is sent to the database (86), no read operation is in flight. ▪

In this proof we already described a situation resulting in more than one read operation being in flight. So we have assertion (2) [parallel reading] automatically.

Assertion (3) [priority constraint] is easily shown:
Without loss of generality, assume that both a read request and a write request exists and at least one operation has been executed before. I have to prove that whatever operation read or write was executed last, the next request accepted by the protector will be the read request. Now consider the two cases:

(1) The last operation executed was a read operation:
Then we stay in facet RDERS, and obviously the read request will be satisfied next.

(3) The last operation executed was a write operation:
Because FIRST_WR in facet WRRS is false, no further write request is acceptable. Therefore, the protector will change to facet RDERS and execute the read operation.

4.1.3 Comparison with the Monitor Solution

It's rather difficult to compare two solutions when one cannot observe much similiarities. The most significant difference stems from the fact that in contrast to the monitor solution (fig.11a and 11b) I don't need a queue storing user's write and read requests in the CSSA solution.
In the CSSA environment, a request was defined to occur when the corresponding message is received in the protector's mailbox. In the monitor concept, a user request can only come into existence from the programmer's point of view, when the execution of the called monitor procedure is initiated. Translated into the monitor terminology, this is the time stamp when the request **enters the monitor**. However, from the monitor implementation's point of view, user requests already exist before the time stamps of their

procedure entry points. On the implementation level, incoming requests are registered and stored into an internal queue, which the programmer has no access to. In some sense, this initial queue and not the programmer-defined queues must be regarded as the monitor data structure corresponding to the CSSA mailbox. Hypothetically, a monitor solution similar to the CSSA solution could be derived if the following condition were met: The initial monitor queue is not a queue, rather it is a data structure with almost arbitrary programmable access functions. In CSSA we can select nearly arbitrary messages by defining suitable facet structures and assertions.

4.2 Other Versions of the Readers Writers Problem

Constraints (1) [mutual exclusion] and (2) [parallel reads] formulated in 4.1.1 are common to all versions of the readers_writers problem. Therefore, only a third constraint will be stated to give specifications for the following specializations of the general problem.

4.2.1 The Fair Readers Writers Problem

The readers_writers problem with readers_priority allowed writers to starve (cf. case 2.2.2.1 in the proof of assertion (1)). Substituting assertion (3) [readers_priority] by

(3a) eventually every request will be served

a solution is required to be fair against readers and writers. That is what I call the **Fair_Readers_Writers Problem**.
Note, that requirement (3a) makes no specific assumptions about the order of serving requests. Thus, the quality of a solution should be measured by the freedom that it leaves for selecting the next request to be served. For example, a pure first_in_first_out scheduling of requests fulfills the requirements, but is too restrictive to be regarded as a good solution.

A really satisfying CSSA solution can be derived from the RDER_PRIO script by simply placing the operations RD and WR into only one facet. The priority ensuring idle operations and the assertion FIRST_WR for the write operation are no more needed. With operations RD and WR occuring in one facet the selection of the next request is totally left to the mailbox manager. From the programmer's point of view this message selection strategy can only assumed to be fair and nothing more. Hence, the proposed CSSA solution fits the problem specification in the best possible way. Even more, this solution is perhaps the most natural and certainly the simplest among all solutions for other versions of the readers_writers problem.

It should be not surprising that - if at all - only a rather com-

plex and not efficient solution for the same problem can be programmed with monitors. Here, the only data structure for storing requests is a queue. Each signal statement can only release the first request of a queue. Hence, a monitor programmer can never reach the state of freedom of a CSSA programmer who can rely on the more general message selection strategy of the mailbox manager.

4.2.2 The FIFO Readers Writers Problem

Another version of the readers_writers problem is given by

(3b) each request has priority over all later requests.

Here, priority is entirely based on order of requests. Obviously, this version postulates a pure FIFO policy for serving read and write requests.

According to the definition of a "request" in 4.1.2, at each time the earliest message received has to be selected from the mailbox. However, in CSSA there is no mechanism which allows selection to be based on receive order. There is no hope to give a correct CSSA solution in the context of the definitions established in section 4.1.1. This context is important, because it is not generally impossible to program a FIFO solution in CSSA. So, I could require that each request is transmitted with a unique natural number indicating the time stamp when it had been sent away. Then the protector only accepts that message with the time stamp of the previously selected message incremented by one. Certainly, users would be obliged to coordinate in keeping the transmitted tags unique and successive. To this aim, before sending a request to the protector they could be forced to request a time stamp from a globally known "clock agent".

A monitor solution for the FIFO_READERS_WRITERS problem is presented in fig. 12. Although the monitor's queueing principle fits very well to the required FIFO strategy, the solution seems to be rather complex. This complexity is essentially due to the fact that at least the conventional monitor construct provides no means of identifying the process at the head of a queue or determining the conditions for which it is waiting. In fact, read and write requests are stored into a single queue (m.users) when they are not immediately satisfiable. But if a writer is dequeued when readers are in flight (readercount > 0), the writer must be queued again on a second queue until it is released by the last reader in the resource. A more profound discussion of these problems is given in [Bloom 79].

4.2.3 Readers Writers with Writers Priority

Substituting assertion (3) of the readers_writers problem with readers_priority by

(3c) write requests have priority over read requests

yields a specification for the readers_writers problem with writers_priority. The construction of a CSSA solution analogue to script RDER_PRIO is left to the reader as a simple exercise.

5. The Alarmclock

5.1 Problem Specification

The alarmclock is a very nice representative of the class of problems where synchronization is based on the **arguments** of requests.
It can be considered as an abstraction of a system device allowing users to fall asleep for some specified time period and to be awakened after this time has elapsed. The alarmclock cannot increment the time by itself, it rather waits for a tick request to be received from some other system facility.

5.2 CSSA Solution

In fig. 13, a CSSA solution is presented which looks very elegant but unfortunately is wrong. Notwithstanding, I enclosed this bad solution because it's a good demonstration of the dangers waiting for a programmer in the distributed environment. So I admit, that after having programmed this solution I accepted it as correct until the last moment before publishing this report.
Although perhaps not visible in their surface structures, the ideas behind the wrong solution and the correct one (fig. 14) are much the same. Therefore, I can afford some documenting of the wrong solution without wasting time.

An agent wishing to sleep for T time units sends a message WAKE_ME(T) to an agent of type ALARM_CLOCK [ALARM_CLOCK for short]. When the WAKE_ME message is accepted by ALARM_CLOCK the sleepy agent is immediately resumed if DELAY_TIME (=T) ≤ 0 (23). For a DELAY_TIME > 0 , the wake request is registered by sending a private message WAKE_UP(DELAY_TIME + NOW) to self (24). The parameter DELAY_TIME + NOW yields the "absolute" time stamp when the sleeping agent has to be awakened. Later the existence of this WAKE_UP message in its mailbox will remind ALARM_CLOCK of the user's reveille. Because ringing is symbolized by sending a reply, the port address implicitly included in the WAKE_ME message must be transfered to the WAKE_UP message via the **inherit** - clause. Now we arrive at the crucial point of the discussion. Each execution of a TICK request increments the "real time" NOW by 1 (28) and sets up a new instance of facet RESUME. The idea is, that in this facet **all** WAKE_UP requests for the current time (**assert** TIME = _NOW, 11) should be satisfied. When no such further WAKE_UP request exists, the execution of the idle operation (15,16) leads back to facet REGISTRY.

At first sight, this strategy seems very sound. But did you get on to the problem? So assume for example, that a WAKE_ME message with DELAY_TIME = 1 has been received and in response ALARM_CLOCK has sent a WAKE_UP message with parameter NOW + 1 to itself. After the next execution of a TICK operation, this WAKE_UP message must be served in facet RESUME. Now, we are faced to the problem that

ALARM_CLOCK stays in facet RESUME only until no (further) WAKE_UP message with TIME = NOW exists in the **mailbox**. When the considered WAKE_UP request has already been received in the mailbox, it will be executed. However, if it has not been delivered for the time being, the current facet will be left via the idle operation and the sleeping agent will sleep forever.
Although from a practical point of view, the probability may be very low that a WAKE_UP message will not have been received when it is needed, this program must be rejected as a wrong solution. The computational model of CSSA only requires message passing times to be finite, but indefinite. That's even true in this case when messages are passed to the sending agent itself.

Obviously, the idle opeation is not the right criterion for leaving facet RESUME. Instead it must be assured that all WAKE_UP requests for the current time value NOW are served before this facet is left. In the "hopefully" correct solution depicted in fig. 14, this is achieved by keeping a relation CALL_LIST recording the **number of agents to be awakened** (NO_OF_SLEEPERS) for each time stamp which ever has been requested.
I.e., a request WAKE_ME(DELAY_TIME) is served as follows: when another wakeup obligation for time DELALAY_TIME + NOW exists, the corresponding call element (of type CALL_ELEM) is assigned to the implicitly declared variable CE1,(33) and its NO_OF_SLEEPERS' part is incremented by 1 (34). Otherwise, a new call element with "key" WAKE_UP_TIME = DELAY_TIME + NOW and NO_OF_SLEEPERS = 1 is inserted into the relation (36-38).
In both cases, ALARM_CLOCK sends a WAKE_UP message to itself (40). These private messages are still necessary since for reasons not to be discussed here, the **inherit** clause is the only mechanism available for saving a reply address from one operation execution to another. In this example, however, the use of private messages could be avoided, when ringing would be implemented by "normal" wakeup messages instead of reply messages. Then, in addition to the DELAY_TIME parameter, a sleepy agent must include an acquaintance to itself in the WAKE_ME message, which later will be used by ALARM_CLOCK as the addressee of the corresponding WAKE_UP message.

After incrementing the time (45), a new instantiation of facet RESUME is only set up when a call element for the new time value exists (46). Compared with the first solution this means a slight compensation for the overhead we had to introduce in the second solution. ALARM_CLOCK then stays in facet RESUME, waiting for the first WAKE_UP request which eventually will arrive in the mailbox, if it was not already there. So, there is no need for an idle operation in this case. Another reason is that the criterion for leaving facet RESUME can be computed **during** the execution of a WAKE_UP operation: If NO_OF_SLEEPERS for the currently served call element has the value 1 (19), then no further wakeup obligation exists for the current time value and facet RESUME may be left. Besides, the current call element can be deleted (20). Otherwise, NO_OF_SLEEPERS is decremented by 1 (22) and no change of facet takes place.

5.3 Comparison with the Monitor Solution

In monitors, queuing is the only means to stop serving a request which later shall be resumed. Because in this example delayed requests have to be resumed in the order of their arguments, the concept of **priority queues** has to be introduced (cf. fig. 16). Most of the alarmclock code in the CSSA solution is needed to implement this data structure which is a necessary prerequisite of all monitor implementation languages. From this point of view, the seemingly more complex CSSA solution doesn't make me so unhappy, rather it indicates the provision of greater freedom and flexibility in the CSSA concepts.

An unpleasantness of this special monitor solution is due to the fact that it is not possible to examine the first element of a queue without dequeueing it first. Thus, the first process in the priority queue has to be dequeued in each tick operation, then tested for its arguments (**while** ac.now < alarmsetting), and possibly requeued again. As already mentioned in 5.2, this awkwardness can be avoided in the CSSA solution by **not** establishing an instance of facet RESUME when no wakeup obligation exists for the current time. Clearly, this shortcoming of the monitor solution is not so severe because adding a test for the priority of the first element in a priority queue as an additional language feature is all what has to be done.

6. The N-Queens Problem

6.1 Problem Specification

The following problem has to be solved:
How can you place N queens on an NxN chessboard in such a way that no two queens can capture each other (i.e. no two queens are in the same row, column or diagonal)?
The CSSA solution described in the next section is expected to find all such possible placings for any given N.
In general, the N-queens problem is a showboy for the class of problems offering "natural" recursive solutions. In some sense, the CSSA script to be developed also defines recursive behaviour. Instead of recursive function calls, agents of a certain type are created dynamically and provided with partial solutions which they are ordered to expand.

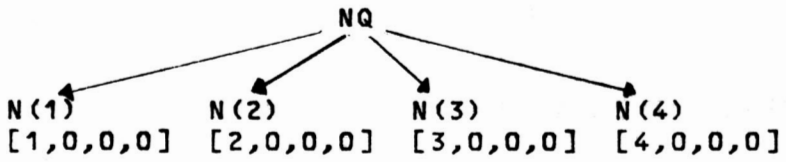
6.2 CSSA Solution

The script NQUEENS in fig. 17 is best described by setting an example for one specific N, say N=4.
After having created an agent NQ of type NQUEENS, a user specifies the command

```
send FIND_SOLUTION(4) to NQ
```

Because N does not equal 1 (which would lead to the immediate answer: "For one queen solution is : 1" [75-79]), NQ sends a message FIND_SOLUTION to each of four newly created anonymous agents of type NODE [84]. We want to identify these new agents as N(1),N(2),N(3) and N(4). Each agent N(i), i=1,...,4 , is provided with the information that a queen has been placed by NQ in column i of the first row.This information is represented in the CSSA script by putting the value i into the first element of the array FIRST_ROW [83]. The other array elements retain the value 0 due to default initialization [71].

In the following course of execution nothing remains to be done for NQ. One can terminate it by sending a STOPQ message [89-93], or let it survive when solutions for other sizes of the chessboard shall be generated.
The current situation is skeletonized in the following tree structure:



For example, N(3) is ordered to find a solution for the 4-queens problem with already one queen placed in column 3 of the first row.

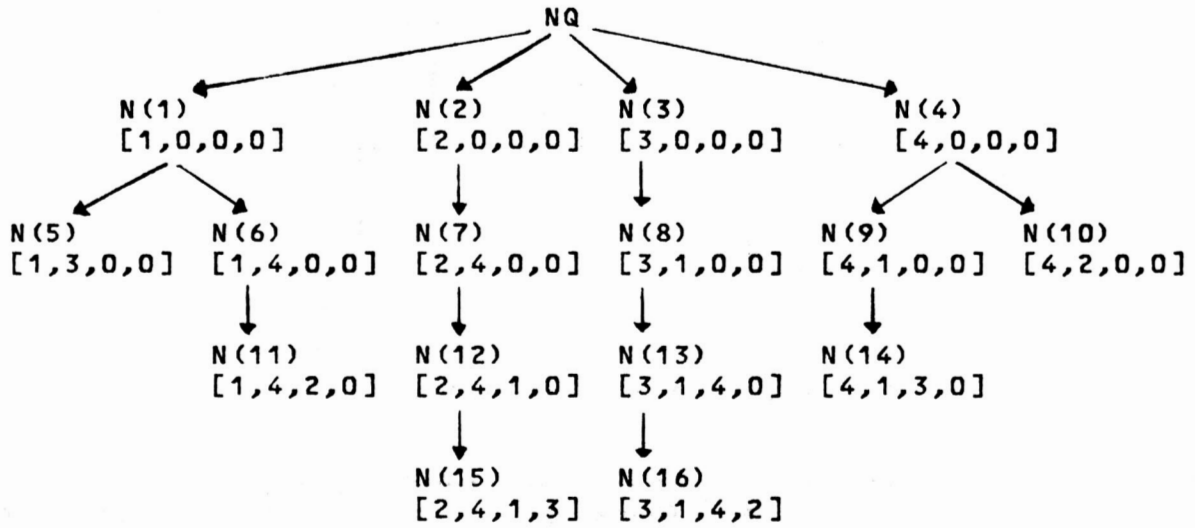
The first action of each agent N(i) is to check whether it already was provided with a complete solution. This would be the case if a queen had been placed in some column of the last (N-th) row [if **not** (Q(N) = 0),19].
The actions to be taken in this case are described later in this section when the condition will be fulfilled for our example execution. Now, for all agents N(i) this condition is not fulfilled. So, they try to find allowable places for the next queen to be placed in the current (second) row. This computation is done in three separated loops.

The first loop [34-36] determines how many queens (ANZ_Q) are already on the board. In our example, ANZ_Q will have the value 1 after leaving the loop.

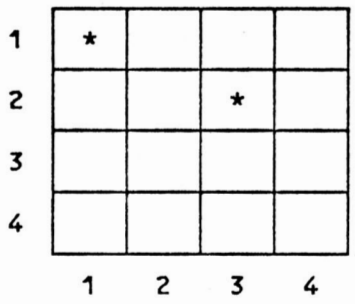
In the second loop [38-46], all columns of the current row are computed, which are **not** allowed for the next queen. To remember this information, the j-th element ($j \in \{1,...,4\}$) of the boolean array NOT_POSSIBLE will become **true** iff no queen may be placed into the j-th column of the current row.

In the third loop [48-53], a new agent of type NODE will be created, one for each allowed extension of the board. If no extension is possible (i.e. NOT_POSSIBLE (j) for j=1,...,4), the loop has no effect.

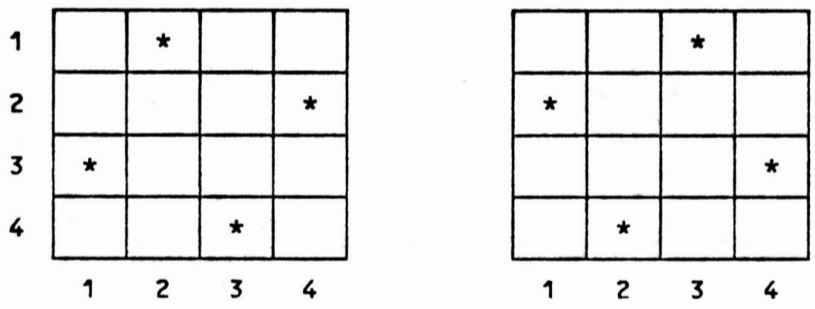
The last action of each NODE agent is to request its own termination (56). Hence, all NODE agents are alive only as long as necessary. After their work has been done they release their processors for the execution of other NODE agents which they possibly have created during their own lifetimes.
The coming into existence of new agents in our example execution is represented in the following tree structure which is an expansion of the previous one.



During the construction of the solution 16 NODE agents have been created. However, due to the self destruction of NODE agents this number should not be misunderstood as the number of agents executing simultaneously on the available processors.
Agents N(5),N(11),N(14) and N(10) terminate without finding an extension of the given partial solution. For example, N(5) is ordered to place a third queen into the third row with respect to the partial solution depicted by the following board.



Obviously, this board cannot be expanded, and no new agent will be created. Only agents N(15) and N(16) are provided with complete solutions:



(boards for [2,4,1,3] and [3,1,4,2]

For their executions the previously mentioned condition **not** (Q(N) = 0) is fulfilled. Therefore, their only task is to send a message of success to the user. The envelope of this message is constructed using the string variable ASW and sent to the user (interface) by

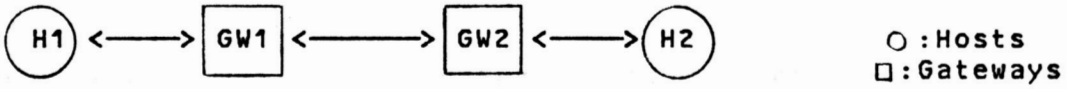
```
send ANSWER(ASW) to interface reply to OK;
```

The reply obligation is not really necessary. Its only purpose is to enhance the attention of the human user observing or not observing messages being displayed on the screen.

7. Window Mechanism

7.1 The Model

In this example CSSA is used to model the following computer system:

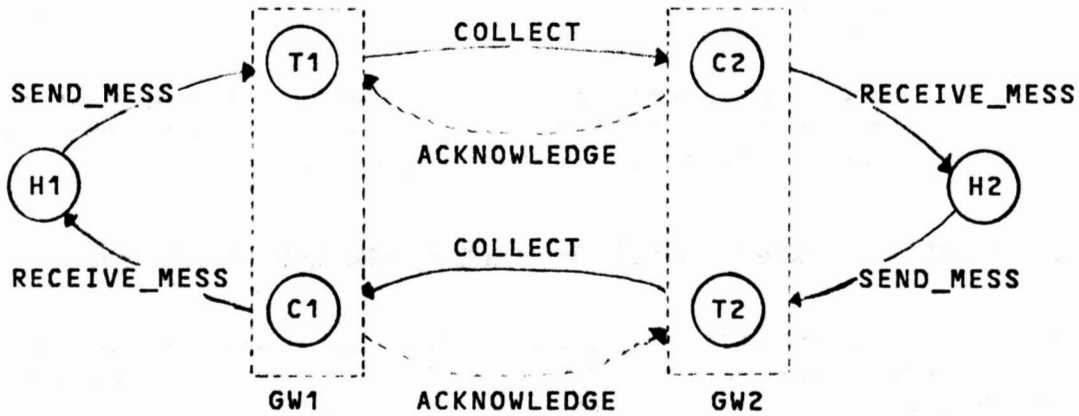


Two host computers communicate via two interface modules which are called gateways. Hi , i∈{1,2} , can send (receive) a complete message to (from) GWi whereas only packets are transmitted and received between GW1 and GW2.
In the model, messages are character strings of variable lengths and the individual characters of the strings are packets.

