TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A formal approach to distributed information systems

Citation for published version (APA):
Houben, G. J. P. M., & Paredaens, J. (1987). *A formal approach to distributed information systems*. (Computing science notes; Vol. 8703). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1987

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

A FORMAL APPROACH TO DISTRIBUTED INFORMATION SYSTEMS

GEERT-JAN HOUBEN
JAN PAREDAENS

87/03

February 1987

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
editor: F.A.J. van Neerven

A FORMAL APPROACH TO DISTRIBUTED INFORMATION SYSTEMS

GEERT-JAN HOUBEN

Department Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, Holland

JAN PAREDAENS

Department Mathematics-Computer Science
University of Antwerp UIA
Universiteitsplein 1, 2610 Antwerpen, Belgium

We describe a mathematical model for distributed information systems, which have a distributed scheduler and in which each site acts as an automaton-like process.
We consider information systems, which contain functional distributed database systems. In a distributed database system the information is stored in several sites. Each site is managed by a machine. A machine is able to execute actions at its site. An action represents some manipulation of the information stored in the site. As such, each machine is responsible for the information in its local database component.
An information system executes transactions. Transactions have to be translated into sequences of actions, that are executed by the machines. This translation is performed by transaction handlers. Therefore, transaction handlers also belong to the system. They get the responsibility for the execution of transactions. Therefore, the transaction handlers have to communicate with several machines. A transaction, which is being executed by the system, should have a consistent view of the information stored in the database. In order to satisfy this condition, the transaction handlers have to operate according to some schedule.

We will describe the processing of the information system, i.e. the operation of both the transaction handlers and the machines and the interface between them. A number of equivalent communication algorithms (schedules) are presented, that guarantee that the transactions have a consistent view of the database component in a machine. The last schedule to be presented is the schedule that should be used in distributed information systems, because it does not require any central time stamping.

Given this schedule we then present some models for the protocol for the communication between handlers and machines independent of the schedule.
Although in these models the way of communication is organized very differently, we will prove that all of these models are equivalent.
Since these models have different degrees of parallelism, we then define a general model, which implies a whole set of models of distributed information systems. Then we argue that one of the models in the set (i.e. the MBIS model) is the most parallel model, which essentially means that handling transactions in parallel is organized in the best possible way.

1 DISTRIBUTED INFORMATION SYSTEMS

1.0 What are distributed information systems ?

In a distributed database system the information is stored in several sites. Each site is managed by a machine. A machine is able to execute actions at its site. An action represents some manipulation of the information stored in the site. As such, each machine is responsible for the information in its local database component.
An information system executes transactions. Transactions have to be translated into sequences of actions, that are executed by the machines. This translation is performed by transaction handlers. Therefore, transaction handlers also belong to the system. They get the responsibility for the execution of transactions. Therefore, the transaction handlers have to communicate with several machines. A transaction, which is being executed by the system, should have a consistent view of the information stored in the database. In order to satisfy this condition, the transaction handlers have to operate according to some schedule.

In the models for distributed information systems we will describe the processing of the information system, i.e. the operation of both the transaction handlers and the machines and the interface between them. We also present a communication schedule, that guarantees that the transactions have a consistent view of the database component in a machine.
Transaction handlers have the responsibility for the execution of at most one transaction at a time. From a transaction a handler produces actions, with which machines are supplied (we say, the handler sends the actions to machines). A machine is able to execute an action autonomously. As a result of such an execution the machine supplies the transaction handler with a reaction (we say, the machine sends a reaction to the handler). When handling a transaction, a transaction handler can only handle a second transaction after the execution of the first one is completed, which implies that for all actions implied by the transaction the handler has received a reaction from a machine. We suppose that every handler always knows both which transaction it has to handle and for every action which machine is able to execute that action.
Machines can execute actions, since they are able to make computations and to store and update information. In order to guarantee a consistent view of the database efficiently, we distinguish three kinds of actions. With a functional action a machine produces a result, that is not dependent on the information stored in the local database component. With a view action, the result is dependent on the information stored in the local database component. With an update action a machine modifies, dependent on the local database component, the information stored in that local database component. The machine will supply the handler, that sent the action to the machine, with a reaction containing the result of the execution in case of a functional or a view action and containing information concerning the validity of the update in case of an update action. Of course, a reaction can have an effect on the handling of the remainder of the transaction.