Each gateway provides two facilities:

- a transmitter (T) as one part of the gateway receives a message m from "its" host and sends the characters of m to the collector in the other gateway.
- a collector (C) as the other part receives packets from the transmitter in the other gateway. When the packets of one message are all received the concatenation of the packets is sent to the host as one message.

We assume that several packets may be simultaneously in flight between T and C. This extended model is described by the following picture:



In the CSSA solution all circled entities will become agents which communicate via the operations labelling the arcs.
So H1 can send a message via T1 and C2 to H2 at the same time when H2 is sending a message via T2 and C1 to H1. Therefore the most effective realization of this CSSA agent structure would supply one (micro-)processor for each of the Ti's and the Ci's.
Now we introduce the following realistic complications:

- packets need not reach the collector in the order they were produced by the transmitter; that means packets can pass each other;

- the transmission medium between the two gateways is unreliable so that packets can be totally lost.
  (It should be noted that this second asssumption is in contrast to the CSSA model of computation which does not allow the loss of messages. In the CSSA solution we therefore simulated this unreliable transmission medium by generating two agents of type FAULTY-CHANNEL (cf. fig. 19, 353-442), which receive and normally forward all messages transmitted between a collector and a transmitter. However, depending on generated random numbers, some messages are received but not transmitted, that means they are lost).

In this situation the communication between a transmitter and a collector becomes most interesting.
We are faced with the problem of providing the hosts H1 and H2 with a reliable "virtual" transmission medium which is based on an unreliable "physical" transmission medium.

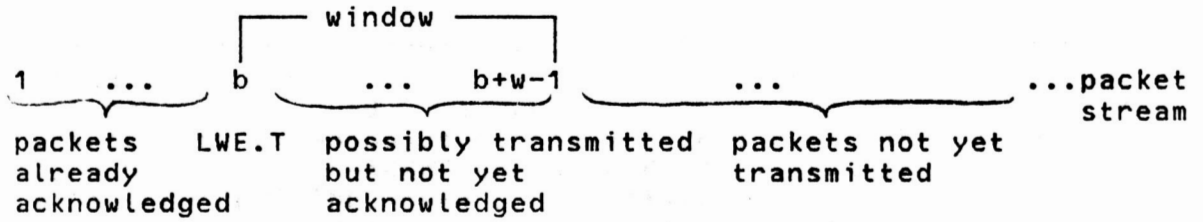
In a simple solution of this problem the transmitter T would send a packet p1 to the collector C, then wait for an acknowledgement, then send a second packet p2, etc. Because the transmission medium is unreliable not alone for packets but also for acknowledgements, some timeout mechanism must be used when waiting for an acknowledgement. When time is exhausted the packet not yet acknowledged must be sent once more. The implementation of receiving duplics is straightforward in this case.

However, this solution is not really satisfying, because it does not utilize the whole capacity of the transmission medium (more than one packet may be in flight between T and C).

7.2 Communication between the Transmitter and the Collector

The CSSA scripts TRANSMITTER and COLLECTOR implement a mechanism which is described informally in [Cerf,Kahn 74]. This so called **window mechanism** works as follows:
The transmitter sends packets (in operation SEND_PACKET, 71-88) to the collector. Received packets (operation COLLECT, 156-247) are acknowledged by the collector. Instead of waiting for an acknowledgement (operation ACKNOWLEDGE, 91-116) before the next packet can be transmitted, the transmitter can hurry on ahead; so several

packets can be in flight without having been acknowledged. Obviously, every packet must carry a unique identification (sequence number) which allows the reconstruction of a message from the individual packets. In order to guarantee the perception of lost packets, the transmitter eventually must stop transmitting new packets. Therefore a common upper bound for the maximum number of packets in flight is agreed upon between the sender and the receiver. This upper bound w is the size of the so called **window**. In the window appears a part of the packet stream which starts just behind the last packet already transmitted and acknowledged, if any. The variable LWE.T is equal to the sequence number of the leftmost packet within the transmitter's window (left window edge).



(the transmitter's window)

If all packets of the current window have been transmitted a next packet can only be transmitted after having received an acknowledgement a with LWE.T < a < LWE.T + W. The receipt of a indicates that the collector has accepted all packets with sequence numbers less than a and now is waiting for the packet with number a. This allows LWE.T to be shifted to the right (LWE.T := a), thereby implicitly advancing the right window edge.
However if a packet -or the acknowledgement itself- is lost, the transmitter never would get the chance of transmitting a new packet. Therefore some timeout mechanism must be used leading to the **assumption** that a packet or an acknowledgement has been lost. Based on this hypothesis the packet at the left window edge will be transmitted for a second time.
In our CSSA solution the retransmission of a packet is processed in the idle_operation (60-66) of facet TRANSMISSION2 in script TRANSMITTER.
Here we use the idle operation as a simulation of a timeout mechanism, since no real time constructs are available in the current version of CSSA.
An immediately subsequent retransmission of the same packet due to the timeout mechanism is prohibited by setting the boolean variable IDLE_ENABLED to true and changing to facet TRANSMISSION1. So, a next activation of the idle operation becomes only effective after an useful acknowledgement advancing the left window edge has been received. In this implementation, every packet will be transmitted at most two times. It should be noted that this number is not the result of a principal decision. A maximal retransmission rate of 3,4 or 100 times could have been programmed with the same ease.

In facet TRANSMISSION the transmitter works according to the following rules

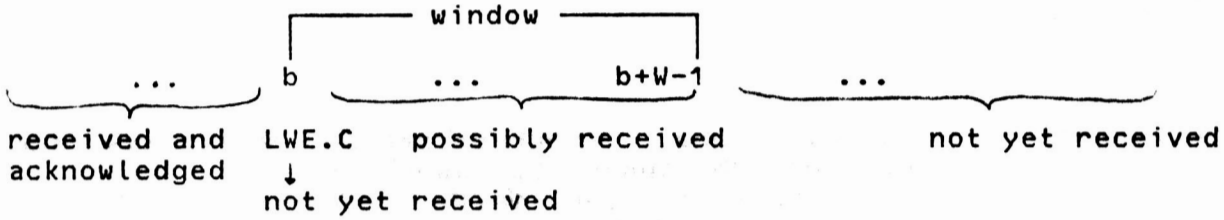
1. The transmitter transmits packets with sequence numbers between LWE.T and up to LWE.T + W - 1.
   Each activation of the operation SEND_PACKET transmits exactly one packet (81). The whole transmission mechanism is initiated by the operation SEND_MESS (25-38) in facet START_TRANSMISSION in which the transmitter sends a "sufficient" number of SEND_PACKET-messages to itself (via procedure ACTIVATE_TRANSMIT, 8-15).
   Later new SEND_PACKET-messages are only produced in operation ACKNOWLEDGE (98,99). The last packet of the currently transmitted "host-message" is provided with the tag ENDE = true (77).

2. The receipt of an acknowledgement with number N indicates that the collector has received all packets with sequence numbers less than N and the packet with number N being the first one not yet received. Note that the acknowledgement N implicitly acknowledges all packets with sequence number less than N. If LWE.T < N ≤ LWE.T + W, the left window edge LWE.T can be set to N (103). Assuming the rest of the message to be transmitted is long enough, N-LWE.T new SEND_PACKET-messages can be produced (98); otherwise the transmitting of all remaining packets of the current message is initiated (99). If the idle operation was disabled before receipt of the acknowledgement, now it will be enabled (104-107).
   An acknowledgement with N ≤ LWE.T is possible if it was overtaken by a later one with a higher number. Such an acknowledgement can simply be discarded. Only an information message will be sent to the interface in this case.
   An acknowledgement with N > LWE.T+W is not possible because at most packet number LWE.T+W-1 can have been transmitted and received before.

An acknowledgement for a packet of an old message (MESS_ID < CURRENT_MESS_ID, 92) has no effect at all.

3. On timeout (no ACKNOWLEDGE- or SEND_PACKET - message can be enabled) the packet with sequence number LWE.T will be retransmitted if the current facet is TRANSMISSION2.

The left edge LWE.C of the collector's window (fig. 20) denotes the least packets number which has not yet been received.



(the collector's window)

Possibly received packets with sequence numbers greater than LWE.C up to the right window edge LWE.C + W - 1 (window packets) are stored in the relation R under their (relative) "window position". Depending on the received packet number CHAR_NO in operation COLLECT the following actions take place:

1. CHAR_NO = LWE.C (179-242):
   Let J be the unique number with 0 < J < W such that all packets with sequence numbers less or equal to LWE.C + J have been received but the packet with number LWE.C + J + 1 not yet being received. Then the currently received first part of the "host-message" can be prolonged by the substring defined by packet numbers LWE.C,LWE.C+1,...,LWE.C + J (194-205).
   The window is adjusted by setting LWE.C to LWE.C + J + 1 and the relation R is updated as an image of the new window (217-236).
   As acknowledgement the number of the new left window edge is sent to the transmitter (207). This implicitly acknowledges packets with numbers less than the index of the new left window edge. When the whole "host-message" has been received it is transmitted to the destination host (184,209-210).

2. LWE.C < CHAR_NO ≤ LWE.C + W - 1:

   The currently received packet is not yet acknowledged but stored in the relation R (173-176). The window is not "moved".

3. CHAR_NO < LWE.C:

   The currently received packet is a duplicate the original of which already has been received. From the collector's point of view the receipt of a duplicate serves as a hint that the (implicit or explicit) acknowledgement for this packet might have been lost. Therefore the current left window edge LWE.C is sent as acknowledgement once again (243).

4. CHAR_NO > LWE.C + W - 1 is not possible.

LITERATURE

[Bloom 79] Bloom, Toby : Synchronization Mechanisms for Modular Programming Languages
MIT/LCS/TR-211, Jan. 1979

[Lis,Sny, Atk,Sch 77] Liskov,B.H.; Snyder,A.; Atkinson,R.;Schaffert,C. : Abstraction Mechanisms in CLU
Comm.ACM (20,8), Aug. 1977, pp564-576

[FRV 81] Fischer,Hans Ludwig; Raulefs,Peter; Voss,Hans : CSSA-Projekt, Arbeitsbericht April 1979 - März 1981
Memo SEKI-BN-81-02, Institut für Informatik, Universität Bonn, 1981 (in german)

[FRV 82] The Programming Language CSSA for Multi-computersystems
SEKI Report, Institut für Informatik III, Universität Bonn, 1982

[Cerf,Kahn 74] Cerf,V.G.; Kahn,R.E. : A Protocol for Packet Network Intercommunication
IEEE Transactions on Communications, Vol-Com-22, No.5, May 1974, pp637-648

APPENDIX

```
B M S - C S S A - C O M P I L E R                                    1981/12/01   11:1!

DEFAULT-OPTIONS:  NOTERM,NOTEST,  CHECK,  OBJECT,  SOURCE,NOSTRUCT,  XREF,RESWD=A,MAXP= 99,MAXE= 99,MAXD=
 OPTIONS IN USE:  NOTERM,NOTEST,  CHECK,  OBJECT,  SOURCE,  STRUCT,  XREF,RESWD=A,MAXP= 99,MAXE= 99,MAXD=

     |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
   1 |type ONE_SLOT_BUFFER is script                                          |  |+1
   2 |                        |                                               |  |
   3 |  var string: BUFFER;                                                   |  |
   4 |                        |                                               |  |
   5 |  facethead READ;                                                       |  |
   6 |                        |                                               |  |
   7 |  facet WRITE  is                                                       |  |+2
   8 |  | public: WR;                                                         |  |
   9 |  |                     |                                               |  |
  10 |  | operation WR(string: MESSAGE) is                                    |  |+3
  11 |  | | BUFFER := MESSAGE;                                                |  |
  12 |  | | replace by READ;                                                  |  |
  13 |  | endoperation                                                        |  |-3
  14 |  endfacet                                                              |  |-2
  15 |                        |                                               |  |*1
  16 |  facet READ is                                                         |  |+4
  17 |  | public: RD;                                                         |  |
  18 |  |                     |                                               |  |
  19 |  | operation RD is                                                     |  |+5
  20 |  | | reply (BUFFER);                                                   |  |
  21 |  | | replace by WRITE;                                                 |  |
  22 |  | endoperation                                                        |  |-5
  23 |  endfacet                                                              |  |-4
  24 |                        |                                               |  |*1
  25 |  initial WRITE                                                         |  |
  26 |                        |                                               |  |
  27 |endscript                                                               |  |-1

  BMS-CSSA-COMPILER - DATE OF RELEASE: 30 SEP 1981  NO ERROR DETECTED
  END OF COMPILING ON 1981/12/01 AT 11:16:06.00    RETURNCODE = 0
  COMPILE-TIME (CPU) =    0.85 SEC.    EXECUTION-TIME = 15.00 SEC.
  NUMBER OF SOURCE-LINES READ =   27     NUMBER OF TOKENS =    62
  NUMBER OF OBJECT-RECORDS GENERATED =    302


                    *** CROSS-REFERENCE-TABLE ***

BUFFER        VARIABLE:STRING                    3    11    20
MESSAGE       CONSTANT:STRING                   10    11
ONE_SLOT_BUF=USER_DEFINED_TYPE:SCRIPT            1
RD            OPERATION:OPER                    17    19
READ          FACET                              5    12    16
WR            OPERATION:OPER                     8    10
WRITE         FACET                              7    21    25

 - MULTIPLE OCCURENCES ON THE SAME LINE ARE MARKED WITH '+'.

 - TOTAL NUMBER OF IDENTIFIERS USED IN THIS PROGRAM:   7
```

fig. 1 : One Slot Buffer Script

```
C S S A - S I M U L A T I O N - S Y S T E M

                    P1  (INTERFACE)

                  P R O C E S S O R
          P2        CONFIGURATION        P3
     P4         P5          P6          P7
```

```
          PROGRAM GENERATED ON 1981/12/01 AT 11:15:51.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1981/12/01 AT 11:48:14.00
          ====================================================

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> DISPLAY                                                          <==

| IDENTIFIER          | TYPE          | VALUE               |
|---------------------|---------------|---------------------|
| ONE_SLOT_BUFFER     | SCRIPT        | ONE_SLOT_BUFFER     |

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : OSB := NEW ONE_SLOT_BUFFER                           <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OPER : RD,WR; PORT : P                                           <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE; SEND RD TO OSB REPLY TO P                               <==
+++      0.000  P1: INTERFACE(1) SENDS RD(),REPLY TO: P TO ONE_SLOT_BUFFER(1)
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                             <==
+++      0.000  P2: ONE_SLOT_BUFFER(1) IS IDLE
+++      0.100  P2: ONE_SLOT_BUFFER(1) RECEIVES RD(),REPLY TO: P
                        FROM INTERFACE(1)
+++      0.100  P2: ONE_SLOT_BUFFER(1) IS IDLE
>>>      0.100  SYSTEM TERMINATED
>>>      0.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND WR("A") TO OSB;RUN                                          <==
+++      0.100  P1: INTERFACE(1) SENDS WR(STRING:"A") TO ONE_SLOT_BUFFER(1)
+++      0.200  P2: ONE_SLOT_BUFFER(1) RECEIVES WR(STRING:"A")
                        FROM INTERFACE(1)
+++      0.200  P2: ONE_SLOT_BUFFER(1) STARTING OPERATION WR(STRING:
                        "A")
+++      0.400  P2: ONE_SLOT_BUFFER(1) PERFORMS FACETTING : WRITE --> READ
+++      0.400  P2: ONE_SLOT_BUFFER(1) STARTING OPERATION RD()
+++      0.500  P2: ONE_SLOT_BUFFER(1) SENDS *REPLY(STRING:"A")
                        TO INTERFACE(1)
+++      0.600  P2: ONE_SLOT_BUFFER(1) PERFORMS FACETTING : READ --> WRITE
+++      0.600  P2: ONE_SLOT_BUFFER(1) IS IDLE
ONE_SLOT_BUFFER(1)
>>>      1.100  SYSTEM TERMINATED
>>>      1.100  INTERFACE(1) : ENTER CSSA COMMAND -
```

fig. 2 : One Slot Buffer Execution Protocol

```
==> MAILBOX                                                              <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
-------------------------------------------------------------------------

>>>      1.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND RD TO OSB REPLY TO P; SEND WR("B") TO OSB;RUN                   <==
+++      1.100  P1: INTERFACE(1) SENDS RD(),REPLY TO: P TO ONE_SLOT_BUFFER(1)
+++      1.100  P1: INTERFACE(1) SENDS WR(STRING:"B") TO ONE_SLOT_BUFFER(1)
+++      1.200  P2: ONE_SLOT_BUFFER(1) RECEIVES RD(),REPLY TO: P
                    FROM INTERFACE(1)
+++      1.200  P2: ONE_SLOT_BUFFER(1) IS IDLE
+++      1.300  P2: ONE_SLOT_BUFFER(1) RECEIVES WR(STRING:"B")
                    FROM INTERFACE(1)
+++      1.300  P2: ONE_SLOT_BUFFER(1) STARTING OPERATION WR(STRING:
                    "B")
+++      1.500  P2: ONE_SLOT_BUFFER(1) PERFORMS FACETTING : WRITE --> READ
+++      1.500  P2: ONE_SLOT_BUFFER(1) STARTING OPERATION RD()
+++      1.600  P2: ONE_SLOT_BUFFER(1) SENDS *REPLY(STRING:"B")
                    TO INTERFACE(1)
+++      1.700  P2: ONE_SLOT_BUFFER(1) PERFORMS FACETTING : READ --> WRITE
+++      1.700  P2: ONE_SLOT_BUFFER(1) IS IDLE
ONE_SLOT_BUFFER(1)
>>>      2.200  SYSTEM TERMINATED
>>>      2.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                              <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
(2) *REPLY(STRING:"B")
-------------------------------------------------------------------------

>>>      2.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> DUMP OSB                                                             <==

+++      2.200  RUNTIME STACK OF ONE_SLOT_BUFFER(1)

                 +-------------------------------+
                 | FACET  WRITE                  |
                 | ENV: SCRIPT  ONE_SLOT_BUFFER  |
                 | LINE:    7                    |
                 |...............................|
                 | SCRIPT  ONE_SLOT_BUFFER       |
                 | ENV:                          |
                 | LINE:   27                    |
                 |...............................|
                 | BUFFER = "B"                  |
                 +-------------------------------+

>>>      2.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                            <==
+++      2.200  ALL EXISTING AGENTS:

+-----------------+---------+-----------+-----------------+
| AGENT           | FACET   | OPERATION | MAILBOX         |
+-----------------+---------+-----------+-----------------+
| P1: INTERFACE(1) *|       |           | *REPLY *REPLY   |
| P2: ONE_SLOT_BUFF | WRITE |           |                 |
+-----------------+---------+-----------+-----------------+

                  CSSA-SESSION-STATISTICS
                  =======================

SESSION  STARTED  AT  11:48:01.00
SESSION TERMINATED AT 11:51:08.00  ON 1981/12/01
REAL-TIME USED :   188.00 SEC.
CPU-TIME  USED :     0.88 SEC.
SIMULATION TIME USED :       2.2000 SEC.
NUMBER OF AGENTS CREATED :     1
NUMBER OF MESSAGES SENT :      6
```

fig. 2 -continued-

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| BLOCKNESTING
  1 |type BOUNDED_BUFFER is script(int: MAX) assert MAX > 0                   | |+1
  2 |                       |                                                 | |
  3 |  public: INS,REM;     |                                                 | |
  4 |                       |                                                 | |
  5 |  functionhead MOD(int: P1,P2) returns int external;                     | |
  6 |  var array (0..MAX-1) of string: BUFFER;                                | |
  7 |  var int: INS_COUNT,REM_COUNT := 0;                                     | |
  8 |                       |                                                 | |
  9 |  facethead BUFFER_EMPTY;                                                | |
 10 |  facethead BUFFER_FULL;                                                 | |
 11 |  facethead NO_CONSTRAINT;                                               | |
 12 |                       |                                                 | |
 13 |  facet BUFFER_EMPTY is                                                  | |+2
 14 |  | include: INS;                                                        | ||
 15 |  endfacet                                                               | |-2
 16 |                       |                                                 |*1
 17 |  facet BUFFER_FULL is                                                   | |+3
 18 |  | include: REM;                                                        | ||
 19 |  endfacet                                                               | |-3
 20 |                       |                                                 |*1
 21 |  facet NO_CONSTRAINT is                                                 | |+4
 22 |  | include: INS,REM;                                                    | ||
 23 |  endfacet                                                               | |-4
 24 |                       |                                                 |*1
 25 |  operation INS(string: MESSAGE) is                                      | |+5
 26 |  | BUFFER(INS_COUNT) := MESSAGE;                                        | ||
 27 |  | INS_COUNT := MOD(INS_COUNT + 1,MAX);                                 | ||
 28 |  | if INS_COUNT = REM_COUNT then replace by BUFFER_FULL;                | | |+6
 29 |  | else replace by NO_CONSTRAINT; endif;                                | | |-6
 30 |  endoperation                                                           | |-5
 31 |                       |                                                 |*1
 32 |  operation REM is                                                       | |+7
 33 |  | reply (BUFFER(REM_COUNT));                                           | ||
 34 |  | REM_COUNT := MOD(REM_COUNT + 1,MAX);                                 | ||
 35 |  | if INS_COUNT = REM_COUNT then replace by BUFFER_EMPTY;               | | |+8
 36 |  | else replace by NO_CONSTRAINT; endif;                                | | |-8
 37 |  endoperation                                                           | |-7
 38 |                       |                                                 |*1
 39 |                       |                                                 | |
 40 |  initial BUFFER_EMPTY                                                   | |
 41 |                       |                                                 | |
 42 |endscript                                                                | |-1
```

fig. 3 : Bounded Buffer Script