A distributed information system is a 3-tuple (H, M, s), where H is a non-empty set of transaction handlers, M is a non-empty set of machines and s is a scheduler.
At any moment, each transaction handler h of H is in one of three modes, depending on what it is doing. If h is not handling a transaction at all, it is in the mode **A** (we say, h is asleep). If h is gathering information needed for the transaction, i.e. it is sending functional and view actions to machines, then h is in the mode **FV**. If h is updating information, as a consequence of the transaction, then h is in the mode **U**. While h is in **FV** it cannot send update actions, while h is in **U** it cannot send functional nor view actions and while h is asleep it cannot send any action at all. We say that h is active, if h is in **FV** or in **U**.

Each machine m of M executes at most one action at a time and after the execution of an action it sends a reaction to the transaction handler that sent that action. If the action was a functional or a view action, then the reaction will contain the result (answer) of the action and if it was an update action, then the reaction contains information concerning the validity of that update.

The scheduler s is responsible for the interface between the elements of H and those of M. Therefore, its main task is to execute a schedule, that satisfies the following three conditions :

it controls the mode transition of each h of H to be **A** → **FV** → **U** → **A**;
it allows the information flow between each handler and each machine to be as described;
it is serializable.

We now present a small example, which should illustrate some of the notions mentioned above.

Suppose two handlers, h_1 and h_2, and four machines, m_1, m_2, m_3 and m_4, belong to our information system. In the machine m_1 we have the addresses and ages of employees. In m_2 are the medical records of employees, and the salaries are in m_3. Machine m_4 is very good at computing the square of a natural number.

In some informal way we now describe three transactions.
Let t_1 be :
get for every employee his age;
consider only the employees older than 60;
get for these employees their medical record.
When t_1 is handled by h_1, then h_1 has to send a view action to m_1 first. Machine m_1 will send a reaction to h_1 and a part of the information contained in this reaction will be used by h_1 to initiate a view action at m_2. When m_2 has sent a reaction to h_1, then h_1 has all the information as required by t_1.

Let t_2 be :
get for every employee his address;
consider only the employees living in Eindhoven;
add for these employees 1000 to their salary.
When t_2 is handled by h_2, then h_2 has to send a view action to m_1 first. Subsequently, h_2 will get a reaction from m_1 and depending on this reaction it will send an update action to m_3. After m_3 has sent a reaction to h_2, h_2 has done everything required by t_2.

Let t_3 be :
compute $c = 7^2$;
add c to every salary.
When h_1 handles t_3, it first sends a functional action to m_4. The reaction of m_4 will be used to initiate an update action at m_3. The work required by t_3 will be completed after a reaction from m_3 is received by h_1.

1.1 A serializable schedule

The schedule for a distributed information system should be serializable. What are serializable schedules ?
First we define serial schedules. When a serial schedule is executed no two transaction handlers are active at the same moment. Hence with a serial schedule there is a function $h : \mathbb{N} \rightarrow H$ indicating the order in which the transaction handlers of H are active.

Two schedules are equivalent if and only if they both result in the same information transition, which means that for every possible state of the information stored in the machines of M the final state will be the same. A serializable schedule is a schedule that is equivalent to a serial schedule. Of course, the most easy kind of serializable schedules are the serial schedules.

We now specify a schedule TSS (from : time stamp schedule), for which we prove that it is equivalent to the serial schedule TSO (from : time stamp order), in which transactions are handled in the order of their entrance in the system, thus of their time stamp.

Note that we suppose that from a transaction t the sets of machines, which have to execute functional actions, view actions and update actions respectively in order to execute t, can be computed.

SCHEDULE TSS :

Suppose $\underline{h}$ is a transaction handler of H and $\underline{t}$ is a transaction that $\underline{h}$ has to handle. Let $t : \mathbb{N} \rightarrow T$, with T the set of all transactions, indicating the order in which the transactions enter the system.

- When $\underline{h}$ goes, in order to execute $\underline{t}$, from **A** to **FV**, the scheduler s gives a time stamp, say j, which is one higher than the previous one, so $\underline{h} = h(j)$ and $\underline{t} = t(j)$, and s calculates from $t(j)$ F_j, V_j, U_j :
  F_j := { machines to which $h(j)$ will send functional actions in order to execute $t(j)$ }
  V_j := { machines to which $h(j)$ will send view actions in order to execute $t(j)$ }
  U_j := { machines to which $h(j)$ will send update actions in order to execute $t(j)$ }

- Before $h(j)$ sends, in order to execute $t(j)$, a view action to a machine m, $h(j)$ waits until m does not belong to $\bigcup_{i<j} U_i$.

- When $h(j)$ goes, while executing $t(j)$, from **FV** to **U**, then :
  $F_j := \varnothing$
  $V_j := \varnothing$

- Before $h(j)$ sends, in order to execute $t(j)$, an update action to a machine m, $h(j)$ waits until m does not belong to $\bigcup_{i<j} (V_i \cup U_i)$.

- When $h(j)$ goes after executing $t(j)$, from **U** to **A**, then :
  $U_j := \varnothing$

END SCHEDULE TSS

Of course, F_j can be omitted from this schedule.

We will now prove the serializability of TSS in showing the equivalence with the (serial) schedule TSO in which the transaction handlers execute the transactions in order of their time stamp, which means that if $i < j$, then $t(i)$ is executed (by $h(i)$) before $t(j)$ is executed (by $h(j)$), so $h(i)$ is active (with $t(i)$) before $h(j)$ is active (with $t(j)$).

Let m be a machine of M, and let $i < j$ and $t(i)$ the transaction to be handled by $h(i)$ and $t(j)$ the transaction to be handled by $h(j)$.

- When m is in at most one of the sets V_i, V_j, U_i and U_j, then there is no problem, since it is easy to see that executing first $t(i)$ then $t(j)$ would have the same effect (i.e. results in the same state of the information in m).

- Of course, there is no problem either, when m only belongs to both V_i and U_i or only to both V_j and U_j.

- When m only belongs to both V_i and V_j then there is no problem, since $h(i)$ does not change anything in m.

- When m belongs to both V_j and U_i, but not to U_j, then, since m in U_i and $i < j$, $h(j)$ will send a view action to m after $U_i := \varnothing$, so when $h(i)$ is not executing $t(i)$ any more.

- When m belongs to both V_i and U_j, but not to U_i, then, since m in V_i and $i < j$, $h(j)$ will send an update action to m after $V_i := \varnothing$, so when $h(i)$ is not in **FV** any more as far as $t(i)$ is concerned.

- When m belongs to both U_i and U_j, but not to V_j, then, since m in U_i and $i < j$, $h(j)$ will send an update action to m after $U_i := \varnothing$ (V_i is already empty at that moment or m was not in V_i), so when $h(i)$ is not executing $t(i)$ any more.

- When m belongs to V_j, U_i and U_j, then, since m in U_i and $i < j$, $h(j)$ will send a view action to m after $U_i := \varnothing$ (m is not in V_i (any more)), so when $h(i)$ is not executing $t(i)$ any more.

This ends the proof of the serializability.
We therefore have a schedule TSS that fulfills the conditions for the schedule of a distributed information system and in which the time stamps are the key issue.

1.2 A serializable schedule that does not use time stamps

We will now specify a serializable schedule NTSS (from : no time stamps schedule), of which the main advantage will be the absence of time stamps.
Since then at each moment we only have to deal with the transactions being handled in the system at that particular time, we only have to assign to these transactions some unique number.
In the time stamp approach however, all transactions that ever have been handled in the system must have some unique number. Obviously this implies an infinite set of numbers being used. Furthermore, in the time stamp approach there must be some central system, that assigns the time stamps, in order to guarantee the global unicity of the time stamps. In this second approach we do not need such a global clock, so in essence the distributed information system consists only of transaction handlers and machines.

First though, we consider a schedule, called ATSS (from : another time stamp schedule), that also uses time stamps. After proving that this schedule is a schedule for a distributed information system, we will show that in this schedule the time stamps are not really needed and can therefore be omitted, thus obtaining a serializable schedule for a distributed information system that does not make use of time stamps. This schedule will be

called NTSS.

We now specify the schedule ATSS (that uses time stamps) for which we prove that it is a correct schedule for a distributed information system. The serializability of ATSS is proven by showing (indirectly) the equivalence to the schedule TSO where transactions are serially handled in the order of their time stamp.

In ATSS we have that intuitively V_i will be the set of machines which get a view action belonging to $t(i)$, U_i will be the set of machines which get an update action belonging to $t(i)$, $V(m)$ will be the number of transaction handlers that need to view machine m, $U(m)$ will be the number of transaction handlers that need to update machine m, $UV_i(m)$ will be the number of transaction handlers that need to update m before $h(i)$ can view m, $AU_i(m)$ will be the sum of the number of transaction handlers that need to view m before $h(i)$ can update m and the number of transaction handlers that need to update m before $h(i)$ can update m.
Note that functional actions, because of their independence of the state of the machine, can not do any harm as far as the consistency is concerned.