```
          PROGRAM GENERATED ON 1981/11/26 AT 11:25:31.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1981/11/27 AT 11:11:40.00
          ====================================================

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : BB := NEW BOUNDED_BUFFER(3)                                    <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OPER : INS,REM; PORT : P                                                   <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;DISPLAY                                                                   <==
```

IDENTIFIER	TYPE	VALUE
BB	AGENT	BOUNDED_BUFFER(1)
BOUNDED_BUFFER	SCRIPT	BOUNDED_BUFFER
INS	LITERAL	
P	PORT	
REM	LITERAL	

```
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE; SEND INS("111") TO BB; SEND INS("222") TO BB;RUN                  <==
+++      0.000  P1: INTERFACE(1) SENDS INS(STRING:"111") TO BOUNDED_BUFFER(1)
+++      0.000  P1: INTERFACE(1) SENDS INS(STRING:"222") TO BOUNDED_BUFFER(1)
+++      0.000  P2: BOUNDED_BUFFER(1) IS IDLE
+++      0.100  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"111")
                    FROM INTERFACE(1)
+++      0.100  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "111")
NO_CONSTRAINT
+++      0.500  P2: BOUNDED_BUFFER(1) IS IDLE
+++      0.500  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"222")
                    FROM INTERFACE(1)
+++      0.500  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "222")
NO_CONSTRAINT
+++      0.900  P2: BOUNDED_BUFFER(1) IS IDLE
>>>      0.900  SYSTEM TERMINATED
>>>      0.900  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND REM TO BB REPLY TO P; SEND REM TO BB REPLY TO P;SEND REM TO BB REPLY TO P   <==
+++      0.900  P1: INTERFACE(1) SENDS REM(),REPLY TO: P TO BOUNDED_BUFFER(1)
+++      0.900  P1: INTERFACE(1) SENDS REM(),REPLY TO: P TO BOUNDED_BUFFER(1)
+++      0.900  P1: INTERFACE(1) SENDS REM(),REPLY TO: P TO BOUNDED_BUFFER(1)
>>>      0.900  INTERFACE(1) : ENTER CSSA COMMAND -
==> RUN; MAILBOX; MAILBOX BB                                                   <==
+++      1.000  P2: BOUNDED_BUFFER(1) RECEIVES REM(),REPLY TO: P
                    FROM INTERFACE(1)
+++      1.000  P2: BOUNDED_BUFFER(1) STARTING OPERATION REM()
+++      1.100  P2: BOUNDED_BUFFER(1) SENDS *REPLY(STRING:"111")
                    TO INTERFACE(1)
+++      1.100  P2: BOUNDED_BUFFER(1) RECEIVES REM(),REPLY TO: P
                    FROM INTERFACE(1)
NO_CONSTRAINT
+++      1.400  P2: BOUNDED_BUFFER(1) STARTING OPERATION REM()
+++      1.500  P2: BOUNDED_BUFFER(1) SENDS *REPLY(STRING:"222")
                    TO INTERFACE(1)
+++      1.500  P2: BOUNDED_BUFFER(1) RECEIVES REM(),REPLY TO: P
                    FROM INTERFACE(1)
BUFFER_EMPTY
+++      1.800  P2: BOUNDED_BUFFER(1) IS IDLE
+++      1.900  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"111")
                    FROM BOUNDED_BUFFER(1)
+++      1.900  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"222")
                    FROM BOUNDED_BUFFER(1)
>>>      2.000  SYSTEM TERMINATED
```

fig. 4 : Bounded Buffer Execution Protocol

```
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"111")
(2) *REPLY(STRING:"222")
-------------------------------------------------------------------------------

MAILBOX OF BOUNDED_BUFFER(1) :
-------------------------------------------------------------------------------
(1) REM(),REPLY TO: P
-------------------------------------------------------------------------------

>>>      2.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND INS("333") TO BB; SEND INS("444") TO BB; SEND INS("555") TO BB;RUN      <==
+++      2.000  P1: INTERFACE(1) SENDS INS(STRING:"333") TO BOUNDED_BUFFER(1)
+++      2.000  P1: INTERFACE(1) SENDS INS(STRING:"444") TO BOUNDED_BUFFER(1)
+++      2.000  P1: INTERFACE(1) SENDS INS(STRING:"555") TO BOUNDED_BUFFER(1)
+++      2.100  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"333")
                    FROM INTERFACE(1)
+++      2.100  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "333")
NO_CONSTRAINT
+++      2.500  P2: BOUNDED_BUFFER(1) STARTING OPERATION REM()
+++      2.600  P2: BOUNDED_BUFFER(1) SENDS *REPLY(STRING:"333")
                    TO INTERFACE(1)
+++      2.600  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"444")
                    FROM INTERFACE(1)
+++      2.600  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"555")
                    FROM INTERFACE(1)
BUFFER_EMPTY
+++      2.900  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "444")
+++      3.000  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"333")
                    FROM BOUNDED_BUFFER(1)
NO_CONSTRAINT
+++      3.300  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "555")
>>>      3.300  REAL-TIME LIMIT EXCEEDED
>>>      3.300  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND INS("666") TO BB; SEND INS("777") TO BB;RUN                           <==
+++      3.300  P1: INTERFACE(1) SENDS INS(STRING:"666") TO BOUNDED_BUFFER(1)
+++      3.300  P1: INTERFACE(1) SENDS INS(STRING:"777") TO BOUNDED_BUFFER(1)
+++      3.600  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"666")
                    FROM INTERFACE(1)
+++      3.600  P2: BOUNDED_BUFFER(1) RECEIVES INS(STRING:"777")
                    FROM INTERFACE(1)
NO_CONSTRAINT
+++      3.700  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "666")
BUFFER_FULL
+++      4.100  P2: BOUNDED_BUFFER(1) IS IDLE
>>>      4.100  SYSTEM TERMINATED
>>>      4.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX; MAILBOX BB                                                        <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"111")
(2) *REPLY(STRING:"222")
(3) *REPLY(STRING:"333")
-------------------------------------------------------------------------------

MAILBOX OF BOUNDED_BUFFER(1) :
-------------------------------------------------------------------------------
(1) INS(STRING:"777")
-------------------------------------------------------------------------------

>>>      4.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND REM TO BB REPLY TO P;RUN                                              <==
+++      4.100  P1: INTERFACE(1) SENDS REM(),REPLY TO: P TO BOUNDED_BUFFER(1)
+++      4.200  P2: BOUNDED_BUFFER(1) RECEIVES REM(),REPLY TO: P
                    FROM INTERFACE(1)
+++      4.200  P2: BOUNDED_BUFFER(1) STARTING OPERATION REM()
+++      4.300  P2: BOUNDED_BUFFER(1) SENDS *REPLY(STRING:"444")
                    TO INTERFACE(1)
NO_CONSTRAINT
+++      4.600  P2: BOUNDED_BUFFER(1) STARTING OPERATION INS(STRING:
                    "777")
```

fig. 4 -continued-

```
BUFFER_FULL
+++      5.000  P2: BOUNDED_BUFFER(1) IS IDLE
+++      5.100  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"444")
                        FROM BOUNDED_BUFFER(1)
>>>      5.200  SYSTEM TERMINATED
>>>      5.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                          <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------
(1) *REPLY(STRING:"111")
(2) *REPLY(STRING:"222")
(3) *REPLY(STRING:"333")
(4) *REPLY(STRING:"444")
-------------------------------------------------------------------------
>>>      5.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX BB                                                       <==
MAILBOX OF BOUNDED_BUFFER(1) :
-------------------------------------------------------------------------
-------------------------------------------------------------------------
>>>      5.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> DUMP BB                                                          <==

+++      5.200  RUNTIME STACK OF BOUNDED_BUFFER(1)

                 ---------------------------------
                  FACET  BUFFER_FULL
                  ENV: SCRIPT  BOUNDED_BUFFER
                  LINE:   17
                 .................................
                  SCRIPT  BOUNDED_BUFFER
                  ENV:
                  LINE:   42
                 .................................
                  MAX = 3
                  ANONYM = (0..2)
                  BUFFER(0) = "777"
                  BUFFER(1) = "555"
                  BUFFER(2) = "666"
                  INS_COUNT = 1
                  REM_COUNT = 1
                 ---------------------------------

>>>      5.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                        <==
 +++      5.200  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			*REPLY *REPLY *REPLY *REPLY
P2: BOUNDED_BUFFE	BUFFER_FU		

```
                  CSSA-SESSION-STATISTICS
                  =======================


SESSION  STARTED  AT  11:11:19.00
SESSION TERMINATED AT 11:20:26.00  ON 1981/11/27
REAL-TIME USED :   547.00 SEC.
CPU-TIME  USED :     1.88 SEC.
SIMULATION TIME USED :       5.2000 SEC.
NUMBER OF AGENTS CREATED :     1
NUMBER OF MESSAGES SENT :    15
```

fig. 4 -continued-

```
bounded_buffer = monitor is create, append, remove;

am= array[message];
rep = record[ slots:am, max:int, nonempty, nonfull: condition]

create = proc(n:int) returns (cvt);
         return (rep${slots:am$new(),
                      max:n,
                      nonempty,nonfull: condition$create()});
         end create;

append = proc(buffer:cvt, x:message) ;
         if am$size(buffer.slots) = max
           then condition$wait(buffer.nonfull);
           end;
         am$addh(buffer.slots,x);
         condition$signal(buffer.nonempty);
         end append;

remove = proc(buffer:cvt) returns (message);
         if am$size(buffer.slots) = 0
           then condition$wait(buffer.nonempty);
           end;
         x:message := am$reml(slots);
         condition$signal(buffer.nonfull);
         return (x);
         end remove;

end bounded_buffer;
```

fig. 5 : Bounded Buffer Monitor

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| BLOCKNESTING
 1  |type PROTECTED_BUFFER is                                                 |
 2  |script                                                                   |+1
 3  |  |                                                                      |
 4  |  |public : INS,REM;                                                     |
 5  |  |                                                                      |
 6  |  |type DATA_BASE is script(int: SIZE)                                   |  +2
 7  |  |               |                                                      |
 8  |  |  var array(1..SIZE) of string: FILE;                                 |
 9  |  |               |                                                      |
10  |  |  facet RD_OR_WR is                                                   |  +3
11  |  |  | public: WR,RD,RD_ACKN;                                            |
12  |  |  | var int: RCOUNT := 0;                                             |
13  |  |  |                 |                                                 |
14  |  |  | operation WR(string: DATA; int: KEY) is                           |  +4
15  |  |  | | reply;                                                          |
16  |  |  | | if KEY > 0 and KEY <= SIZE then                                 |   +5
17  |  |  | |   FILE(KEY) := DATA; endif;                                     |   -5
18  |  |  | endoperation                                                      |  -4
19  |  |  |                 |                                                 | *3
20  |  |  | operation RD(int: KEY) is                                         |  +6
21  |  |  | | RCOUNT := RCOUNT + 1;                                           |
22  |  |  | | if KEY > 0 and KEY <= SIZE then                                 |   +7
23  |  |  | |   reply (FILE(KEY)); endif;                                     |   -7
24  |  |  | endoperation                                                      |  -6
25  |  |  |                 |                                                 | *3
26  |  |  | operation RD_ACKN(int: COUNT) assert COUNT = RCOUNT is            |  +8
27  |  |  | | reply;                                                          |
28  |  |  | | RCOUNT := 0;                                                    |
29  |  |  | endoperation                                                      |  -8
30  |  |  |                 |                                                 | *3
31  |  |  | endfacet                                                          | -3
32  |  |  |                 |                                                 |*2
33  |  |  | initial RD_OR_WR                                                  |
34  |  |  |                 |                                                 |
35  |  |  endscript;                                                          |-2
36  |  |                                                                      |*1
37  |  |(* back on top level of script PROTECTED_BUFFER   *)                  |
38  |  |                                                                      |
39  |  |var DATA_BASE : DB;                                                   |
40  |  |var int : REM_COUNT,INS_COUNT;                                        |
41  |  |functionhead MOD(int : P1,P2) returns int external;                   |
42  |  |var int : MAX;                                                        |
43  |  |                                                                      |
44  |  |facethead BUFFER_EMPTY;                                               |
45  |  |facethead BUFFER_FULL;                                                |
46  |  |facethead NO_CONSTRAINT;                                              |
47  |  |                                                                      |
48  |  |facet CREATION is                                                     |  +9
49  |  | public : CREATE;                                                     |
50  |  |                                                                      |
51  |  | operation CREATE(->MAX) is                                           |  +10
52  |  | | if MAX > 0 then                                                    |   +11
53  |  | | | DB := new DATA_BASE(MAX);                                        |
54  |  | | | replace by BUFFER_EMPTY;                                         |
55  |  | | else print(" Buffer size must be greater than 0 ");                |
56  |  | | endif;                                                             |   -11
```

fig. 6 : Protected Buffer Script

```
B M S - C S S A - C O M P I L E R                                    1981/12/07   16:3

    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
 57 | endoperation                                                            |  -10
 58 |endfacet                                                                 |  -9
 59 |                                                                         | *1
 60 |facet BUFFER_EMPTY is                                                    |  +12
 61 | include : INS;                                                          |
 62 |endfacet                                                                 |  -12
 63 |                                                                         | *1
 64 |facet NO_CONSTRAINT is                                                   |  +13
 65 | include : INS,REM;                                                      |
 66 |endfacet                                                                 |  -13
 67 |                                                                         | *1
 68 |facet BUFFER_FULL is                                                     |  +14
 69 | include : REM;                                                          |
 70 |endfacet                                                                 |  -14
 71 |                                                                         | *1
 72 |operation INS(string : M) is                                             |  +15
 73 |  port : ACCEPTED;                                                       |
 74 |  oper : WR;                                                             |
 75 |  ---------------                                                        |
 76 |  send WR(M,INS_COUNT + 1) to DB reply to ACCEPTED;                      |
 77 |  INS_COUNT := MOD(INS_COUNT + 1, MAX);                                  |
 78 |  wait ACCEPTED;                                                         |
 79 |  if INS_COUNT = REM_COUNT then replace by BUFFER_FULL;                  |  +16
 80 |  else replace by NO_CONSTRAINT;                                         |
 81 |  endif;                                                                 |  -16
 82 |endoperation                                                             |  -15
 83 |                                                                         | *1
 84 |operation REM is                                                         |  +17
 85 |  port : ACCEPTED;                                                       |
 86 |  var string : M;                                                        |
 87 |  oper : RD;                                                             |
 88 |  ---------------                                                        |
 89 |  send RD(REM_COUNT + 1) to DB reply to ACCEPTED;                        |
 90 |  REM_COUNT := MOD(REM_COUNT + 1, MAX);                                  |
 91 |  wait ACCEPTED(M);                                                      |
 92 |  reply (M);                                                             |
 93 |  if REM_COUNT = INS_COUNT then replace by BUFFER_EMPTY;                 |  +18
 94 |  else replace by NO_CONSTRAINT;                                         |
 95 |  endif;                                                                 |  -18
 96 |endoperation                                                             |  -17
 97 |                                                                         | *1
 98 |initial CREATION                                                         |
 99 |                                                                         |
100 |endscript                                                                |  -1
```

fig. 6 -continued-

```
          PROGRAM GENERATED ON 1981/12/07 AT 16:36:49.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1981/12/07 AT 16:54:19.00
          ====================================================

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> DISPLAY                                                              <==

| IDENTIFIER          | TYPE          | VALUE               |
|---------------------|---------------|---------------------|
| DATA_BASE           | SCRIPT        | DATA_BASE           |
| PROTECTED_BUFFER    | SCRIPT        | PROTECTED_BUFFER    |

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : PB := NEW PROTECTED_BUFFER                               <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OPER : CREATE,INS,REM; PORT : P                                      <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE; SEND CREATE(3) TO PB;RUN                                    <==
+++      0.000  P1: INTERFACE(1) SENDS CREATE(INT:3) TO PROTECTED_BUFFER(1)
+++      0.000  P2: PROTECTED_BUFFER(1) IS IDLE
+++      0.100  P2: PROTECTED_BUFFER(1) RECEIVES CREATE(INT:3)
                        FROM INTERFACE(1)
+++      0.100  P2: PROTECTED_BUFFER(1) STARTING OPERATION CREATE(INT:
                        3)
+++      0.300  P2: PROTECTED_BUFFER(1) CREATES DATA_BASE(1)(INT:
                        3)
+++      0.300  P3: DATA_BASE(1) IS IDLE
BUFFER_EMPTY
+++      0.400  P2: PROTECTED_BUFFER(1) IS IDLE
>>>      0.400  SYSTEM TERMINATED
>>>      0.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND REM TO PB REPLY TO P; RUN                                       <==
+++      0.400  P1: INTERFACE(1) SENDS REM(),REPLY TO: P TO PROTECTED_BUFFER(1)
+++      0.500  P2: PROTECTED_BUFFER(1) RECEIVES REM(),REPLY TO: P
                        FROM INTERFACE(1)
+++      0.500  P2: PROTECTED_BUFFER(1) IS IDLE
>>>      0.500  SYSTEM TERMINATED
>>>      0.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND INS("A") TO PB;RUN                                              <==
+++      0.500  P1: INTERFACE(1) SENDS INS(STRING:"A") TO PROTECTED_BUFFER(1)
+++      0.600  P2: PROTECTED_BUFFER(1) RECEIVES INS(STRING:"A")
                        FROM INTERFACE(1)
+++      0.600  P2: PROTECTED_BUFFER(1) STARTING OPERATION INS(STRING:
                        "A")
+++      0.700  P2: PROTECTED_BUFFER(1) SENDS WR(STRING:"A",INT:
                        1),REPLY TO: ACCEPTED TO DATA_BASE(1)
+++      0.900  P2: PROTECTED_BUFFER(1) IS IDLE
ACCEPTED FROM PROTECTED_BUFFER(1)
+++      1.600  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                        "A",INT:1)
+++      1.700  P3: DATA_BASE(1) SENDS *REPLY() TO PROTECTED_BUFFER(1)
+++      1.900  P3: DATA_BASE(1) IS IDLE
+++      2.600  P2: PROTECTED_BUFFER(1) RECEIVES *REPLY() FROM DATA_BASE(1)
NO_CONSTRAINT
+++      2.800  P2: PROTECTED_BUFFER(1) STARTING OPERATION REM()
+++      2.900  P2: PROTECTED_BUFFER(1) SENDS RD(INT:1),REPLY TO: ACCEPTED
                        TO DATA_BASE(1)
+++      3.100  P2: PROTECTED_BUFFER(1) IS IDLE
+++      3.600  P3: DATA_BASE(1) RECEIVES RD(INT:1),REPLY TO: ACCEPTED
                        FROM PROTECTED_BUFFER(1)
+++      3.600  P3: DATA_BASE(1) STARTING OPERATION RD(INT:1)
+++      3.900  P3: DATA_BASE(1) SENDS *REPLY(STRING:"A") TO PROTECTED_BUFFER(1)
+++      3.900  P3: DATA_BASE(1) IS IDLE
+++      4.600  P2: PROTECTED_BUFFER(1) RECEIVES *REPLY(STRING:
                        "A") FROM DATA_BASE(1)
+++      4.700  P2: PROTECTED_BUFFER(1) SENDS *REPLY(STRING:"A")
                        TO INTERFACE(1)
BUFFER_EMPTY
```

fig. 7 : Protected Buffer Execution Protocol

```
+++      4.900  P2: PROTECTED_BUFFER(1) IS IDLE
PROTECTED_BUFFER(1)
>>>      5.600  SYSTEM TERMINATED
>>>      5.600  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX; SEND INS("B") TO PB; SEND INS("C") TO PB;RUN                    <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
-------------------------------------------------------------------------------

+++      5.600  P1: INTERFACE(1) SENDS INS(STRING:"B") TO PROTECTED_BUFFER(1)
+++      5.600  P1: INTERFACE(1) SENDS INS(STRING:"C") TO PROTECTED_BUFFER(1)
+++      5.700  P2: PROTECTED_BUFFER(1) RECEIVES INS(STRING:"B")
                          FROM INTERFACE(1)
+++      5.700  P2: PROTECTED_BUFFER(1) STARTING OPERATION INS(STRING:
                          "B")
+++      5.800  P2: PROTECTED_BUFFER(1) SENDS WR(STRING:"B",INT:
                          2),REPLY TO: ACCEPTED TO DATA_BASE(1)
+++      5.800  P2: PROTECTED_BUFFER(1) RECEIVES INS(STRING:"C")
                          FROM INTERFACE(1)
+++      6.000  P2: PROTECTED_BUFFER(1) IS IDLE
ACCEPTED FROM PROTECTED_BUFFER(1)
+++      6.700  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                          "B",INT:2)
+++      6.800  P3: DATA_BASE(1) SENDS *REPLY() TO PROTECTED_BUFFER(1)
+++      7.000  P3: DATA_BASE(1) IS IDLE
+++      7.700  P2: PROTECTED_BUFFER(1) RECEIVES *REPLY() FROM DATA_BASE(1)
NO_CONSTRAINT
+++      7.900  P2: PROTECTED_BUFFER(1) STARTING OPERATION INS(STRING:
                          "C")
+++      8.000  P2: PROTECTED_BUFFER(1) SENDS WR(STRING:"C",INT:
                          3),REPLY TO: ACCEPTED TO DATA_BASE(1)
+++      8.200  P2: PROTECTED_BUFFER(1) IS IDLE
ACCEPTED FROM PROTECTED_BUFFER(1)
+++      8.700  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                          "C",INT:3)
+++      8.800  P3: DATA_BASE(1) SENDS *REPLY() TO PROTECTED_BUFFER(1)
+++      9.000  P3: DATA_BASE(1) IS IDLE
+++      9.700  P2: PROTECTED_BUFFER(1) RECEIVES *REPLY() FROM DATA_BASE(1)
NO_CONSTRAINT
+++      9.900  P2: PROTECTED_BUFFER(1) IS IDLE
>>>     10.000  SYSTEM TERMINATED
>>>     10.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND INS("D") TO PB; SEND INS("E") TO PB;RUN                             <==
+++     10.000  P1: INTERFACE(1) SENDS INS(STRING:"D") TO PROTECTED_BUFFER(1)
+++     10.000  P1: INTERFACE(1) SENDS INS(STRING:"E") TO PROTECTED_BUFFER(1)
+++     10.100  P2: PROTECTED_BUFFER(1) RECEIVES INS(STRING:"D")
                          FROM INTERFACE(1)
+++     10.100  P2: PROTECTED_BUFFER(1) STARTING OPERATION INS(STRING:
                          "D")
+++     10.200  P2: PROTECTED_BUFFER(1) SENDS WR(STRING:"D",INT:
                          1),REPLY TO: ACCEPTED TO DATA_BASE(1)
+++     10.200  P2: PROTECTED_BUFFER(1) RECEIVES INS(STRING:"E")
                          FROM INTERFACE(1)
+++     10.400  P2: PROTECTED_BUFFER(1) IS IDLE
ACCEPTED FROM PROTECTED_BUFFER(1)
+++     11.100  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                          "D",INT:1)
+++     11.200  P3: DATA_BASE(1) SENDS *REPLY() TO PROTECTED_BUFFER(1)
+++     11.400  P3: DATA_BASE(1) IS IDLE
+++     12.100  P2: PROTECTED_BUFFER(1) RECEIVES *REPLY() FROM DATA_BASE(1)
BUFFER_FULL
+++     12.300  P2: PROTECTED_BUFFER(1) IS IDLE
>>>     12.300  SYSTEM TERMINATED
>>>     12.300  INTERFACE(1) : ENTER CSSA COMMAND -
==> NOOBSERVE; SEND REM TO PB REPLY TO P; SEND REM TO PB REPLY TO P; RUN     <==

>>>     21.400  SYSTEM TERMINATED
>>>     21.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX;                                                                <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
(2) *REPLY(STRING:"B")
(3) *REPLY(STRING:"C")
(4) *REPLY(STRING:"D")
```

fig. 7 -continued-

```
>>>    21.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND REM TO PB REPLY TO P; SEND REM TO PB REPLY TO P;RUN                    <==
>>>    24.500  SYSTEM TERMINATED
>>>    24.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                                    <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
(2) *REPLY(STRING:"B")
(3) *REPLY(STRING:"C")
(4) *REPLY(STRING:"D")
(5) *REPLY(STRING:"E")
-------------------------------------------------------------------------------

>>>    24.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> DUMP DATA_BASE(1); DUMP PB                                                 <==

+++    24.500  RUNTIME STACK OF DATA_BASE(1)

                +-------------------------------+
                | FACET  RD_OR_WR               |
                | ENV: SCRIPT  DATA_BASE        |
                | LINE:   11                    |
                |...............................|
                | RCOUNT = 5                    |
                +-------------------------------+
                | SCRIPT  DATA_BASE             |
                | ENV:                          |
                | LINE:   35                    |
                |...............................|
                | SIZE = 3                      |
                | ANONYM = (1..3)               |
                | FILE(1) = "D"                 |
                | FILE(2) = "E"                 |
                | FILE(3) = "C"                 |
                +-------------------------------+

+++    24.500  RUNTIME STACK OF PROTECTED_BUFFER(1)

                +-------------------------------+
                | FACET  BUFFER_EMPTY           |
                | ENV: SCRIPT  PROTECTED_BUFFER |
                | LINE:   60                    |
                |...............................|
                +-------------------------------+
                | SCRIPT  PROTECTED_BUFFER      |
                | ENV:                          |
                | LINE.  100                    |
                |...............................|
                | DATA_BASE = DATA_BASE         |
                | DB = DATA_BASE(1)             |
                | REM_COUNT = 2                 |
                | INS_COUNT = 2                 |
                | MAX = 3                       |
                +-------------------------------+

>>>    24.500  INTERFACE(1) : ENTER CSSA COMMAND -
>>>    24.600  SYSTEM TERMINATED
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------------
(1) *REPLY(STRING:"A")
(2) *REPLY(STRING:"B")
(3) *REPLY(STRING:"C")
(4) *REPLY(STRING:"D")
(5) *REPLY(STRING:"E")
-------------------------------------------------------------------------------
```

fig. 7 -continued-

```
>>>    24.600  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND INS("F") TO PB; RUN; MAILBOX                                    <==
>>>    29.700  SYSTEM TERMINATED
MAILBOX OF INTERFACE(1) :
```

```
(1) *REPLY(STRING:"A")
(2) *REPLY(STRING:"B")
(3) *REPLY(STRING:"C")
(4) *REPLY(STRING:"D")
(5) *REPLY(STRING:"E")
(6) *REPLY(STRING:"F")
```

```
>>>    29.700  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                            <==
 +++    29.700  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			*REPLY *REPLY *REPLY *REPLY *REPLY *REPLY
P2: PROTECTED_BUF	BUFFER_EM		
P3: DATA_BASE(1)	RD_OR_WR		

```
                CSSA-SESSION-STATISTICS
                =======================

SESSION  STARTED  AT  16:52:59.00
SESSION TERMINATED AT 17:06:42.00  ON 1981/12/07
REAL-TIME USED :  823.00 SEC.
CPU-TIME  USED :    3.06 SEC.
SIMULATION TIME USED :     29.7000 SEC.
NUMBER OF AGENTS CREATED :    2
NUMBER OF MESSAGES SENT :   43
```

fig. 7 -continued-

```
protected_buffer = monitor is create, append, remove;

rep = record[ slots:buffer, nonempty, nonfull: condition ]

create = proc() returns (cvt);
        return (rep${slots:buffer$create(),
                nonempty,nonfull: condition$create()});
        end create;

append = proc(pb:cvt, x:message) ;
        if buffer$full(pb.slots) then condition$wait(pb.nonfull) end;
        buffer$append(pb.slots, x);
        condition$signal(pb.nonempty);
        end append;

remove = proc(pb:cvt) returns (message);
        if buffer$empty(pb.slots) then condition$wait(pb.nonempty) end;
        x:message := buffer$remove(pb.slots);
        condition$signal(pb.nonfull);
        return (x);
        end remove;

end bounded_buffer;
```

fig. 8 : Protected Buffer Monitor

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| BLOCKNESTING
 1  |type RDER_PRIO is script                                                 |+1
 2  |                 |                                                       |
 3  |  type DATA_BASE is script(int: SIZE)                                    | +2
 4  |                 | |                                                     | |
 5  |    var array(1..SIZE) of string: FILE;                                  | |
 6  |                 | |                                                     | |
 7  |    facet RD_OR_WR is                                                    | +3
 8  |    | public: WR,RD,RD_ACKN;                                             | |
 9  |    | var int: RCOUNT := 0;                                              | |
10  |    |                 | |                                               | |
11  |    | operation WR(string: DATA; int: KEY) is                           | +4
12  |    | | reply;                                                          | |
13  |    | | if KEY > 0 and KEY <= SIZE then                                 | +5
14  |    | |   FILE(KEY) := DATA; endif;                                     | -5
15  |    | endoperation                                                      |-4
16  |    |                 | |                                               |*3
17  |    | operation RD(int: KEY) is                                         | +6
18  |    | | RCOUNT := RCOUNT + 1;                                           | |
19  |    | | if KEY > 0 and KEY <= SIZE then                                 | +7
20  |    | |   reply (FILE(KEY)); endif;                                     | -7
21  |    | endoperation                                                      |-6
22  |    |                 | |                                               |*3
23  |    | operation RD_ACKN(int: COUNT) assert COUNT = RCOUNT is            | +8
24  |    | | reply;                                                          | |
25  |    | | RCOUNT := 0;                                                    | |
26  |    | endoperation                                                      | -8
27  |    |                 | |                                               |*3
28  |    | endfacet                                                          |-3
29  |    |                 | |                                               |*2
30  |    | initial RD_OR_WR                                                  | |
31  |    |                 | |                                               | |
32  |  endscript;                                                            |-2
33  |                      |                                                 |*1
34  |                      |                                                 | |
35  |var DATA_BASE: DB;                                                      | |
36  |var int: RDER_COUNT := 0;                                               | |
37  |var bool: FIRST_OPER := true;                                           | |
38  |                      |                                                 | |
39  |facethead WRRS;                                                         | |
40  |                      |                                                 | |
41  |facet CREATION is                                                       | +9
42  | public: CREATE;                                                        | |
43  |                      |                                                 | |
44  | operation CREATE(int: SIZE) is                                         | +10
45  | | DB := new DATA_BASE(SIZE);                                           | |
46  | | replace by WRRS;                                                     | |
47  | endoperation                                                           | -10
48  |                      |                                                 |*9
49  |endfacet                                                                |-9
50  |                      |                                                 |*1
51  |facet RDERS is                                                          | +11
52  | public: RD;                                                            | |
53  |                      |                                                 | |
54  | operation RD(int: KEY) is                                              | +12
55  | | send RD(KEY) to DB inherit;                                          | |
56  | | RDER_COUNT := RDER_COUNT + 1;                                        | |
```

fig. 9 : Readers Priority Script

```
B M S - C S S A - C O M P I L E R                                   1981/12/07   15:42

    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
 57 |  endoperation                                                           |-12
 58 |                    |                                                   |*11
 59 |  operation idle is replace by WRRS;                                     |+13
 60 |  endoperation                                                           |-13
 61 |                    |                                                   |*11
 62 |endfacet                                                                 |-11
 63 |                    |                                                  |*1
 64 |facet WRRS is                                                            |+14
 65 |  public: WR;                                                            |
 66 |  var bool: FIRST_WR := true;                                            |
 67 |                    |                                                    |
 68 |  operation WR(string: DATA; int: KEY)  assert FIRST_WR  is              |+15
 69 |  | oper: RD_ACKN;                                                       |
 70 |  | port: R_ACKN,W_ACKN;                                                 |
 71 |  | --------------------                                                 |
 72 |  | if FIRST_OPER then FIRST_OPER := false; endif;                       |+16-16
 73 |  | FIRST_WR := false;                                                   |*15
 74 |  |                 |                                                    |
 75 |  | if RDER_COUNT > 0 then                                               |+17
 76 |  | | send RD_ACKN(RDER_COUNT) to DB reply to R_ACKN;                    |
 77 |  | | wait R_ACKN ;                                                      |
 78 |  | | RDER_COUNT := 0;                                                   |
 79 |  | endif;                                                               |-17
 80 |  | send WR(DATA,KEY) to DB reply to W_ACKN;                             |*15
 81 |  | wait W_ACKN;                                                         |
 82 |  endoperation                                                           |-15
 83 |                    |                                                  |*14
 84 |  operation idle  is                                                     |+18
 85 |  | if not FIRST_OPER then replace by RDERS; endif;                      |+19-19
 86 |  endoperation                                                           |-18
 87 |                    |                                                  |*14
 88 |endfacet                                                                 |-14
 89 |                    |                                                 |*1
 90 |initial CREATION                                                         |
 91 |                    |                                                  |
 92 |endscript                                                               |-1
```

fig. 9 -continued-

```
PROGRAM GENERATED ON 1981/12/07 AT 15:49:41.00
BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

PROTOCOL OF CSSA SESSION ON 1981/12/07 AT 16:02:06.00
=====================================================

>>>     0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;DISPLAY                                                          <==
```

IDENTIFIER	TYPE	VALUE
DATA_BASE	SCRIPT	DATA_BASE
RDER_PRIO	SCRIPT	RDER_PRIO

```
>>>     0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : PRT := NEW RDER_PRIO                                  <==
>>>     0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OPER : RD,WR,CREATE; PORT : P;                                    <==
>>>     0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE; SEND CREATE(1) TO PRT; SEND WR("A",1) TO PRT;RUN         <==
+++     0.000  P1: INTERFACE(1) SENDS CREATE(INT:1) TO RDER_PRIO(1)
+++     0.000  P1: INTERFACE(1) SENDS WR(STRING:"A",INT:1) TO RDER_PRIO(1)
+++     0.000  P2: RDER_PRIO(1) IS IDLE
+++     0.100  P2: RDER_PRIO(1) RECEIVES CREATE(INT:1) FROM INTERFACE(1)
+++     0.100  P2: RDER_PRIO(1) STARTING OPERATION CREATE(INT:
                            1)
+++     0.200  P2: RDER_PRIO(1) CREATES DATA_BASE(1)(INT:1)
+++     0.200  P2: RDER_PRIO(1) RECEIVES WR(STRING:"A",INT:1)
                              FROM INTERFACE(1)
+++     0.200  P3: DATA_BASE(1) IS IDLE
+++     0.300  P2: RDER_PRIO(1) PERFORMS FACETTING : CREATION --> WRRS
+++     0.300  P2: RDER_PRIO(1) STARTING OPERATION WR(STRING:
                            "A",INT:1)
+++     0.800  P2: RDER_PRIO(1) SENDS WR(STRING:"A",INT:1),REPLY TO: W_ACKN
                              TO DATA_BASE(1)
+++     0.900  P2: RDER_PRIO(1) IS IDLE
+++     1.100  P3: DATA_BASE(1) RECEIVES WR(STRING:"A",INT:1),REPLY TO: W_ACKN
                              FROM RDER_PRIO(1)
+++     1.100  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                            "A",INT:1)
+++     1.200  P3: DATA_BASE(1) SENDS *REPLY() TO RDER_PRIO(1)
+++     1.400  P3: DATA_BASE(1) IS IDLE
+++     2.100  P2: RDER_PRIO(1) RECEIVES *REPLY() FROM DATA_BASE(1)
+++     2.100  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     2.300  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     2.300  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     2.400  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     2.400  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     2.600  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     2.600  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     2.700  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     2.700  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     2.900  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     2.900  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.000  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     3.000  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.200  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     3.200  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.300  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     3.300  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.500  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     3.500  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.600  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     3.600  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     3.800  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     3.800  P2: RDER_PRIO(1) STARTING OPERATION IDLE
>>>     3.800  REAL-TIME LIMIT EXCEEDED
>>>     3.800  INTERFACE(1) : ENTER CSSA COMMAND -
```

fig. 10 : Readers Priority Execution Protocol

```
==> SEND RD(1) TO PRT REPLY TO P; SEND WR("B",1) TO PRT;                    <==
+++      3.800  P1: INTERFACE(1) SENDS RD(INT:1),REPLY TO: P TO RDER_PRIO(1)
+++      3.800  P1: INTERFACE(1) SENDS WR(STRING:"B",INT:1) TO RDER_PRIO(1)
>>>      3.800  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND RD(1) TO PRT REPLY TO P; SEND RD(1) TO PRT REPLY TO P;             <==
+++      3.800  P1: INTERFACE(1) SENDS RD(INT:1),REPLY TO: P TO RDER_PRIO(1)
+++      3.800  P1: INTERFACE(1) SENDS RD(INT:1),REPLY TO: P TO RDER_PRIO(1)
>>>      3.800  INTERFACE(1) : ENTER CSSA COMMAND -
==> RUN;                                                                    <==
+++      3.900  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      3.900  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      4.100  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++      4.100  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      4.100  P2: RDER_PRIO(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM INTERFACE(1)
+++      4.100  P2: RDER_PRIO(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM INTERFACE(1)
+++      4.100  P2: RDER_PRIO(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM INTERFACE(1)
+++      4.200  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      4.200  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      4.400  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++      4.400  P2: RDER_PRIO(1) STARTING OPERATION RD(INT:1)
+++      4.500  P2: RDER_PRIO(1) SENDS RD(INT:1),REPLY TO: P TO DATA_BASE(1)
+++      4.500  P2: RDER_PRIO(1) RECEIVES WR(STRING:"B",INT:1)
                          FROM INTERFACE(1)
+++      4.600  P2: RDER_PRIO(1) STARTING OPERATION RD(INT:1)
+++      4.700  P2: RDER_PRIO(1) SENDS RD(INT:1),REPLY TO: P TO DATA_BASE(1)
+++      4.800  P2: RDER_PRIO(1) STARTING OPERATION RD(INT:1)
+++      4.900  P2: RDER_PRIO(1) SENDS RD(INT:1),REPLY TO: P TO DATA_BASE(1)
+++      4.900  P3: DATA_BASE(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM RDER_PRIO(1)
+++      4.900  P3: DATA_BASE(1) STARTING OPERATION RD(INT:1)
+++      5.000  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      5.100  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      5.100  P2: RDER_PRIO(1) STARTING OPERATION WR(STRING:
                         "B",INT:1)
+++      5.200  P3: DATA_BASE(1) SENDS *REPLY(STRING:"A") TO INTERFACE(1)
+++      5.200  P3: DATA_BASE(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM RDER_PRIO(1)
+++      5.200  P3: DATA_BASE(1) STARTING OPERATION RD(INT:1)
+++      5.500  P2: RDER_PRIO(1) SENDS RD_ACKN(INT:3),REPLY TO: R_ACKN
                          TO DATA_BASE(1)
+++      5.500  P3: DATA_BASE(1) SENDS *REPLY(STRING:"A") TO INTERFACE(1)
+++      5.500  P3: DATA_BASE(1) IS IDLE
+++      5.600  P2: RDER_PRIO(1) IS IDLE
+++      5.800  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"A") FROM DATA_BASE(1)
+++      5.800  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"A") FROM DATA_BASE(1)
+++      5.900  P3: DATA_BASE(1) RECEIVES RD_ACKN(INT:3),REPLY TO: R_ACKN
                          FROM RDER_PRIO(1)
+++      5.900  P3: DATA_BASE(1) IS IDLE
+++      6.000  P3: DATA_BASE(1) RECEIVES RD(INT:1),REPLY TO: P
                          FROM RDER_PRIO(1)
+++      6.000  P3: DATA_BASE(1) STARTING OPERATION RD(INT:1)
+++      6.300  P3: DATA_BASE(1) SENDS *REPLY(STRING:"A") TO INTERFACE(1)
+++      6.300  P3: DATA_BASE(1) STARTING OPERATION RD_ACKN(INT:
                         3)
+++      6.400  P3: DATA_BASE(1) SENDS *REPLY() TO RDER_PRIO(1)
+++      6.500  P3: DATA_BASE(1) IS IDLE
+++      6.800  P1: INTERFACE(1) RECEIVES *REPLY(STRING:"A") FROM DATA_BASE(1)
+++      6.900  P2: RDER_PRIO(1) RECEIVES *REPLY() FROM DATA_BASE(1)
+++      7.100  P2: RDER_PRIO(1) SENDS WR(STRING:"B",INT:1),REPLY TO: W_ACKN
                          TO DATA_BASE(1)
+++      7.200  P2: RDER_PRIO(1) IS IDLE
+++      7.900  P3: DATA_BASE(1) RECEIVES WR(STRING:"B",INT:1),REPLY TO: W_ACKN
                          FROM RDER_PRIO(1)
+++      7.900  P3: DATA_BASE(1) STARTING OPERATION WR(STRING:
                         "B",INT:1)
+++      8.000  P3: DATA_BASE(1) SENDS *REPLY() TO RDER_PRIO(1)
+++      8.200  P3: DATA_BASE(1) IS IDLE
+++      8.900  P2: RDER_PRIO(1) RECEIVES *REPLY() FROM DATA_BASE(1)
+++      8.900  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      9.100  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++      9.100  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      9.200  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      9.200  P2: RDER_PRIO(1) STARTING OPERATION IDLE
>>>      9.300  REAL-TIME LIMIT EXCEEDED
>>>      9.300  INTERFACE(1) : ENTER CSSA COMMAND -
```

fig. 10 -continued-