SCHEDULE ATSS :

Initialize V_i and U_i to be $\varnothing$ for all i, and $V(m)$, $U(m)$, $UV_i(m)$, $AU_i(m)$ to be 0 for all i and all m of M.

Suppose $\underline{h}$ is a transaction handler of H and $\underline{t}$ is a transaction $\underline{h}$ has to handle.

- When $\underline{h}$ goes, in order to execute $\underline{t}$, from **A** to **FV**, the scheduler s gives a time stamp, say j, which is one higher than the previous one, so $\underline{h} = h(j)$ and $\underline{t} = t(j)$, and calculates :
  V_j := { machines to which $h(j)$ will send view actions in order to execute $t(j)$ }
  U_j := { machines to which $h(j)$ will send update actions in order to execute $t(j)$ }
  $UV_j(m) := U(m)$ for all m in V_j
  $AU_j(m) := V(m) + U(m)$ for all m in U_j
  $V(m) := V(m) + 1$ for all m in V_j
  $U(m) := U(m) + 1$ for all m in U_j

- Before $h(j)$ sends, in order to execute $t(j)$, a view action to a machine m, $h(j)$ waits until
  $UV_j(m) = 0$.

- When $h(j)$ goes, while executing $t(j)$, from **FV** to **U**, then :
  $AU_i(m) := AU_i(m) - 1$ if $AU_i(m) > 0$, for all $i \neq j$ and all m of V_j
  $V(m) := V(m) - 1$ for all m of V_j
  $V_j := \varnothing$

- Before $h(j)$ sends, in order to execute $t(j)$, an update action to a machine m, $h(j)$ waits until
  $AU_j(m) = 0$.

- Before $h(j)$ goes, after executing $t(j)$, from **U** to **A**, then :
  $UV_i(m) := UV_i(m) - 1$ if $UV_i(m) > 0$, for all $i \neq j$ and all m of U_j
  $AU_i(m) := AU_i(m) - 1$ if $AU_i(m) > 0$, for all $i \neq j$ and all m of U_j
  $U(m) := U(m) - 1$ for all m of U_j

$U_j := \varnothing$

END SCHEDULE ATSS

It is trivial to prove that at each moment $V(m)$ is the number of transaction handlers $h(i)$ with m in V_i, and $U(m)$ is the number of transaction handlers $h(i)$ with m in U_i. $UV_j(m)$ is the number of $h(i)$ with $i < j$ and m in both U_i and V_j. $AU_j(m)$ is the sum of the number of $h(i)$ with $i < j$ and m in both V_i and U_j, and the number of $h(i)$ with $i < j$ and m in both U_i and U_j.

To demonstrate the correctness of this schedule ATSS, we consider the next schedule BTSS (from : binary time stamp schedule), that obviously controls the state transition of each transaction handler to be **A** → **FV** → **U** → **A** and allows the information flow between each transaction handler and each machine to be as required.

We will use mV_iU_j and mU_iU_j, where :
mV_iU_j equals 1, if m belongs to V_i and U_j, and mV_iU_j equals 0 else.
mU_iU_j equals 1, if m belongs to U_i and U_j, and mU_iU_j equals 0 else.

SCHEDULE BTSS :

Initialize V_i and U_i to be empty for all i and mV_iU_j and mU_iU_j to be 0 for all m of M, i and j.

Suppose $\underline{h}$ is a transaction handler of H and $\underline{t}$ is a transaction $\underline{h}$ has to handle.

- When $\underline{h}$ goes, in order to execute $\underline{t}$, from **A** to **FV**, the scheduler s gives a time stamp, say j, which is one higher than the previous one, so $\underline{h} = h(j)$ and $\underline{t} = t(j)$, and s calculates from $t(j)$:
  V_j := { machines to which $h(j)$ sends view actions in order to execute $t(j)$ }
  U_j := { machines to which $h(j)$ sends update actions in order to execute $t(j)$ }
  for all m of V_j
    $mV_jU_i := 1$ for all i with m in U_i and $i < j$
  for all m of U_j
    $mU_jU_i := mU_iU_j := 1$ for all i with m in U_i and $i < j$
    $mV_iU_j := 1$ for all i with m in V_i and $i < j$

- Before $h(j)$ sends, in order to execute $t(j)$, a view action to a machine m, $h(j)$ waits until
  $mV_jU_i = 0$ for all $i < j$.