```
+++      9.400  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++      9.400  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      9.500  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      9.500  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      9.700  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++      9.700  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++      9.800  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++      9.800  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.000  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     10.000  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.100  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     10.100  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.300  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     10.300  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.400  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     10.400  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.600  P2: RDER_PRIO(1) PERFORMS FACETTING : WRRS --> RDERS
+++     10.600  P2: RDER_PRIO(1) STARTING OPERATION IDLE
+++     10.700  P2: RDER_PRIO(1) PERFORMS FACETTING : RDERS --> WRRS
+++     10.700  P2: RDER_PRIO(1) STARTING OPERATION IDLE
>>>     10.700  REAL-TIME LIMIT EXCEEDED
>>>     10.700  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                          <==
 +++     10.700  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			*REPLY *REPLY *REPLY
P2: RDER_PRIO(1) *	WRRS	IDLE	
P3: DATA_BASE(1)	RD_OR_WR		

```
                  CSSA-SESSION-STATISTICS
                  =======================

SESSION  STARTED  AT  15:56:03.00
SESSION TERMINATED AT 16:17:13.00  ON 1981/12/07
REAL-TIME USED : 1270.00 SEC.
CPU-TIME  USED :    2.07 SEC.
SIMULATION TIME USED :     10.7000 SEC.
NUMBER OF AGENTS CREATED :    2
NUMBER OF MESSAGES SENT :   18
```

fig. 10 -continued-

```
readers_priority = monitor is create,
                              startread,
                              endread,
                              startwrite,
                              endwrite;

rep = record[readercount: int,
             busy:boolean,
             readers, writers:condition];

create = proc() returns (cvt);
         return(rep${readercount: 0,
                busy:false,
                readers,writers:condition$create()});
         end create;

startread = proc(m:cvt);
         if m.busy then condition$wait(m.readers) end;
         m.readercount:= m.readercount + 1;
         condition$signal(m.readers);
         end startread;

endread = proc(m:cvt);
         m.readercount:= m.readercount - 1;
         if m.readercount:=0
           then condition$signal(m.writers)
           end;
         end endread;

startwrite = proc(m:cvt);
         if m.readercount > 0 | m.busy
            then condition$wait(m.writers)
            end;
         m.busy:=true;
         end startwrite;

endwrite = proc(m:cvt);
         m.busy:=false;
         if condition$queue(m.readers)
            then condition$signal(m.readers)
            else condition$signal(m.writers)
            end;
         end endwrite;

end readers_priority;
```

fig. 11a : Readers Priority Monitor

```
protected_data_base = cluster is create,read,write;

rep = record[m: readers_priority,d: data_base]

create = proc()returns(cvt);
        return (rep${m: readers_priority$create(),
                d: data_base$create()});
        end create;

read = proc(pdb: cvt) returns(data);
        readers_priority$startread(pdb.m);
        x:data :=data_base$read(pdb.d);
        readers_priority$endread(pdb.m);
        return (x);
        end read;

write = proc(pdb: cvt, x:data);
        readers_priority$startwrite(pdb.m);
        data_base$write((pdb.d, x);
        readers_priority$endwrite(pdb.m);
        end write;

end protected_data_base;
```

fig. 11b : Readers Priority Protected Resource Module

```
first_come_first_serve = monitor is create, startread, endread, startwrite, endwrite;

rep = record[ busy: boolean,
              readercount: integer,
              users, writer: condition]

create = proc() returns (cvt);
         return(rep${busy:false, readercount:0, users, writer: condition$create()};
         end create;

startread = proc(m: cvt)
         if m.busy | condition$queue(m.writer) | condition$queue(m.users)
           then condition$wait(m.users);
           end;
         m.readercount:=m.readercount + 1;
         condition$signal(m.users); %start all readers
         end startread;

endread = proc(m:cvt);
         m.readercount := m.readercount - 1;
         if m.readercount = 0
           then if condition$queue(m.writer)
                   then condition$signal(m.writer)
                   else condition$signal(m.users)
                   end;
           end;
         %anyone on the writers queue has been waiting longer than those on users queue
         end endread;

startwrite = proc(m:cvt);
         if m.readercount > 0 | m.busy
           then condition$wait(m.users);
           end;
         if m.readercount > 0
           then condition$wait(m.writer);
           end;
         m.busy := true;
         end startwrite;

endwrite = proc(m:cvt);
         m.busy:=false;
         condition$signal(m.users);
         end endwrite;

end first_come_first_serve;
```

fig. 12 : First Come First Serve Monitor

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
 1  |type ALARM_CLOCK is                                                      |
 2  |script                                                                   |+1
 3  |  var int: NOW := 0;                                                     ||
 4  |                                                                         ||
 5  |  facet REGISTRY is                                                      |+2
 6  |  | public: TICK,WAKE_ME;                                                ||
 7  |  |                                                                      ||
 8  |  | facet RESUME is                                                      |+3
 9  |  | | private: WAKE_UP;                                                  ||
10  |  | |                                                                    ||
11  |  | | operation WAKE_UP(int: TIME) assert TIME=NOW is                    |+4
12  |  | | | reply;                                                           ||
13  |  | | endoperation                                                       |-4
14  |  | |                                                                    |*3
15  |  | | operation idle is leave;                                           |+5
16  |  | | endoperation                                                       |-5
17  |  | endfacet                                                             |-3
18  |  |                                                                      |*2
19  |  |                                                                      ||
20  |  | operation WAKE_ME(int: DELAY_TIME) is                                |+6
21  |  | | oper: WAKE_UP;                                                     ||
22  |  | | ----------------------------                                       ||
23  |  | | if DELAY_TIME <= 0 then reply;                                     |+7
24  |  | | else send WAKE_UP(DELAY_TIME + NOW) to self inherit; endif;        |-7
25  |  | endoperation                                                         |-6
26  |  |                                                                      |*2
27  |  | operation TICK is                                                    |+8
28  |  | | NOW := NOW + 1;                                                    ||
29  |  | | setup RESUME;                                                      ||
30  |  | endoperation                                                         |-8
31  |  |                                                                      |*2
32  |  endfacet                                                              |-2
33  |                                                                        |*1
34  |  initial REGISTRY                                                      ||
35  |                                                                        ||
36  |endscript                                                               |-1
```

fig. 13 : Alarmclock Script (Wrong Solution)

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
 1  |type ALARM_CLOCK is                                                      |+1
 2  |script                                                                   |
 3  |                                                                         |
 4  |facet REGISTRY is                                                        | +2
 5  |  public : TICK,WAKE_ME;                                                 | |
 6  |                                                                         | |
 7  |  type CALL_ELEM is record int : WAKE_UP_TIME;                           | |+3
 8  |                       |  --> int : NO_OF_SLEEPERS;                      | ||
 9  |                       endrecord;                                        | |-3
10  |  relation CALL_LIST of CALL_ELEM;                                       | *2
11  |  var int : NOW := 0;                                                    | |
12  |                                                                         | |
13  |  facet RESUME is                                                        | |+4
14  |    private : WAKE_UP;                                                   | ||
15  |                                                                         | ||
16  |    operation WAKE_UP(int : TIME) assert TIME = NOW is                   | ||+5
17  |      reply;                                                             | |||
18  |      find CALL_LIST(TIME) -> CE do                                      | |||+6
19  |        if CE.NO_OF_SLEEPERS = 1 then                                    | ||||+7
20  |          delete CE in CALL_LIST;                                        | ||||
21  |          leave;                                                         | ||||
22  |        else CE.NO_OF_SLEEPERS := CE.NO_OF_SLEEPERS - 1;                 | ||||
23  |        endif;                                                           | ||||-7
24  |      endfind;                                                           | |||-6
25  |    endoperation                                                         | ||-5
26  |  endfacet                                                               | |-4
27  |                                                                         | *2
28  |  operation WAKE_ME(int : DELAY_TIME) is                                 | |+8
29  |    oper : WAKE_UP;                                                      | ||
30  |    var CALL_ELEM : CE;                                                  | ||
31  |    ------------------                                                  | ||
32  |    if DELAY_TIME <= 0 then reply;                                       | ||+9
33  |    else find CALL_LIST(DELAY_TIME + NOW) -> CE1 do                      | |||+10
34  |         | CE1.NO_OF_SLEEPERS := CE1.NO_OF_SLEEPERS + 1;                 | ||||
35  |         otherwise do                                                    | ||||
36  |           | CE.WAKE_UP_TIME := DELAY_TIME + NOW;                        | ||||
37  |           | CE.NO_OF_SLEEPERS := 1;                                     | ||||
38  |           | insert CE into CALL_LIST;                                   | ||||
39  |         endfind;                                                        | |||-10
40  |         send WAKE_UP(DELAY_TIME + NOW) to self inherit;                 | ||*9
41  |    endif;                                                               | ||-9
42  |  endoperation                                                           | |-8
43  |                                                                         | *2
44  |  operation TICK is                                                      | |+11
45  |    NOW := NOW + 1;                                                      | ||
46  |    find CALL_LIST(NOW) -> CE do                                         | |||+12
47  |    | setup RESUME;                                                      | |||
48  |    endfind;                                                             | |||-12
49  |  endoperation                                                           | |-11
50  |                                                                         | *2
51  |endfacet                                                                 | -2
52  |                                                                         *1
53  |initial REGISTRY                                                         |
54  |                                                                         |
55  |endscript                                                                |-1
```

fig. 14 : Alarmclock Script (Correct Solution)

```
=========================================================================

          PROGRAM GENERATED ON 1981/12/22 AT 11:59:34.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1982/01/18 AT 11:05:29.00
          =======================================================

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;DISPLAY                                                               <==
```

IDENTIFIER	TYPE	VALUE
ALARM_CLOCK	SCRIPT	ALARM_CLOCK

```
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : AC := NEW ALARM_CLOCK;                                     <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OPER : TICK,WAKE_ME;PORT : P                                           <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE;                                                               <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND WAKE_ME(3) TO AC REPLY TO P; RUN                                  <==
+++      0.000  P1: INTERFACE(1) SENDS WAKE_ME(INT:3),REPLY TO: P
                        TO ALARM_CLOCK(1)
+++      0.000  P2: ALARM_CLOCK(1) IS IDLE
+++      0.100  P2: ALARM_CLOCK(1) RECEIVES WAKE_ME(INT:3),REPLY TO: P
                        FROM INTERFACE(1)
+++      0.100  P2: ALARM_CLOCK(1) STARTING OPERATION WAKE_ME(INT:
                       3)
+++      0.700  P2: ALARM_CLOCK(1) SENDS WAKE_UP(INT:3),REPLY TO: P
                        TO ALARM_CLOCK(1)
+++      0.700  P2: ALARM_CLOCK(1) RECEIVES WAKE_UP(INT:3),REPLY TO: P
                        FROM ALARM_CLOCK(1)
+++      0.700  P2: ALARM_CLOCK(1) IS IDLE
>>>      0.700  SYSTEM TERMINATED
>>>      0.700  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND TICK TO AC;RUN                                                    <==
+++      0.700  P1: INTERFACE(1) SENDS TICK() TO ALARM_CLOCK(1)
+++      0.800  P2: ALARM_CLOCK(1) RECEIVES TICK() FROM INTERFACE(1)
+++      0.800  P2: ALARM_CLOCK(1) STARTING OPERATION TICK()
+++      1.000  P2: ALARM_CLOCK(1) IS IDLE
>>>      1.000  SYSTEM TERMINATED
>>>      1.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND WAKE_ME(2) TO AC REPLY TO P; RUN                                  <==
+++      1.100  P1: INTERFACE(1) SENDS WAKE_ME(INT:2),REPLY TO: P
                        TO ALARM_CLOCK(1)
+++      1.200  P2: ALARM_CLOCK(1) RECEIVES WAKE_ME(INT:2),REPLY TO: P
                        FROM INTERFACE(1)
+++      1.200  P2: ALARM_CLOCK(1) STARTING OPERATION WAKE_ME(INT:
                       2)
+++      1.600  P2: ALARM_CLOCK(1) SENDS WAKE_UP(INT:3),REPLY TO: P
                        TO ALARM_CLOCK(1)
+++      1.600  P2: ALARM_CLOCK(1) RECEIVES WAKE_UP(INT:3),REPLY TO: P
                        FROM ALARM_CLOCK(1)
+++      1.600  P2: ALARM_CLOCK(1) IS IDLE
>>>      1.600  SYSTEM TERMINATED
>>>      1.600  INTERFACE(1) : ENTER CSSA COMMAND -
==>         SEND TICK TO AC                                                <==
+++      1.600  P1: INTERFACE(1) SENDS TICK() TO ALARM_CLOCK(1)
>>>      1.600  INTERFACE(1) : ENTER CSSA COMMAND -
==> TRACE AC; SEND TICK TO AC; RUN                                         <==
+++      1.600  P1: INTERFACE(1) SENDS TICK() TO ALARM_CLOCK(1)
+++      1.700  P2: ALARM_CLOCK(1) RECEIVES TICK() FROM INTERFACE(1)
```

fig. 15 : Alarmclock Execution Protocol

```
___      1.700  TRACING ALARM_CLOCK(1) / REGISTRY ____________________________
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE
                FOUND : TICK()
                STARTING OPERATION TICK()
                LINE  45: NOW := 2
                LINE  46: WAKE_UP_TIME := 2
                LINE  46: RECORD NOT FOUND
_____________________________________________________________________________

>>>      1.700  REAL-TIME LIMIT EXCEEDED
>>>      1.700  INTERFACE(1) : ENTER CSSA COMMAND -                          <==
==>
>>>      1.700  INTERFACE(1) : ENTER CSSA COMMAND -                          <==
==> RUN

___      1.900  TRACING ALARM_CLOCK(1) / REGISTRY / TICK ____________________
_____________________________________________________________________________

___      1.900  TRACING ALARM_CLOCK(1) / REGISTRY ____________________________
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE - NO MESSAGE FOUND
_____________________________________________________________________________

+++      1.900  P2: ALARM_CLOCK(1) IS IDLE
+++      1.900  P2: ALARM_CLOCK(1) RECEIVES TICK() FROM INTERFACE(1)

___      1.900  TRACING ALARM_CLOCK(1) / REGISTRY ____________________________
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                SEARCHING NEXT_MESSAGE
                FOUND : TICK()
                STARTING OPERATION TICK()
                LINE  45: NOW := 3
                LINE  46: WAKE_UP_TIME := 3
                LINE  46: RECORD FOUND
_____________________________________________________________________________

>>>      1.900  REAL-TIME LIMIT EXCEEDED
>>>      1.900  INTERFACE(1) : ENTER CSSA COMMAND -
==> RUN                                                                     <==

___      2.200  TRACING ALARM_CLOCK(1) / REGISTRY / TICK ____________________
                LINE  47: FACETTING : REGISTRY --> RESUME
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                LINE  16: TIME := 3
                STARTING OPERATION WAKE_UP(INT:3)
_____________________________________________________________________________


___      2.300  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ____________________
                LINE  17: SEND *REPLY() TO INTERFACE(1)
_____________________________________________________________________________


___      2.300  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ____________________
                LINE  18: WAKE_UP_TIME := 3
                LINE  18: RECORD FOUND
                LINE  22: NO_OF_SLEEPERS := 1
_____________________________________________________________________________


___      2.600  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ____________________
_____________________________________________________________________________


___      2.600  TRACING ALARM_CLOCK(1) / RESUME ____________________________
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_UP(INT:3),REPLY TO: P
                LINE  16: TIME := 3
                STARTING OPERATION WAKE_UP(INT:3)
_____________________________________________________________________________
```

fig. 15 -continued-

```
>>>      2.600  REAL-TIME LIMIT EXCEEDED
>>>      2.600  INTERFACE(1) : ENTER CSSA COMMAND -
==> RUN                                                                    <==
+++      2.600  P1: INTERFACE(1) RECEIVES *REPLY() FROM ALARM_CLOCK(1)

___      2.700  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ________________
                LINE  17: SEND *REPLY() TO INTERFACE(1)
_____________________________________________________________________

___      2.700  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ________________
                LINE  18: WAKE_UP_TIME := 3
                LINE  18: RECORD FOUND
                LINE  20: RECORD DELETED
_____________________________________________________________________

___      3.100  TRACING ALARM_CLOCK(1) / RESUME / WAKE_UP ________________
                LINE  21: FACETTING : RESUME --> REGISTRY
_____________________________________________________________________

___      3.100  TRACING ALARM_CLOCK(1) / REGISTRY ________________________
                SEARCHING NEXT_MESSAGE - NO MESSAGE FOUND
_____________________________________________________________________

+++      3.100  P2: ALARM_CLOCK(1) IS IDLE
+++      3.600  P1: INTERFACE(1) RECEIVES *REPLY() FROM ALARM_CLOCK(1)
>>>      3.700  SYSTEM TERMINATED
>>>      3.700  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND WAKE_ME(0) TO AC REPLY TO P;RUN                                   <==
+++      3.700  P1: INTERFACE(1) SENDS WAKE_ME(INT:0),REPLY TO: P
                        TO ALARM_CLOCK(1)
+++      3.800  P2: ALARM_CLOCK(1) RECEIVES WAKE_ME(INT:0),REPLY TO: P
                        FROM INTERFACE(1)

___      3.800  TRACING ALARM_CLOCK(1) / REGISTRY ________________________
                SEARCHING NEXT_MESSAGE
                FOUND : WAKE_ME(INT:0),REPLY TO: P
                LINE  28: DELAY_TIME := 0
                STARTING OPERATION WAKE_ME(INT:0)
_____________________________________________________________________

___      4.000  TRACING ALARM_CLOCK(1) / REGISTRY / WAKE_ME _____________
                LINE  32: SEND *REPLY() TO INTERFACE(1)
_____________________________________________________________________

___      4.000  TRACING ALARM_CLOCK(1) / REGISTRY / WAKE_ME _____________
_____________________________________________________________________

___      4.000  TRACING ALARM_CLOCK(1) / REGISTRY / WAKE_ME _____________
_____________________________________________________________________

___      4.000  TRACING ALARM_CLOCK(1) / REGISTRY ________________________
                SEARCHING NEXT_MESSAGE - NO MESSAGE FOUND
_____________________________________________________________________

+++      4.000  P2: ALARM_CLOCK(1) IS IDLE
>>>      4.000  REAL-TIME LIMIT EXCEEDED
>>>      4.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> NOTRACE AC                                                             <==
>>>      4.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                              <==
 +++      4.000  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			*REPLY *REPLY
P2: ALARM_CLOCK(1	REGISTRY		

```
                CSSA-SESSION-STATISTICS
                =======================


SESSION  STARTED  AT  11:05:24.00
SESSION TERMINATED AT 11:15:47.00  ON 1982/01/18
REAL-TIME USED :  623.00 SEC.
CPU-TIME  USED :    2.02 SEC.
SIMULATION TIME USED :      4.0000 SEC.
NUMBER OF AGENTS CREATED :    1
NUMBER OF MESSAGES SENT :   11
```

fig. 15 -continued-

```
alarmclock = monitor is create, wakeme, tick;

pq=priority_queue;

rep= record[wakeup: pq, now: int];

create = proc() returns(cvt);
        return (rep${wakeup: pq$create(), now: 0});
        end create;

wakeme = proc(ac: cvt, time: int)
        alarmsetting: int := time+ac.now;
        while ac.now < alarmsetting do
           pq$wait(ac.wakeup, alarmsetting)
           end;
        %the while statement is necessary because the first process on the
        %queue is awakened every tick.
        pq$signal(ac.wakeup);
        %in case the next process has same wakeup time.
        end wakeme;

tick = procedure(ac:cvt);
        ac.now := ac.now + 1;
        pq$signal(ac.wakeup);
        end tick;

end alarmclock;
```

fig. 16 : Alarmclock Monitor

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| BLOCKNESTING
 1  type NQUEENS is script                                                    |+1
 2                |                                                         |
 3    type NODE is script(int:N) assert N > 0                                 |+2
 4                ||                                                        |
 5      facet ONLY_FACET is                                                   |+3
 6                ||                                                        |
 7      public: FIND_SOLUTION;                                                |
 8                ||                                                        |
 9      var array(1..N) of int : Q;                                           |
10                ||                                                        |
11      operation FIND_SOLUTION(-> Q) is                                      |+4
12                ||                                                        |
13        var int: I,ANZ_Q;                                                   |
14        var array(1..N) of bool: NOT_POSSIBLE;                              |
15        port : OK;                                                          |
16        oper : ANSWER;                                                      |
17        functionhead GENSTRING(int : N) returns string external;            |
18        ------------------------------------------------------              |
19        if not( Q(N) = 0)                                                   |+5
20        then begin                                                          |+6
21        |     var string : ASW ;                                            |
22        |     ------------------------------------                         |
23        |     ASW := "One solution for " + GENSTRING(N) +                   |
24        | ||     " queens is : ";                                           |
25        |     loop for I in 1..N-1  do                                      |+7
26        |     | ASW := ASW + GENSTRING(Q(I)) + ",";                         |
27        |     endloop;                                                      |-7
28        |     ASW := ASW + GENSTRING(Q(N));                                 |*6
29        |     send ANSWER(ASW) to interface reply to OK;                    |
30        |     wait OK;                                                      |
31        |     end;                                                          |-6
32        |       ||                                                          |*5
33        else                                                                |
34        loop for I in 1..N  while not( Q(I) = 0)                            |+8
35        | do ANZ_Q := ANZ_Q + 1;                                            |
36        endloop;                                                            |-8
37        |       ||                                                          |*5
38        loop for I in 1..ANZ_Q do                                           |+9
39        | NOT_POSSIBLE(Q(I)) := true;                                       |
40        | if Q(I) + (ANZ_Q - I) + 1 <=  N then                              |+10
41        | | NOT_POSSIBLE(Q(I) + ANZ_Q - I + 1) := true;                     |
42        | endif;                                                            |-10
43        | if Q(I) - (ANZ_Q - I) - 1 >= 1 then                               |+11
44        | | NOT_POSSIBLE(Q(I) - (ANZ_Q - I) - 1) := true;                   |
45        | endif;                                                            |-11
46        endloop;                                                            |-9
47        |       ||                                                          |*5
48        loop for I in 1..N do                                               |+12
49        | if not NOT_POSSIBLE(I)  then                                      |+13
50        | | Q(ANZ_Q + 1) := I;                                              |
51        | | send FIND_SOLUTION(Q) to new ownmode(N);                        |
52        | endif;                                                            |-13
53        endloop;                                                            |-12
54        endif;                                                              |-5
55                ||                                                        |*4
56        terminate;                                                          |
```

fig. 17 : N-Queens Script

```
B M S - C S S A - C O M P I L E R                                   1981/10/21   09:09:

    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| B L O C K N E S T I N G
 57 |      | endoperation                                                      |  |-4
 58 |      endfacet                                                           |  |-3
 59 |                 ||                                                      |  |*2
 60 |      initial ONLY_FACET;                                                |  |
 61 |    endscript;                                                           |  |-2
 62 |                 |                                                       | |*1
 63 |                 |                                                       | |
 64 |    facet ONLY_FACET is                                                  |  |+14
 65 |    |            |                                                       |  |
 66 |    |  public : STOPQ, FIND_SOLUTION;                                    |  |
 67 |    |            |                                                       |  |
 68 |    |  operation FIND_SOLUTION(int: N) assert N > 0 is                   |  |  |+15
 69 |    |  |         |                                                       |  |  |
 70 |    |  |  var int: I;                                                    |  |  |
 71 |    |  |  var array (1..N) of int: FIRST_ROW;                            |  |  |
 72 |    |  |  port : OK;                                                     |  |  |
 73 |    |  |  oper : ANSWER;                                                 |  |  |
 74 |    |  |  ------------------------------                                 |  |  |
 75 |    |  |  if N = 1                                                       |  |  |  |+16
 76 |    |  |  then                                                           |  |  |  |
 77 |    |  |  | send ANSWER("For one queen solution is : 1")                  |  |  |  |
 78 |    |  |  |            | to interface reply to OK;                       |  |  |  |
 79 |    |  |  | wait OK;                                                     |  |  |  |
 80 |    |  |  |         |                                                    |  |  |  |
 81 |    |  |  else                                                           |  |  |  |
 82 |    |  |  loop for I in 1..N do                                          |  |  |  |  |+17
 83 |    |  |  | FIRST_ROW(1) := I;                                           |  |  |  |  |
 84 |    |  |  | send FIND_SOLUTION(FIRST_ROW) to new NODE(N);                |  |  |  |  |
 85 |    |  |  endloop;                                                       |  |  |  |  |-17
 86 |    |  |  endif;                                                         |  |  |  |-16
 87 |    |  endoperation;                                                     |  |  |-15
 88 |    |            |                                                       |  | |*14
 89 |    |  operation STOPQ is                                                |  |  |+18
 90 |    |  |         |                                                       |  |  |
 91 |    |  |  terminate;                                                     |  |  |
 92 |    |  |         |                                                       |  |  |
 93 |    |  endoperation                                                      |  |  |-18
 94 |    |            |                                                       |  | |*14
 95 |    endfacet                                                             |  | |-14
 96 |                 |                                                       | |*1
 97 |    initial ONLY_FACET                                                   |  | |
 98 |                 |                                                       |  | |
 99 |  endscript                                                              |  | |-1
```

fig. 17 -continued-

```
==========================================================================

          PROGRAM GENERATED ON 1982/01/18 AT 11:26:35.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1982/01/18 AT 11:43:19.00
          ====================================================

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;DISPLAY                                                          <==

| IDENTIFIER | TYPE | VALUE |
|---|---|---|
| NODE | SCRIPT | NODE |
| NQUEENS | SCRIPT | NQUEENS |

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : NQ := NEW NQUEENS; OPER : FIND_SOLUTION               <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> OBSERVE; SEND FIND_SOLUTION(4) TO NQ;RUN                          <==
+++      0.000  P1: INTERFACE(1) SENDS FIND_SOLUTION(INT:4) TO NQUEENS(1)
+++      0.000  P2: NQUEENS(1) IS IDLE
+++      0.100  P2: NQUEENS(1) RECEIVES FIND_SOLUTION(INT:4) FROM INTERFACE(1)
+++      0.100  P2: NQUEENS(1) STARTING OPERATION FIND_SOLUTION(INT:
                    4)
+++      0.500  P2: NQUEENS(1) CREATES NODE(1)(INT:4)
+++      0.500  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:1,INT:
                    0,INT:0,INT:0) TO NODE(1)
+++      0.500  P3: NODE(1) IS IDLE
+++      0.700  P2: NQUEENS(1) CREATES NODE(2)(INT:4)
+++      0.700  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:2,INT:
                    0,INT:0,INT:0) TO NODE(2)
+++      0.700  P4: NODE(2) IS IDLE
+++      0.800  P4: NODE(2) RECEIVES FIND_SOLUTION(INT:2,INT:
                    0,INT:0,INT:0) FROM NQUEENS(1)
+++      0.800  P4: NODE(2) STARTING OPERATION FIND_SOLUTION(INT:
                    2,INT:0,INT:0,INT:0)
+++      0.900  P2: NQUEENS(1) CREATES NODE(3)(INT:4)
+++      0.900  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:3,INT:
                    0,INT:0,INT:0) TO NODE(3)
+++      0.900  P5: NODE(3) IS IDLE
+++      1.000  P5: NODE(3) RECEIVES FIND_SOLUTION(INT:3,INT:
                    0,INT:0,INT:0) FROM NQUEENS(1)
+++      1.000  P5: NODE(3) STARTING OPERATION FIND_SOLUTION(INT:
                    3,INT:0,INT:0,INT:0)
+++      1.100  P2: NQUEENS(1) CREATES NODE(4)(INT:4)
+++      1.100  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:4,INT:
                    0,INT:0,INT:0) TO NODE(4)
+++      1.100  P6: NODE(4) IS IDLE
+++      1.100  P2: NQUEENS(1) IS IDLE
+++      1.100  P3: NODE(1) RECEIVES FIND_SOLUTION(INT:1,INT:
                    0,INT:0,INT:0) FROM NQUEENS(1)
+++      1.100  P3: NODE(1) STARTING OPERATION FIND_SOLUTION(INT:
                    1,INT:0,INT:0,INT:0)
+++      2.300  P5: NODE(3) CREATES NODE(5)(INT:4)
+++      2.300  P5: NODE(3) SENDS FIND_SOLUTION(INT:3,INT:1,INT:
                    0,INT:0) TO NODE(5)
+++      2.300  P7: NODE(5) IS IDLE
+++      2.400  P4: NODE(2) CREATES NODE(6)(INT:4)
+++      2.400  P4: NODE(2) SENDS FIND_SOLUTION(INT:2,INT:4,INT:
                    0,INT:0) TO NODE(6)
```

fig. 18 : N-Queens Execution Protocol

```
+++      2.400  P2: NODE(6) IS IDLE
+++      2.500  P2: NODE(6) RECEIVES FIND_SOLUTION(INT:2,INT:
                        4,INT:0,INT:0) FROM NODE(2)
+++      2.500  P4: NODE(2) TERMINATED
+++      2.500  P2: NODE(6) STARTING OPERATION FIND_SOLUTION(INT:
                        2,INT:4,INT:0,INT:0)
+++      2.500  P3: NODE(1) CREATES NODE(7)(INT:4)
+++      2.500  P3: NODE(7) IS IDLE
+++      2.500  P3: NODE(1) SENDS FIND_SOLUTION(INT:1,INT:3,INT:
                        0,INT:0) TO NODE(7)
+++      2.500  P3: NODE(7) RECEIVES FIND_SOLUTION(INT:1,INT:
                        3,INT:0,INT:0) FROM NODE(1)
+++      2.500  P3: NODE(7) STARTING OPERATION FIND_SOLUTION(INT:
                        1,INT:3,INT:0,INT:0)
>>>      2.500  REAL-TIME LIMIT EXCEEDED
>>>      2.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;STATUS                                                    <==
 +++      2.500  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			
P2: NQUEENS(1)	ONLY_FACE		
P3: NODE(1) *	ONLY_FACE	FIND_SOLU	N
P5: NODE(3) *	ONLY_FACE	FIND_SOLU	N
P6: NODE(4)	ONLY_FACE		
P7: NODE(5)	ONLY_FACE		
P2: NODE(6) *	ONLY_FACE	FIND_SOLU	N
P3: NODE(7) *	ONLY_FACE	FIND_SOLU	N

```
>>>      2.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;SYSSTATUS                                                 <==
+++      2.500  SYSTEM STATUS:
```

PROC.	UTIL.	AGENTS
P1	100%	INTERFACE(1)*
P2	0%	NODE(6)* NQUEENS(1)
P3	56%	NODE(7)* NODE(1)*
P4	68%	
P5	60%	NODE(3)*
P6	1%	NODE(4)
P7	1%	NODE(5)

BUS	UTIL.	#MSGS	AVG-QLEN	MESSAGES
B1	5%	4	0.00	
B2	4%	2	0.00	
B3	5%	2	0.00	
B4	5%	2	0.00	
B5	0%	1	0.00	FIND_SOLUTION
B6	1%	0	0.00	

```
>>>      2.500  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                       <==
+++      2.600  P6: NODE(4) RECEIVES FIND_SOLUTION(INT:4,INT:
                        0,INT:0,INT:0) FROM NQUEENS(1)
+++      2.600  P6: NODE(4) STARTING OPERATION FIND_SOLUTION(INT:
                        4,INT:0,INT:0,INT:0)
+++      2.700  P5: NODE(3) TERMINATED
+++      3.600  P7: NODE(5) RECEIVES FIND_SOLUTION(INT:3,INT:
                        1,INT:0,INT:0) FROM NODE(3)
+++      3.600  P7: NODE(5) STARTING OPERATION FIND_SOLUTION(INT:
                        3,INT:1,INT:0,INT:0)
+++      3.800  P6: NODE(4) CREATES NODE(8)(INT:4)
+++      3.800  P6: NODE(4) SENDS FIND_SOLUTION(INT:4,INT:1,INT:
                        0,INT:0) TO NODE(8)
+++      3.800  P4: NODE(8) IS IDLE
```

fig. 18 -continued-

```
+++      3.800  P3: NODE(1) CREATES NODE(9)(INT:4)
+++      3.800  P5: NODE(9) IS IDLE
+++      4.100  P6: NODE(4) CREATES NODE(10)(INT:4)
+++      4.100  P6: NODE(10) IS IDLE
+++      4.100  P6: NODE(4) SENDS FIND_SOLUTION(INT:4,INT:2,INT:
                        0,INT:0) TO NODE(10)
+++      4.100  P6: NODE(10) RECEIVES FIND_SOLUTION(INT:4,INT:
                        2,INT:0,INT:0) FROM NODE(4)
+++      4.100  P6: NODE(10) STARTING OPERATION FIND_SOLUTION(INT:
                        4,INT:2,INT:0,INT:0)
>>>      4.100  REAL-TIME LIMIT EXCEEDED
>>>      4.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                              <==
+++      4.200  P2: NODE(6) CREATES NODE(11)(INT:4)
+++      4.200  P2: NODE(6) SENDS FIND_SOLUTION(INT:2,INT:4,INT:
                        1,INT:0) TO NODE(11)
+++      4.600  P2: NODE(6) TERMINATED
+++      4.600  P7: NODE(11) IS IDLE
+++      4.800  P3: NODE(7) TERMINATED
+++      4.800  P3: NODE(1) SENDS FIND_SOLUTION(INT:1,INT:4,INT:
                        0,INT:0) TO NODE(9)
+++      4.900  P3: NODE(1) TERMINATED
+++      5.200  P5: NODE(9) RECEIVES FIND_SOLUTION(INT:1,INT:
                        4,INT:0,INT:0) FROM NODE(1)
+++      5.200  P5: NODE(9) STARTING OPERATION FIND_SOLUTION(INT:
                        1,INT:4,INT:0,INT:0)
+++      5.300  P4: NODE(8) RECEIVES FIND_SOLUTION(INT:4,INT:
                        1,INT:0,INT:0) FROM NODE(4)
+++      5.300  P4: NODE(8) STARTING OPERATION FIND_SOLUTION(INT:
                        4,INT:1,INT:0,INT:0)
+++      5.400  P6: NODE(4) TERMINATED
+++      5.600  P7: NODE(5) CREATES NODE(12)(INT:4)
+++      5.600  P7: NODE(11) RECEIVES FIND_SOLUTION(INT:2,INT:
                        4,INT:1,INT:0) FROM NODE(6)
+++      5.600  P7: NODE(11) STARTING OPERATION FIND_SOLUTION(INT:
                        2,INT:4,INT:1,INT:0)
+++      5.600  P2: NODE(12) IS IDLE
>>>      5.600  REAL-TIME LIMIT EXCEEDED
>>>      5.600  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                              <==
+++      6.400  P6: NODE(10) TERMINATED
+++      6.600  P7: NODE(5) SENDS FIND_SOLUTION(INT:3,INT:1,INT:
                        4,INT:0) TO NODE(12)
+++      7.000  P5: NODE(9) CREATES NODE(13)(INT:4)
+++      7.000  P5: NODE(9) SENDS FIND_SOLUTION(INT:1,INT:4,INT:
                        2,INT:0) TO NODE(13)
+++      7.000  P3: NODE(13) IS IDLE
+++      7.100  P2: NODE(12) RECEIVES FIND_SOLUTION(INT:3,INT:
                        1,INT:4,INT:0) FROM NODE(5)
+++      7.100  P2: NODE(12) STARTING OPERATION FIND_SOLUTION(INT:
                        3,INT:1,INT:4,INT:0)
+++      7.200  P4: NODE(8) CREATES NODE(14)(INT:4)
+++      7.200  P4: NODE(14) IS IDLE
+++      7.200  P4: NODE(8) SENDS FIND_SOLUTION(INT:4,INT:1,INT:
                        3,INT:0) TO NODE(14)
+++      7.200  P4: NODE(14) RECEIVES FIND_SOLUTION(INT:4,INT:
                        1,INT:3,INT:0) FROM NODE(8)
+++      7.200  P4: NODE(14) STARTING OPERATION FIND_SOLUTION(INT:
                        4,INT:1,INT:3,INT:0)
>>>      7.200  REAL-TIME LIMIT EXCEEDED
>>>      7.200  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                              <==
+++      7.300  P5: NODE(9) TERMINATED
+++      7.700  P7: NODE(5) TERMINATED
+++      8.000  P7: NODE(11) CREATES NODE(15)(INT:4)
+++      8.000  P7: NODE(11) SENDS FIND_SOLUTION(INT:2,INT:4,INT:
                        1,INT:3) TO NODE(15)
+++      8.000  P5: NODE(15) IS IDLE
+++      8.100  P3: NODE(13) RECEIVES FIND_SOLUTION(INT:1,INT:
                        4,INT:2,INT:0) FROM NODE(9)
+++      8.100  P3: NODE(13) STARTING OPERATION FIND_SOLUTION(INT:
                        1,INT:4,INT:2,INT:0)
+++      8.200  P7: NODE(11) TERMINATED
+++      8.400  P4: NODE(8) TERMINATED
+++      9.300  P2: NODE(12) CREATES NODE(16)(INT:4)
+++      9.300  P2: NODE(12) SENDS FIND_SOLUTION(INT:3,INT:1,INT:
                        4,INT:2) TO NODE(16)
```

fig. 18 -continued-

```
+++      9.300  P6: NODE(16) IS IDLE
+++      9.400  P5: NODE(15) RECEIVES FIND_SOLUTION(INT:2,INT:
                          4,INT:1,INT:3) FROM NODE(11)
+++      9.400  P5: NODE(15) STARTING OPERATION FIND_SOLUTION(INT:
                          2,INT:4,INT:1,INT:3)
+++      9.600  P2: NODE(12) TERMINATED
+++      9.900  P4: NODE(14) TERMINATED
One solution for 4 queens is : 2,4,1,3"),REPLY TO: OK TO INTERFACE(1)
+++     10.400  P5: NODE(15) IS IDLE
+++     10.600  P3: NODE(13) TERMINATED
+++     10.700  P6: NODE(16) RECEIVES FIND_SOLUTION(INT:3,INT:
                          1,INT:4,INT:2) FROM NODE(12)
+++     10.700  P6: NODE(16) STARTING OPERATION FIND_SOLUTION(INT:
                          3,INT:1,INT:4,INT:2)
One solution for 4 queens is : 2,4,1,3"),REPLY TO: OK FROM NODE(15)
One solution for 4 queens is : 3,1,4,2"),REPLY TO: OK TO INTERFACE(1)
+++     11.700  P6: NODE(16) IS IDLE
One solution for 4 queens is : 3,1,4,2"),REPLY TO: OK FROM NODE(16)
>>>     12.100  SYSTEM TERMINATED
>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                          <==
MAILBOX OF INTERFACE(1) :

(1) ANSWER(STRING:"One solution for 4 queens is : 2,4,1,3"),REPLY TO: OK
(2) ANSWER(STRING:"One solution for 4 queens is : 3,1,4,2"),REPLY TO: OK

>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> REPLY 1                                                          <==
+++     12.100  P1: INTERFACE(1) SENDS *REPLY() TO NODE(15)
>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                          <==
MAILBOX OF INTERFACE(1) :

(1) ANSWER(STRING:"One solution for 4 queens is : 2,4,1,3"),REPLY TO: OK
(2) ANSWER(STRING:"One solution for 4 queens is : 3,1,4,2"),REPLY TO: OK