- When $h(j)$ goes, while executing $t(j)$, from **FV** to **U**, then :
  $mV_jU_i := 0$ for all i and all m of V_j
  $V_j := \varnothing$

- Before $h(j)$ sends, in order to execute $t(j)$, an update action to a machine m, $h(j)$ waits until
  $mV_iU_j = mU_iU_j = 0$ for all $i < j$.

- When $h(j)$ goes after executing $t(j)$, from **U** to **A**, then :
  $mU_jU_i := mV_iU_j := mU_iU_j := 0$ for all i and m of U_j
  $U_j := \varnothing$

END SCHEDULE BTSS

We will now prove the serializability of BTSS and then by showing the equivalence between ATSS and BTSS, we will prove the serializability of ATSS.

We will show that BTSS is equivalent to the schedule TSO in which the transaction handlers execute the transactions in order of their time stamp, that is if $i < j$, then $t(i)$ is executed before $t(j)$, so $h(i)$ is active before $h(j)$.

Let m be a machine of M, and let $i < j$ and let $t(i)$ be the transaction to be handled by $h(i)$ and $t(j)$ the transaction to be handled by $h(j)$.

- When m is in at most one of the sets V_i, V_j, U_i and U_j, then there is no problem, since it is easy to see that executing first $t(i)$ then $t(j)$ would have the same effect.

- Of course, there is no problem either, when m only belongs to both V_i and U_i or only to both V_j and U_j.

- When m only belongs to both V_i and V_j, then there is no problem, since $h(i)$ does not change anything in m.

- When m belongs to both V_j and U_i, but not to U_j, then, since m in U_i and $i < j$, $h(j)$ will send a view action to m after $mV_jU_i := 0$, so when $h(i)$ is not executing $t(i)$ any more.

- When m belongs to both V_i and U_j, but not to U_i, then, since m in V_i and $i < j$, $h(j)$ will send an update action to m after $mV_iU_j := 0$, so when $h(i)$ is not in **FV** any more as far as $t(i)$ is concerned.

- When m belongs to both U_i and U_j, but not to V_j, then, since m in U_i and $i < j$, $h(j)$ will send an update action to m after $mU_iU_j := 0$ (mV_iU_j is (already) 0 at that moment), so when $h(i)$ is not executing $t(i)$ any more.

- When m belongs to V_j, U_i and U_j, then, since m in U_i and $i < j$, $h(j)$ will send a view action to m after $mV_jU_i := 0$ (then $mU_jU_i = 0$ and $mV_iU_j = 0$), so when $h(i)$ is not executing $t(i)$ any more.

So we have proven the serializability of BTSS.

It is rather trivial to prove that the following are invariants :
$mV_iU_j = 1$ iff m in V_i and m in U_j ;
$mU_iU_j = 1$ iff m in U_i and m in U_j.

We will show the equivalence between ATSS and BTSS, by proving Q where Q is :
$$UV_j(m) = \sum_{i<j} mV_jU_i \quad \wedge \quad AU_j(m) = \sum_{i<j} (mV_iU_j + mU_iU_j).$$

It is trivial that Q holds at initialization.

When $h(j)$ goes from **A** to **FV**, Q holds if Q' holds, where Q' stands for :
At the moment where $h(j)$ goes from **A** to **FV**, $V(m)$ is the number of $h(i)$ with $i < j$

and m in V_i, and $U(m)$ is the number of $h(i)$ with $i < j$ and m in U_i.
It is trivial to prove that Q' holds.

With Q it is clear that the conditions for which $h(j)$ has to wait before sending view or update actions, are the same in both schedules.

When $h(j)$ goes from **FV** to **U**, mV_jU_i becomes 0 for all i and m of V_j. mV_jU_i was 1 only if m in V_j and m in U_i and $i > j$. This follows from : mV_jU_i only became 1 if m in both V_j and U_i, and if $i < j$ then mV_jU_i already has become 0, since this was the condition for which $h(j)$ was waiting before sending a view action.
So mV_jU_i changes from 1 to 0 if m in V_j and m in U_i and $i > j$. Therefore for all i and m of V_j, $AU_i(m)$ has to decrease by 1 (if possible), in order to keep Q invariant, since $AU_i(m) = \sum_{l<i} (mV_lU_i + mU_lU_i)$ and in the set of mV_lU_i and mU_lU_i with $l < i$, there is only one mV_lU_i that changes from 1 to 0.