>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> REPLY 2                                                          <==
+++     12.100  P1: INTERFACE(1) SENDS *REPLY() TO NODE(16)
>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;STATUS                                                          <==
 +++     12.100  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			ANSWER ANSWER
P2: NQUEENS(1)	ONLY_FACE		
P5: NODE(15)	ONLY_FACE	FIND_SOLU	N
P6: NODE(16)	ONLY_FACE	FIND_SOLU	N

```
>>>     12.100  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;RUN                                                             <==
+++     12.300  P5: NODE(15) RECEIVES *REPLY() FROM INTERFACE(1)
+++     12.300  P6: NODE(16) RECEIVES *REPLY() FROM INTERFACE(1)
+++     12.400  P5: NODE(15) TERMINATED
+++     12.400  P6: NODE(16) TERMINATED
>>>     12.400  SYSTEM TERMINATED
>>>     12.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;STATUS                                                          <==
 +++     12.400  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			ANSWER ANSWER
P2: NQUEENS(1)	ONLY_FACE		

fig. 18 -continued-

```
>>>    12.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND FIND_SOLUTION(2) TO NQ;RUN                                    <==
+++    12.400  P1: INTERFACE(1) SENDS FIND_SOLUTION(INT:2) TO NQUEENS(1)
+++    12.500  P2: NQUEENS(1) RECEIVES FIND_SOLUTION(INT:2) FROM INTERFACE(1)
+++    12.500  P2: NQUEENS(1) STARTING OPERATION FIND_SOLUTION(INT:
                       2)
+++    12.900  P2: NQUEENS(1) CREATES NODE(17)(INT:2)
+++    12.900  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:1,INT:
                       0) TO NODE(17)
+++    12.900  P7: NODE(17) IS IDLE
+++    13.100  P2: NQUEENS(1) CREATES NODE(18)(INT:2)
+++    13.100  P2: NODE(18) IS IDLE
+++    13.100  P2: NQUEENS(1) SENDS FIND_SOLUTION(INT:2,INT:
                       0) TO NODE(18)
+++    13.100  P2: NODE(18) RECEIVES FIND_SOLUTION(INT:2,INT:
                       0) FROM NQUEENS(1)
+++    13.100  P2: NODE(18) STARTING OPERATION FIND_SOLUTION(INT:
                       2,INT:0)
+++    13.600  P7: NODE(17) RECEIVES FIND_SOLUTION(INT:1,INT:
                       0) FROM NQUEENS(1)
+++    13.600  P7: NODE(17) STARTING OPERATION FIND_SOLUTION(INT:
                       1,INT:0)
+++    14.100  P2: NQUEENS(1) IS IDLE
+++    14.300  P2: NODE(18) TERMINATED
+++    14.800  P7: NODE(17) TERMINATED
>>>    14.800  SYSTEM TERMINATED
>>>    14.800  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;STATUS                                                            <==
 +++    14.800  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) * P2: NQUEENS(1)	ONLY_FACE		ANSWER ANSWER

```
>>>    14.800  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX
MAILBOX OF INTERFACE(1) :
(1) ANSWER(STRING:"One solution for 4 queens is : 2,4,1,3"),REPLY TO: OK
(2) ANSWER(STRING:"One solution for 4 queens is : 3,1,4,2"),REPLY TO: OK

>>>    14.800  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                          <==
 +++    14.800  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) * P2: NQUEENS(1)	ONLY_FACE		ANSWER ANSWER

```
                CSSA-SESSION-STATISTICS
                =======================

SESSION  STARTED  AT  11:42:59.00
SESSION TERMINATED AT 11:49:31.00  ON 1982/01/18
REAL-TIME USED :  392.00 SEC.
CPU-TIME  USED :    6.34 SEC.
SIMULATION TIME USED :     14.8000 SEC.
NUMBER OF AGENTS CREATED :   19
NUMBER OF MESSAGES SENT :   24
```

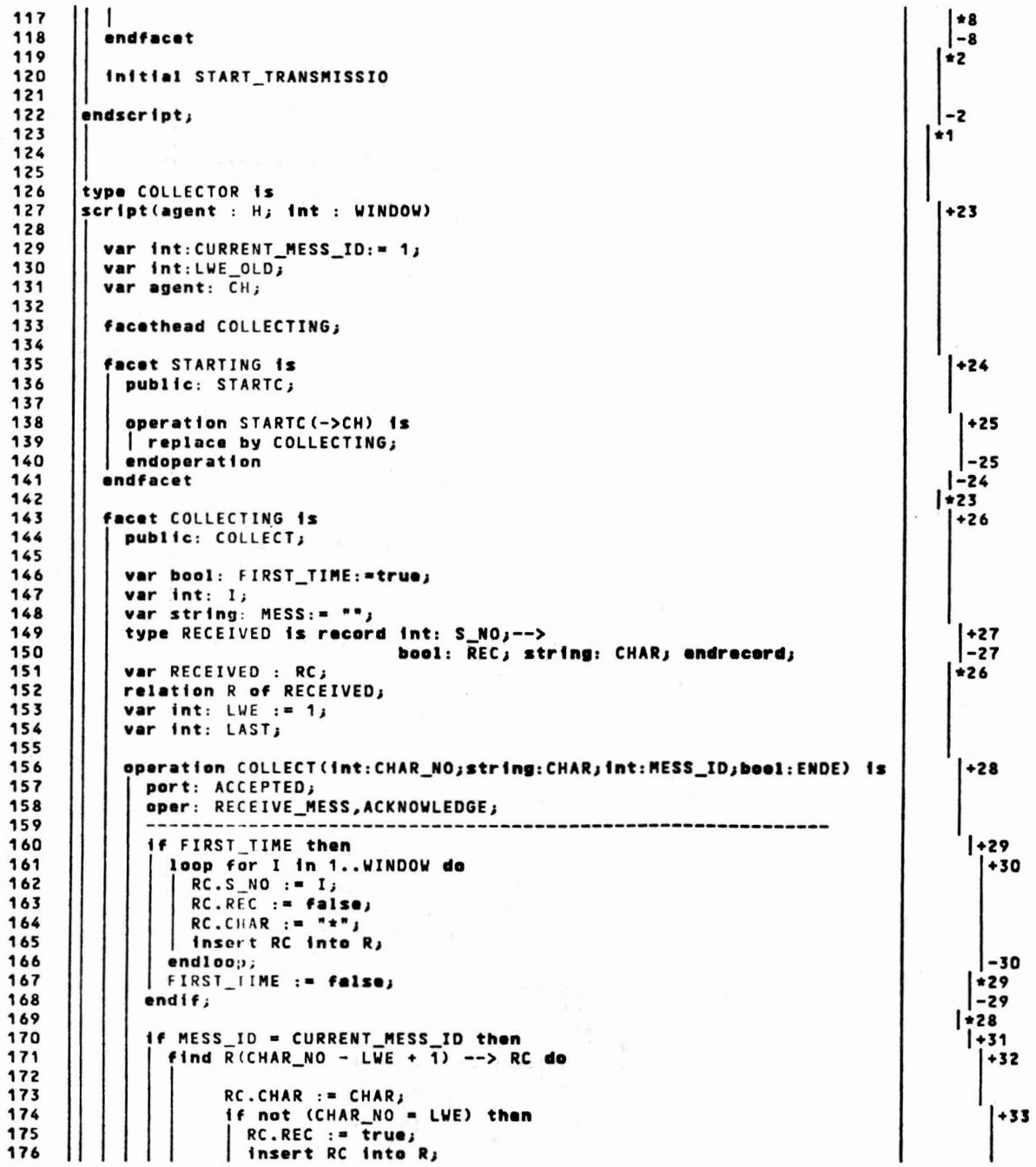
fig. 18 -continued-

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+----7--| BLOCKNESTING
 1  |type START_NETWORK is                                                    |
 2  |script(int : UP1,GEN_ZUF1,UP2,GEN_ZUF2;                                  |+1
 3  |      bool : SHOW1,SHOW2)   assert UP1 > 0 and UP2 > 0                   |
 4  |                                                                         |
 5  |type TRANSMITTER is                                                      |
 6  |script(agent : CH; int : WINDOW; agent : H)                              | +2
 7  |                                                                         |
 8  | procedure ACTIVATE_TRANSMIT(int:N) is                                   |  +3
 9  | | var int: I;                                                           |
10  | | oper: SEND_PACKET;                                                    |
11  | | ------------------                                                    |
12  | | loop for I in 1..N do                                                 |   +4
13  | | | send SEND_PACKET to self;                                           |
14  | | endloop;                                                              |  -4
15  | endprocedure;                                                           | -3
16  |                                                                         |*2
17  | var string: MESS:="";                                                   |
18  | var int: LENGTH_OF_MESS, CURRENT_MESS_ID := 0;                          |
19  |                                                                         |
20  | facethead TRANSMISSION1;                                                |
21  |                                                                         |
22  | facet START_TRANSMISSION is                                             |  +5
23  | | public: SEND_MESS;                                                    |
24  | |                                                                       |
25  | | operation SEND_MESS(string: M) is                                     |   +6
26  | | | functionhead LENGTH (string: P1) returns int external;              |
27  | | | oper: NEXT_MESS;                                                    |
28  | | | ------------------------------------                                |
29  | | | CURRENT_MESS_ID := CURRENT_MESS_ID + 1;                             |
30  | | | MESS := M;                                                          |
31  | | | LENGTH_OF_MESS := LENGTH(MESS);                                     |
32  | | | if LENGTH_OF_MESS < WINDOW then                                     |    +7
33  | | | | call ACTIVATE_TRANSMIT(LENGTH_OF_MESS);                           |
34  | | | else call ACTIVATE_TRANSMIT(WINDOW);                                |
35  | | | endif;                                                              |   -7
36  | | | send NEXT_MESS to H;                                                |  *6
37  | | | replace by TRANSMISSION1;                                           |
38  | | endoperation                                                          |  -6
39  | |                                                                       | *5
40  | endfacet                                                                | -5
41  |                                                                         |*2
42  | facet TRANSMISSION1 is                                                  |  +8
43  | | public: ACKNOWLEDGE;                                                  |
44  | | private: SEND_PACKET;                                                 |
45  | |                                                                       |
46  | | functionhead MOD (int: P1,P2) returns int external;                   |
47  | | functionhead SUBSTR (string:P1;int:P2,P3) returns string external;    |
48  | | functionhead GENSTRING(int:P1) returns string external;               |
49  | | type PACKET is record int:CHAR_NO;--> string:CHAR;int: MESS_ID;       |   +9
50  | |                      bool:ENDE; endrecord;                            |  -9
51  | | var PACKET: P;                                                        | *8
52  | | relation WINDOW_PACKETS  of PACKET;                                   |
53  | | var int: S_NO,LWE := 1;                                               |
54  | | var bool: IDLE_ENABLED := false;                                      |
55  | | oper: COLLECT;                                                        |
56  | |                                                                       |
```

fig. 19 : Window Mechanism Script

```
57  facet TRANSMISSION2 is
58    include : ACKNOWLEDGE,SEND_PACKET;
59
60    operation idle is
61      find WINDOW_PACKETS(LWE) --> P do
62        send COLLECT(P) to CH;
63      endfind;
64      IDLE_ENABLED := false;
65      leave;
66    endoperation
67
68  endfacet
69
70
71  operation SEND_PACKET is
72
73    P.CHAR := SUBSTR(MESS,S_NO,S_NO);
74    P.CHAR_NO := S_NO;
75    P.MESS_ID := CURRENT_MESS_ID;
76    if S_NO = LENGTH_OF_MESS then
77      P.ENDE := true;
78    else P.ENDE := false;
79    endif;
80    insert P into WINDOW_PACKETS;
81    send COLLECT(P) to CH;
82    S_NO := S_NO + 1;
83    if S_NO = 2 then
84      IDLE_ENABLED := true;
85      setup TRANSMISSION2;
86    endif;
87
88  endoperation
89
90
91  operation ACKNOWLEDGE(int:N,MESS_ID) is
92    if MESS_ID = CURRENT_MESS_ID then
93      if N = LENGTH_OF_MESS + 1 then
94        replace by START_TRANSMISSION;
95      else
96        if LWE < N and N <= LWE + WINDOW then
97          if N + WINDOW - 1 <= LENGTH_OF_MESS then
98            call ACTIVATE_TRANSMIT(N - LWE);
99          else call ACTIVATE_TRANSMIT(LENGTH_OF_MESS -
100                                      (LWE + WINDOW - 1));
101         endif;
102
103         LWE := N;
104         if not IDLE_ENABLED then
105           IDLE_ENABLED := true;
106           setup TRANSMISSION2;
107         endif;
108       else
109         if LWE = N then
110           print("N= " + GENSTRING(N) + " ignored");
111         else print("N = " + GENSTRING(N) + "  out of range");
112         endif;
113       endif;
114     endif;
115   endif;
116 endoperation
```

fig. 19 -continued-



```
117 |  |                                                                    |  *8
118 |  endfacet                                                            |  -8
119 |                                                                      | *2
120 |  initial START_TRANSMISSIO                                           |
121 |                                                                      |
122 endscript;                                                             | -2
123 |                                                                     *1
124 |
125 |
126 type COLLECTOR is
127 script(agent : H; int : WINDOW)                                        | +23
128 |
129 |  var int:CURRENT_MESS_ID:= 1;
130 |  var int:LWE_OLD;
131 |  var agent: CH;
132 |
133 |  facethead COLLECTING;
134 |
135 |  facet STARTING is                                                   | +24
136 |  | public: STARTC;                                                   |
137 |  |                                                                   |
138 |  | operation STARTC(->CH) is                                         |  +25
139 |  | | replace by COLLECTING;                                          |
140 |  | endoperation                                                      |  -25
141 |  endfacet                                                            | -24
142 |                                                                     *23
143 |  facet COLLECTING is                                                 | +26
144 |  | public: COLLECT;                                                  |
145 |  |                                                                   |
146 |  | var bool: FIRST_TIME:=true;                                       |
147 |  | var int: I;                                                       |
148 |  | var string: MESS:= "";                                            |
149 |  | type RECEIVED is record int: S_NO;-->                             |  +27
150 |  |                         bool: REC; string: CHAR; endrecord;       |  -27
151 |  | var RECEIVED : RC;                                                | *26
152 |  | relation R of RECEIVED;                                           |
153 |  | var int: LWE := 1;                                                |
154 |  | var int: LAST;                                                    |
155 |  |                                                                   |
156 |  | operation COLLECT(int:CHAR_NO;string:CHAR;int:MESS_ID;bool:ENDE) is |  +28
157 |  |  port: ACCEPTED;                                                  |
158 |  |  oper: RECEIVE_MESS,ACKNOWLEDGE;                                  |
159 |  |  ------------------------------------------------------------     |
160 |  |  if FIRST_TIME then                                               |  +29
161 |  |  | loop for I in 1..WINDOW do                                     |  +30
162 |  |  |  | RC.S_NO := I;                                               |
163 |  |  |  | RC.REC := false;                                            |
164 |  |  |  | RC.CHAR := "*";                                             |
165 |  |  |  | insert RC into R;                                           |
166 |  |  | endloop;                                                       |  -30
167 |  |  | FIRST_TIME := false;                                           | *29
168 |  |  endif;                                                           | -29
169 |  |                                                                  *28
170 |  |  if MESS_ID = CURRENT_MESS_ID then                                |  +31
171 |  |  | find R(CHAR_NO - LWE + 1) --> RC do                            |  +32
172 |  |  |                                                                |
173 |  |  |     RC.CHAR := CHAR;                                           |
174 |  |  |     if not (CHAR_NO = LWE) then                                |  +33
175 |  |  |     | RC.REC := true;                                          |
176 |  |  |     | insert RC into R;                                        |
```

fig. 19 -continued-

```
177     if ENDE then LAST := CHAR_NO; endif;                                    +34-34
178     send ACKNOWLEDGE(LWE,CURRENT_MESS_ID) to CH;                            *33
179   else  (* CHAR_NO = LWE *)
180     find R(1) --> RC1 do                                                    +35
181     if ENDE then                                                            +36
182       send ACKNOWLEDGE(LWE + 1,CURRENT_MESS_ID) to CH;
183       MESS := MESS + RC1.CHAR;
184       send RECEIVE_MESS(MESS) to H reply to ACCEPTED;
185       CURRENT_MESS_ID := CURRENT_MESS_ID + 1;
186       wait ACCEPTED;
187       LWE_OLD := LWE + 1;
188       replace by COLLECTING;
189       (* EFFECT: new initialization       *)
190       (*         of local variables       *)
191     else begin                                                              +37
192         var int: J ;
193         ----------------------------------------
194         MESS := MESS + RC1.CHAR;
195         loop for J1 in 2..WINDOW until EXITLOOP do                          +38
196           find  R(J1) --> RC do                                             +39
197             if RC.REC then                                                  +40
198               MESS := MESS + RC.CHAR;
199             else signal EXITLOOP;
200             endif;                                                          -40
201           endfind;                                                          -39
202           if J1 = WINDOW then J := WINDOW + 1;endif;                        +41-41
203           exit EXITLOOP is                                                  *38
204             J := J1;
205         endloop;                                                            -38
206         LWE := LWE + J - 1;                                                 *37
207         send ACKNOWLEDGE(LWE,CURRENT_MESS_ID) to CH;
208         if LWE = LAST + 1 then                                              +42
209           send RECEIVE_MESS(MESS) to H
210                reply to ACCEPTED;
211           CURRENT_MESS_ID := CURRENT_MESS_ID + 1;
212           wait ACCEPTED;
213           LWE_OLD := LWE;
214           replace by COLLECTING;
215         else begin                                                          +43
216
217           I := 2;
218           loop while I + J - 1 <= WINDOW do                                 +44
219           find R(I) --> RC2 do                                              +45
220             find R(I + J - 1) --> RC3 do                                    +46
221             delete RC2 in R;
222             RC2.REC := RC3.REC;
223             RC2.CHAR := RC3.CHAR;
224             I := I + 1;
225             insert RC2 into R;
226             endfind;                                                        -46
227           endfind;                                                          -45
228           endloop;                                                          -44
229           loop while I <= WINDOW do                                         +47
230             find R(I) --> RC2 do                                            +48
231               delete RC2 in R;
232               RC2.REC := false;
233               insert RC2 into R;
234             endfind;                                                        -48
235             I:= I + 1;                                                      *47
236           endloop;                                                          -47
```

fig. 19 -continued-

```
                          end;                                          -43
                        endif;                                          -42
                      end;                                              -37
                    endif;                                              -36
                    endfind;                                            -35
                endif;                                                  -33
            otherwise do send ACKNOWLEDGE(LWE,CURRENT_MESS_ID) to CH;   *32
            endfind;                                                    -32
          else send ACKNOWLEDGE(LWE_OLD,CURRENT_MESS_ID - 1) to CH;     *31
          endif;                                                        -31
      endoperation                                                      -28
                                                                        *26
    endfacet                                                            -26
                                                                        *23
    initial  STARTING;

endscript;                                                              -23
                                                                        *1

type HOST is
script                                                                  +49

    var agent : TRM;

    facethead WORKING;

    facet INITH is                                                      +50

      public : STARTH;

      operation STARTH(->TRM) is                                        +51
              replace by WORKING;
      endoperation                                                      -51
                                                                        *50
    endfacet                                                            -50
                                                                        *49

    facet WORKING is                                                    +52

      public : TRANSMIT,RECEIVE_MESS,NEXT_MESS;
      private : TWIST_AND_SEND;

      var int : I,J;
      var bool : READY_TO_TRM,READY_TO_TWIST := true;
      var string : M;
      type REC_ELEM is record int : IX; --> string : MESS; endrecord;   +53-53
      relation REC_MESS of REC_ELEM;                                    *52
      var REC_ELEM : RE;

      functionhead LENGTH( string : P1 ) returns int external;

      operation TRANSMIT(->M) assert READY_TO_TRM is                    +54
        if M = "" then                                                  +55
          print(" <empty message>  cannot be transmitted");
        else
              I := 0;
              READY_TO_TRM := false;
              send TWIST_AND_SEND to self;
        endif;                                                          -55
      endoperation;                                                     -54
```