When $h(j)$ goes from **U** to **A**, mV_iU_j, mU_iU_j and mU_jU_i become 0 for all i and m of U_j. mV_iU_j was 1 only if m in V_i and m in U_j and $i > j$. Therefore for all i and m of U_j, $UV_i(m)$ has to decrease by 1 (if possible), in order to keep Q invariant, since $UV_i(m) = \sum_{l<i} mV_iU_l$ and in the set of mV_iU_l with $l < i$ there is only one mV_iU_l that changes from 1 to 0.
$mU_iU_j = mU_jU_i$ was 1 only if m in U_i and m in U_j and $i > j$. Therefore for all i and m of U_j, $AU_i(m)$ has to decrease by 1 (if possible), in order to keep Q invariant, since $AU_i(m) = \sum_{l<i} (mV_lU_i + mU_lU_i)$ and in the set of mV_lU_i and mU_lU_i with $l < i$ there is only one mU_lU_i that changes from 1 to 0.

Therefore Q holds.

The claim was that from the schedule ATSS, we could derive a schedule NTSS, that, in contrast to ATSS, would certainly not make use of time stamps.

Before specifying NTSS, we will define the following :

There is some "super transaction handler" that assigns to each transaction that enters the system a transaction handler by which the transaction will be handled.
TID is the set of transaction id-numbers. We denote the transaction with id-number i by t_i.
HID is the set of transaction handler id-numbers. We denote the transaction handler with id-number i by h_i.
$\theta : TID \rightarrow HID$ assigns to each transaction id-number the id-number of the transaction handler that will handle the transaction with that id-number.
In this information system $H = \{h_1,...,h_k\}$ and $HID = \{1,...,k\}$.

Now we will specify NTSS.

SCHEDULE NTSS :

Initialize V_i and U_i to be $\varnothing$ for all i of *HID*, and $V(m)$, $U(m)$, $UV_i(m)$ and $AU_i(m)$ to be 0 for all i of *HID* and all m of M.

Suppose h_j has to handle t_l, so $j = \theta(l)$.

- When h_j goes in order to execute t_l from **A** to **FV**, then s calculates from t_l :
  V_j := { machines to which h_j sends view actions in order to execute t_l ($j = \theta(l)$) }
  U_j := { machines to which h_j sends update actions in order to execute t_l ($j = \theta(l)$) }
  $UV_j(m) := U(m)$ for all m in V_j
  $AU_j(m) := V(m) + U(m)$ for all m in U_j
  $V(m) := V(m) + 1$ for all m in V_j
  $U(m) := U(m) + 1$ for all m in U_j

- Before h_j sends, in order to execute t_l, a view action to a machine m, h_j waits until $UV_j(m) = 0$.

- When h_j goes, while executing t_l, from **FV** to **U**, then :
  $AU_i(m) := AU_i(m) - 1$ if $AU_i(m) > 0$, for all i of $HID - \{j\}$ and m of V_j
  $V(m) := V(m) - 1$ for all m of V_j
  $V_j := \varnothing$

- Before h_j sends, in order to execute t_l, an update action to a machine m, h_j waits until $AU_j(m) = 0$.

- When h_j goes, after executing t_l, from **U** to **A**, then :
  $UV_i(m) := UV_i(m) - 1$ if $UV_i(m) > 0$, for all i of $HID - \{j\}$ and m of U_j
  $AU_i(m) := AU_i(m) - 1$ if $AU_i(m) > 0$, for all i of $HID - \{j\}$ and m of U_j
  $U(m) := U(m) - 1$ for all m of U_j
  $U_j := \varnothing$

END SCHEDULE NTSS

We now claim that ATSS and NTSS, as we just specified, are in fact the same schedule, since the only difference between them is the fact that where in ATSS time stamps are mentioned, in NTSS transaction (handler) id-numbers are mentioned. And when we observe ATSS, we can see that in ATSS we did not use any aspect of the time stamps other than the identification of transactions and transaction handlers. So we can replace the time stamps by id-numbers. Therefore ATSS and NTSS are quite the same schedule.
So NTSS is a serializable schedule for a distributed information system that does not use time stamps.

When we want to use this schedule NTSS, we have to assume that the mode transitions happen exclusively, since mode transitions imply manipulating common variables (e.g. $AU_i(m)$ and $UV_i(m)$). Of course, the testing before sending actions should also happen in an exclusive way, because of the use of common variables (e.g. $UV_j(m)$).
Therefore, we could decide to store all this control information (i.e. the common variables implied by the schedule) in one machine and we could access that information by the sending of control actions (i.e. the update and view actions as implied by the schedule). It is trivial that these control actions can not obey the rules of the scheduler (e.g. before every (real) update action a view (control) action is needed to the special machine containing the common information).

2 SOME MODELS FOR DISTRIBUTED INFORMATION SYSTEMS

2.0 Introduction

From now on, we will assume that a distributed information system operates according to the NTSS schedule. We did not yet specify for a distributed information system (operating according to NTSS), how the transaction handlers and the machines communicate with each other. We now will specify some models of distributed information systems, where for every model we describe in which specific way the communication inside the system is organized. So, when describing some model, we will focus on the communication and not on the schedule any more.

We will see later on, that each of the models that we will present here is a special case of the general model GDIS. Then we want to find the most parallel model, that is a special case of GDIS. However, to be able to find this most parallel model, we must study the way of communication in the GDIS model. The organization of the communication can be influenced by a number of aspects. In almost each of the models, that we present here, one of these aspects is treated in some special way and therefore one can easily learn how this aspect should be treated in the most parallel way.

We will introduce first of all a model called SBIS (from : single buffer information system). The other models will then be described starting from the description of the SBIS model. The models we will describe are :
- single buffer information system (SBIS)
- bounded buffer information system (BBIS)
- sequential information system (SeqIS)
- simple information system (SimIS)
- multiple buffer information system (MBIS)

Some notational definitions

Let X be a set, with $\sigma \notin X$. Then, X^* is the set defined by :
$\sigma \in X^*$;
$a \in X \wedge b \in X^* \Rightarrow a|b \in X^*$;
$a \in X^* \Rightarrow \sigma|a = a \wedge a|\sigma = a$;
$a \in X^* \wedge b \in X^* \wedge c \in X^* \Rightarrow (a|b)|c = a|(b|c)$.
First, tail and size are defined by :
if $x = a|b$, where $a \in X$ and $b \in X^*$,
then $\text{first}(x) = a$ and $\text{tail}(x) = b$ and $\text{size}(x) = 1 + \text{size}(b)$;
$\text{size}(\sigma) = 0$.
$X^+ = X^* - \{\sigma\}$.
So, $|$ denotes the concatenation and σ denotes the empty sequence. First determines the first element of a sequence, tail determines the sequence obtained by removing the first element of a sequence and size determines the number of elements of a sequence (here an element is an element of X, i.e. a sequence of size 1).

η is the empty transaction, which is the transaction for which nothing has to be done.
Note that we suppose that every part of a transaction is a transaction. So when we are handling a transaction t and we know that action a can now be sent to a machine, we then get a new transaction t', which in essence is t without a.

When we write **and** in the description of the communication, we mean the conditional conjunction.

$\mathbb{P}(X)$ denotes the power set of the set X.

If X is a non-empty set, then the value of pick(X) is arbitrarily one of the elements of X.

We also need first and tail for sequences of pairs, where first is defined in such a way that it determines the first element of the sequence with some specific second component (tail is defined analogous).
Let $x \in (A \times B)^+$, so $x = (a,b)|y$, where $a \in A$, $b \in B$ and $y \in (A \times B)^*$.
Then, for $c \in B$:
first$(x,b) = (a,b)$ and tail$(x,b) = y$;
first(x,c) = first(y,c) and tail$(x,c) = (a,b)|$tail(y,c), if $c \neq b$;
first$(\sigma,c) = \sqcup$.
And, for $C \subseteq B$:
first-set$(x,C) = (a,b)$ and tail-set$(x,C) = y$, if $b \in C$;
first-set(x,C) = first-set(y,C) and tail-set$(x,C) = (a,b)|$tail-set(y,C), if $b \notin C$;
first-set$(\sigma,C) = \sqcup$.

2.1 Single Buffer Information System

In the single buffer information system, which we will abbreviate SBIS, there is one single buffer for every handler or machine in the system. We will use the SBIS as the basis for all other models, so the other models are models that in some way or another have some special characteristics, that differ them from the SBIS.
Note that we say that some distributed information system is an SBIS in stead of saying that the system is a system operating according to the SBIS model. Similarly for the other models.