fig. 19 -continued-

```
297     |   |
298     |   |   operation NEXT_MESS is
299     |   |   | READY_TO_TWIST := true;
300     |   |   endoperation;
301     |   |
302     |   |   operation TWIST_AND_SEND assert READY_TO_TWIST is
303     |   |   | oper : SEND_MESS;
304     |   |   | functionhead SUBSTR(string:P1;int:P2;int:P3)
305     |   |   |              returns string external;
306     |   |   | ----------------------------------------
307     |   |   | READY_TO_TWIST := false;
308     |   |   | I := I + 1;
309     |   |   | send SEND_MESS(M) to TRM;
310     |   |   |
311     |   |   | if I < LENGTH(M) then
312     |   |   | | M := SUBSTR(M,2,LENGTH(M)) + SUBSTR(M,1,1);
313     |   |   | | send TWIST_AND_SEND to self;
314     |   |   | else READY_TO_TRM := true;
315     |   |   | endif;
316     |   |   endoperation;
317     |   |
318     |   |   operation RECEIVE_MESS(string : MESS) is
319     |   |   | reply;
320     |   |   | J := J + 1;
321     |   |   | RE.IX := J;
322     |   |   | RE.MESS := MESS;
323     |   |   | insert RE into REC_MESS;
324     |   |   |
325     |   |   | if J >= LENGTH(MESS) then
326     |   |   | | begin port : OK;
327     |   |   | |       oper : ACKN;
328     |   |   | |       functionhead GENSTRING(int : P1) returns string external;
329     |   |   | |       var string : OUT;
330     |   |   | |       -----------------
331     |   |   | |       send ACKN("Host received fully twisted message")
332     |   |   | |            to interface reply to OK;
333     |   |   | |       loop for J in 1..LENGTH(MESS) do
334     |   |   | |       | find REC_MESS(J) -> MS do
335     |   |   | |       | | OUT := GENSTRING(J) + ". message : " + MS.MESS;
336     |   |   | |       | | send ACKN(OUT) to interface;
337     |   |   | |       | endfind;
338     |   |   | |       endloop;
339     |   |   | |
340     |   |   | |       J := 0;
341     |   |   | |       wait OK;
342     |   |   | | end;
343     |   |   | endif;
344     |   |   endoperation;
345     |   |
346     |   endfacet
347     |
348     |   initial INITH;
349     |
350     endscript;
351     |
352     |
353     type FAULTY_CHANNEL is
354     script(agent : C)
355     |
356     var agent : T;
```

```
|*52
| +56
| -56
|*52
| +57
|
|
|
|
|
|
|
| +58
|
|
| -58
|-57
|*52
| +59
|
|
|
|
|
| +60
| +61
|
|
|
|
| +62
| +63
|
|
| -63
| -62
| *61
|
|
|
| -61
|-60
|-59
|*52
|-52
|*49
|
|
|
|-49
|*1
|
|
| +64
|
|
```

fig. 19 -continued-

```
357 var int : UP,GEN_ZUF,ZUF,I;                                                  *64
358 var bool : SHOW;
359 |
360 functionhead RANDOM(int:LOW,UP) returns int external;
361 |
362 facethead TRANSMITTING;
363 |
364 facet INITIALIZATION is                                                      +65
365 | public : STARTCH;
366 |
367 | operation STARTCH(->T;->UP;->GEN_ZUF;->SHOW) is                            +66
368 | | if UP > 10 then UP := 10; endif;                                        +67-67
369 | | loop for I in 1..GEN_ZUF do                                             +68
370 | | | ZUF := RANDOM(1,UP);
371 | | endloop;                                                                -68
372 | | replace by TRANSMITTING;                                               *66
373 | endoperation                                                             -66
374 |                                                                          *65
375 endfacet                                                                   -65
376 |                                                                         *64
377 facet TRANSMITTING is                                                      +69
378 | public : COLLECT,ACKNOWLEDGE,SET_CHNL_PARMS;
379 |
380 | functionhead GENSTRING(int:P1) returns string external;
381 | var int : COUNT := 1;
382 | var string : MESS;
383 |
384 | operation COLLECT(int:P1;string:P2;int: P3;bool:P4) is                     +70
385 | |
386 | | if SHOW then                                                            +71
387 | | | MESS := "(" + GENSTRING(P1) + " , " + P2 + " , "
388 | | |           + GENSTRING(P3) + " , " ;
389 | | | if P4 then MESS := MESS + "T)" ;                                      +72
390 | | | else MESS := MESS + "F)" ;
391 | | | endif;                                                                -72
392 | | endif;                                                                 -71
393 | |                                                                       *70
394 | | if COUNT = 1 then ZUF := RANDOM(1,UP); endif;                           +73-73
395 | | if COUNT < ZUF then                                                     +74
396 | | | send COLLECT(P1,P2,P3,P4) to C;
397 | | | COUNT := COUNT + 1;
398 | | | if SHOW then print("sends COLLECT" + MESS); endif;                     +75-75
399 | | else COUNT := 1;                                                      *74
400 | | |    if SHOW then                                                      +76
401 | | |    | print("forgets COLLECT" + MESS);
402 | | |    endif;                                                           -76
403 | | endif;                                                                -74
404 | |                                                                       *70
405 | endoperation                                                            -70
406 |                                                                         *69
407 | operation ACKNOWLEDGE(int:P1,P2) is                                     +77
408 | |
409 | | if SHOW then                                                            +78
410 | | | MESS := "(" + GENSTRING(P1) + "," + GENSTRING(P2) + ")";
411 | | endif;                                                                -78
412 | |                                                                       *77
413 | | if COUNT = 1 then ZUF := RANDOM(1,UP); endif;                           +79-79
414 | |                                                                       *77
415 | | if COUNT < ZUF then                                                     +80
416 | | | send ACKNOWLEDGE(P1,P2) to T;
```

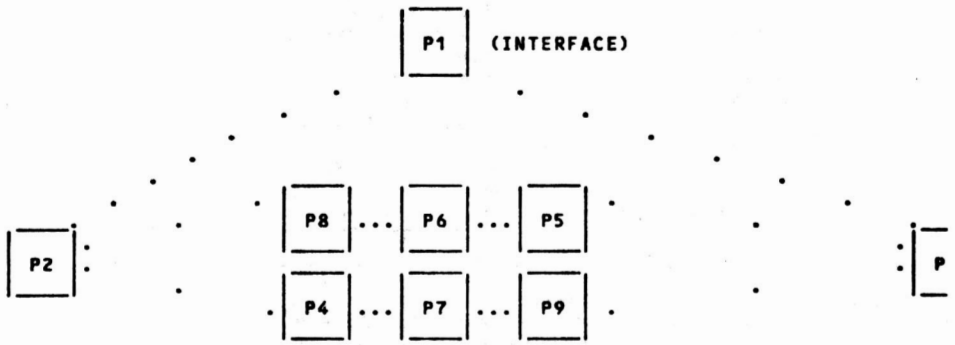
fig. 19 -continued-

```
          | COUNT := COUNT + 1;                                          *80
          | if SHOW then print("sends ACKNOWLEDGE" + MESS); endif;       +81-81
        else COUNT := 1;                                                 *80
          |   if SHOW then                                               +82
          |   | print("forgets ACKNOWLEDGE" + MESS);
          |   endif;                                                     -82
        endif;                                                           -80
                                                                         *77
    endoperation                                                         -77
                                                                         *69
    operation SET_CHNL_PARMS(-> UP,GEN_ZUF,SHOW) assert UP > 0 is        +83

      if UP > 10 then UP := 10; endif;                                   +84-84
      loop for I in 1..GEN_ZUF do                                        +85
      | ZUF := RANDOM(1,UP);
      endloop;                                                           -85
      COUNT := 1;                                                        *83

    endoperation                                                         -83
                                                                         *69
  endfacet                                                               -69
                                                                         *64
  initial  INITIALIZATION;


  endscript;                                                             -64
                                                                         *1


  facet ONLY_FCT is                                                      +86
    public: START_SYSTEM;
    operation START_SYSTEM(int:WINDOW_SIZE) assert WINDOW_SIZE > 0 is    +87

      const HOST : H1 := new HOST;
      const HOST : H2 := new HOST;

      const COLLECTOR : C2 := new COLLECTOR(H1,WINDOW_SIZE);
      const COLLECTOR : C1 := new COLLECTOR(H2,WINDOW_SIZE);

      const FAULTY_CHANNEL : CH1 := new FAULTY_CHANNEL(C1);
      const FAULTY_CHANNEL : CH2 := new FAULTY_CHANNEL(C2);

      const TRANSMITTER : T1 := new TRANSMITTER(CH1,WINDOW_SIZE,H1);
      const TRANSMITTER : T2 := new TRANSMITTER(CH2,WINDOW_SIZE,H2);

      oper: STARTCH,STARTH,STARTC,ACQ_TO_HOST;

      -----------------------------------------------------------

      send STARTCH(T1,UP1,GEN_ZUF1,SHOW1) to CH1;
      send STARTCH(T2,UP2,GEN_ZUF2,SHOW2) to CH2;

      send STARTH(T1) to H1;
      send STARTH(T2) to H2;

      send STARTC(CH1) to C1;
      send STARTC(CH2) to C2;

      send ACQ_TO_HOST(H1,H2,CH1,CH2) to interface;
                                                                         *87

      terminate;

    endoperation                                                         -87
                                                                         *86
  endfacet                                                               -86
                                                                         *1
  initial ONLY_FCT;

  endscript                                                              -1
```

```
BMS-CSSA-COMPILER - DATE OF RELEASE: 30 SEP 1981  NO ERROR DETECTED
END OF COMPILING ON 1982/01/21 AT 14:28:44.00    RETURNCODE = 0
COMPILE-TIME (CPU) =  28.34 SEC.    EXECUTION-TIME =   177.00 SEC.
NUMBER OF SOURCE-LINES READ =  486     NUMBER OF TOKENS =  2171
NUMBER OF OBJECT-RECORDS GENERATED =   5254
```

fig. 19 -continued-

```
C S S A - S I M U L A T I O N - S Y S T E M
```




```
          PROGRAM GENERATED ON 1981/11/17 AT 12:27:21.00
          BY BMS-CSSA-COMPILER (VERS. 30 SEP 1981)

          PROTOCOL OF CSSA SESSION ON 1981/12/04 AT 12:08:57.00

>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> ;DISPLAY                                                          <==
```

IDENTIFIER	TYPE	VALUE
COLLECTOR	SCRIPT	COLLECTOR
FAULTY_CHANNEL	SCRIPT	FAULTY_CHANNEL
HOST	SCRIPT	HOST
START_NETWORK	SCRIPT	START_NETWORK
TRANSMITTER	SCRIPT	TRANSMITTER

```
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : NTW := NEW START_NETWORK(600,10,500,0,TRUE,TRUE)      <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> VAR AGENT : H1,H2,CH1,CH2; OPER : START_SYSTEM,TRANSMIT,SET_CHNL_PARMS  <==
>>>      0.000  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND START_SYSTEM(2) TO NTW; RUN; MAILBOX                        <==
>>>     10.300  SYSTEM TERMINATED
MAILBOX OF INTERFACE(1) :
(1) ACQ_TO_HOST(AGENT:HOST(1),AGENT:HOST(2),AGENT:FAULTY_CHANNEL(1)
                          ,AGENT:FAULTY_CHANNEL(2))

>>>     10.300  INTERFACE(1) : ENTER CSSA COMMAND -
==> RECEIVE 1 (H1,H2,CH1,CH2); STATUS; DISPLAY                        <==
 +++     10.300  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			ACQ_TO_HOST
P3: HOST(1)	WORKING		
P4: HOST(2)	WORKING		
P5: COLLECTOR(1)	COLLECTIN		
P6: COLLECTOR(2)	COLLECTIN		
P7: FAULTY_CHANNE	TRANSMITT		
P8: FAULTY_CHANNE	TRANSMITT		
P9: TRANSMITTER(1	START_TRA		
P2: TRANSMITTER(2	START_TRA		

fig. 20 : Window Mechanism Execution Protocol

IDENTIFIER	TYPE	VALUE
CH1	AGENT	FAULTY_CHANNEL(1)
CH2	AGENT	FAULTY_CHANNEL(2)
COLLECTOR	SCRIPT	COLLECTOR
FAULTY_CHANNEL	SCRIPT	FAULTY_CHANNEL
HOST	SCRIPT	HOST
H1	AGENT	HOST(1)
H2	AGENT	HOST(2)
NTW	AGENT	START_NETWORK(1)
SET_CHNL_PARMS	LITERAL	
START_NETWORK	SCRIPT	START_NETWORK
START_SYSTEM	LITERAL	
TRANSMIT	LITERAL	
TRANSMITTER	SCRIPT	TRANSMITTER

```
>>>    10.300  INTERFACE(1) : ENTER CSSA COMMAND -
==> SEND TRANSMIT("ABCD") TO H1; SEND TRANSMIT("XYZ") TO H2 ;RUN                    <==
***    16.700  FAULTY_CHANNEL(2) : forgets COLLECT(1 , X , 1 , F)
***    16.800  FAULTY_CHANNEL(1) : sends COLLECT(1 , A , 1 , F)
***    17.800  FAULTY_CHANNEL(1) : sends COLLECT(2 , B , 1 , F)
***    17.900  FAULTY_CHANNEL(2) : sends COLLECT(2 , Y , 1 , F)
***    18.800  FAULTY_CHANNEL(1) : sends COLLECT(1 , A , 1 , F)
***    18.900  FAULTY_CHANNEL(2) : sends COLLECT(1 , X , 1 , F)
***    21.900  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,1)
***    22.300  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(1,1)
***    23.500  TRANSMITTER(2) : N= 1 ignored
***    24.000  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(3,1)
***    24.500  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,1)
***    25.900  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,1)
***    27.100  FAULTY_CHANNEL(1) : forgets COLLECT(3 , C , 1 , F)
***    28.300  FAULTY_CHANNEL(1) : sends COLLECT(2 , B , 1 , F)
***    28.500  FAULTY_CHANNEL(2) : sends COLLECT(3 , Z , 1 , T)
***    29.300  FAULTY_CHANNEL(1) : sends COLLECT(4 , D , 1 , T)
***    29.500  FAULTY_CHANNEL(2) : sends COLLECT(3 , Z , 1 , T)
***    30.300  FAULTY_CHANNEL(1) : sends COLLECT(3 , C , 1 , F)
***    30.700  TRANSMITTER(1) : N= 3 ignored
***    32.000  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(4,1)
***    32.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,1)
***    33.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,1)
***    34.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(5,1)
***    34.800  TRANSMITTER(1) : N= 3 ignored
***    35.800  TRANSMITTER(1) : N= 3 ignored
***    36.600  FAULTY_CHANNEL(2) : forgets COLLECT(1 , Y , 2 , F)
***    37.900  FAULTY_CHANNEL(2) : sends COLLECT(2 , Z , 2 , F)
***    39.800  FAULTY_CHANNEL(2) : sends COLLECT(1 , Y , 2 , F)
***    40.300  FAULTY_CHANNEL(1) : sends COLLECT(1 , B , 2 , F)
***    41.400  FAULTY_CHANNEL(1) : sends COLLECT(2 , C , 2 , F)
***    42.300  FAULTY_CHANNEL(1) : forgets COLLECT(1 , B , 2 , F)
***    44.900  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(4,1)
***    45.900  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(1,2)
***    47.100  TRANSMITTER(2) : N= 1 ignored
***    47.500  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(3,2)
***    48.300  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,2)
***    50.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,2)
***    52.000  FAULTY_CHANNEL(2) : sends COLLECT(3 , X , 2 , T)
***    53.000  FAULTY_CHANNEL(2) : sends COLLECT(3 , X , 2 , T)
***    53.500  FAULTY_CHANNEL(1) : sends COLLECT(2 , C , 2 , F)
***    54.500  FAULTY_CHANNEL(1) : sends COLLECT(3 , D , 2 , F)
***    55.500  FAULTY_CHANNEL(1) : sends COLLECT(4 , A , 2 , T)
***    55.500  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(4,2)
***    56.400  FAULTY_CHANNEL(1) : forgets COLLECT(3 , D , 2 , F)
***    57.300  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,2)
***    58.500  TRANSMITTER(1) : N= 3 ignored
***    59.000  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(4,2)
***    60.100  FAULTY_CHANNEL(2) : forgets COLLECT(1 , Z , 3 , F)
***    61.000  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(5,2)
***    61.300  FAULTY_CHANNEL(2) : forgets COLLECT(2 , X , 3 , F)
***    62.400  FAULTY_CHANNEL(2) : sends COLLECT(1 , Z , 3 , F)
***    63.700  FAULTY_CHANNEL(1) : sends COLLECT(4 , A , 2 , T)
***    65.700  FAULTY_CHANNEL(1) : sends COLLECT(1 , C , 3 , F)
***    66.800  FAULTY_CHANNEL(1) : sends COLLECT(2 , D , 3 , F)
***    67.700  FAULTY_CHANNEL(1) : forgets COLLECT(1 , C , 3 , F)
```

fig. 20 -continued-

```
***     68.400  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(4,2)
***     70.000  FAULTY_CHANNEL(2) : sends ACKNOWLEDGE(2,3)
***     72.200  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(5,2)
***     73.700  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,3)
***     74.400  FAULTY_CHANNEL(2) : forgets COLLECT(3 , Y , 3 , T)
***     75.600  FAULTY_CHANNEL(2) : sends COLLECT(2 , X , 3 , F)
***     76.300  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,3)
***     79.000  FAULTY_CHANNEL(1) : sends COLLECT(2 , D , 3 , F)
***     79.600  FAULTY_CHANNEL(2) : forgets ACKNOWLEDGE(3,3)
***     80.000  FAULTY_CHANNEL(1) : sends COLLECT(3 , A , 3 , F)
***     80.900  FAULTY_CHANNEL(1) : forgets COLLECT(4 , B , 3 , T)
***     82.100  FAULTY_CHANNEL(1) : sends COLLECT(3 , A , 3 , F)
***     82.900  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(3,3)
***     84.100  TRANSMITTER(1) : N= 3 ignored
***     84.600  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(4,3)
***     85.600  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(4,3)
***     87.200  TRANSMITTER(1) : N= 4 ignored
***     88.500  FAULTY_CHANNEL(1) : sends COLLECT(4 , B , 3 , T)
***     91.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(5,3)
***     96.500  FAULTY_CHANNEL(1) : sends COLLECT(1 , D , 4 , F)
***     97.500  FAULTY_CHANNEL(1) : forgets COLLECT(2 , A , 4 , F)
***     99.700  FAULTY_CHANNEL(1) : sends COLLECT(1 , D , 4 , F)
***    103.200  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,4)
***    104.600  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,4)
***    106.400  TRANSMITTER(1) : N= 2 ignored
***    107.700  FAULTY_CHANNEL(1) : sends COLLECT(3 , B , 4 , F)
***    108.700  FAULTY_CHANNEL(1) : sends COLLECT(2 , A , 4 , F)
***    111.200  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(2,4)
***    112.400  TRANSMITTER(1) : N= 2 ignored
***    112.900  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(4,4)
***    117.400  FAULTY_CHANNEL(1) : sends COLLECT(4 , C , 4 , T)
***    118.300  FAULTY_CHANNEL(1) : forgets COLLECT(4 , C , 4 , T)
***    120.800  FAULTY_CHANNEL(1) : sends ACKNOWLEDGE(5,4)
>>>    127.400  SYSTEM TERMINATED
>>>    127.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> MAILBOX                                                          <==
MAILBOX OF INTERFACE(1) :
-------------------------------------------------------------------------
(1) ACQ_TO_HOST(AGENT:HOST(1),AGENT:HOST(2),AGENT:FAULTY_CHANNEL(1)
                          ,AGENT:FAULTY_CHANNEL(2))
(2) ACKN(STRING:"Host received fully twisted message"),REPLY TO: OK
(3) ACKN(STRING:"3. message : CDAB")
(4) ACKN(STRING:"4. message : DABC")
(5) ACKN(STRING:"2. message : BCDA")
(6) ACKN(STRING:"1. message : ABCD")
-------------------------------------------------------------------------

>>>    127.400  INTERFACE(1) : ENTER CSSA COMMAND -
==> REPLY 2; SEND TRANSMIT("A") TO H1;RUN                             <==
***    133.600  FAULTY_CHANNEL(1) : forgets COLLECT(1 , A , 5 , T)
***    134.700  FAULTY_CHANNEL(1) : forgets COLLECT(1 , A , 5 , T)
>>>    134.700  SYSTEM TERMINATED
>>>    134.700  INTERFACE(1) : ENTER CSSA COMMAND -
==> TERMINATE                                                        <==
 +++    134.700  ALL EXISTING AGENTS:
```

AGENT	FACET	OPERATION	MAILBOX
P1: INTERFACE(1) *			ACQ_TO_HOST ACKN ACKN ACKN ACKN ACKN
P3: HOST(1)	WORKING		
P4: HOST(2)	WORKING		
P5: COLLECTOR(1)	COLLECTIN		
P6: COLLECTOR(2)	COLLECTIN		
P7: FAULTY_CHANNE	TRANSMITT		
P8: FAULTY_CHANNE	TRANSMITT		
P9: TRANSMITTER(1	TRANSMISS		
P2: TRANSMITTER(2	TRANSMISS		

```
                    CSSA-SESSION-STATISTICS
                    =======================


SESSION  STARTED  AT  12:08:27.00
SESSION TERMINATED AT 12:18:17.00  ON 1981/12/04
REAL-TIME USED :  592.00 SEC.
CPU-TIME  USED :   12.27 SEC.
SIMULATION TIME USED :    134.7000 SEC.
NUMBER OF AGENTS CREATED :    9
NUMBER OF MESSAGES SENT :  224
```

fig. 20 -continued-