We will now describe an SBIS.

H is a set of transaction handlers.
M is a set of machines.

There is a function $id : H \cup M \rightarrow ID$, where :
id is a one-to-one function,
$HID \subset ID$ and $\mathbf{A}\, h : h \in H : id(h) \in HID$,
$MID \subset ID$ and $\mathbf{A}\, m : m \in M : id(m) \in MID$.
So id assigns to each handler and each machine some unique identification. HID is the set of identifications of all handlers, MID is the set of identifications of all machines.

Let h be a handler, so $h \in H$, with $id(h) = i$, say.
Then $h = (S,T,A,R,O,OP,AP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times MID \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times MID \times T \times S)$, is an action producer.

Informally :
S is the set of states in which h can be, T is the set of transactions h is able to handle, A is the set of actions which h can send to machines, R is the set of reactions which h can receive from machines, O is the set of results which h can produce (as an answer on a transaction), OP is a function that states how h determines for each message from a

machine (containing a reaction) together with a state, a result and a new state. AP is a function that states how h determines given a state together with a transaction, a set, in which each element contains an action, a machine (identification) to which the action can be sent, the transaction that is then left to handle, and a new state.

The configuration of h, denoted by C_i, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times (R \times MID)^*$.
We define : if $C_i = (s, t, o, p, n, r)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$.

Informally :
$C_i \bullet S$ is the state of h, $C_i \bullet T$ is the sequence of transactions that are waiting to be handled by h, $C_i \bullet O$ is the sequence of results that h has produced so far, $C_i \bullet P$ is the part of transaction t that is left to execute, where t is the transaction currently being handled, $C_i \bullet N$ is the number of actions that h has sent to machines, but for which no reaction is yet received, $C_i \bullet R$ is the buffer in which the reaction from the machines are kept until they can be accepted.

Let m be a machine, so $m \in M$, with $id(m) = j$, say.
Then $m = (S, A, R, E)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \rightarrow R \times S$, is an executor.

Informally :
S is the set of actions in which m can be, A is the set of actions which m can execute, R is the set of reactions which m can produce, E states how m computes given an action together with a state, a reaction and a new state.

The configuration of m, denoted by C_j, will be an element of
$S \times (A \times HID)^*$.
We define : if $C_j = (s, a)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$.

Informally :
$C_j \bullet S$ is the state in which m is and $C_j \bullet A$ is the buffer in which the actions from handlers are kept until they can be executed.

With these definitions we have described what handlers and machines are, so now we have to define how they communicate with each other.

We can describe how h operates, by stating that handler h (with $id(h) = i$) continuously executes the following :

if $C_i \bullet R \neq \sigma$ **and** first$(C_i \bullet R) = (r, \bar{j})$
→
$(o, s) := OP(r, \bar{j}, C_i \bullet S)$;
$C_i \bullet S := s$; $C_i \bullet O := C_i \bullet O \,|\, o$; $C_i \bullet N := C_i \bullet N - 1$; $C_i \bullet R :=$ tail$(C_i \bullet R)$
[] $C_i \bullet R = \sigma$ **and** $C_i \bullet P \neq \eta$
→
$pa := AP(C_i \bullet P, C_i \bullet S)$;
if $pa \neq \varnothing$
→

$(a, \bar{j}, p, s) := \text{pick}(pa)$;
$C_i \bullet S := s$; $C_i \bullet P := p$; $C_i \bullet N := C_i \bullet N + 1$; $C_{\bar{j}} \bullet A := C_{\bar{j}} \bullet A \,|\, (a, i)$
[] $pa = \varnothing$
→
skip
fi
[] $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N = 0$ **and** first$(C_i \bullet T) = p$
→
$C_i \bullet T := \text{tail}(C_i \bullet T)$; $C_i \bullet P := p$
[] $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N \neq 0$
→
skip
fi

Initially the following should hold : $C_i \bullet N = 0$.

Machine m (with $id(m) = j$) continuously executes the following :

if $C_j \bullet A \neq \sigma$ **and** first$(C_j \bullet A) = (a, \bar{i})$
→
$(r, s) := E(a, C_j \bullet S)$;
$C_j \bullet S := s$; $C_j \bullet A := \text{tail}(C_j \bullet A)$; $C_{\bar{i}} \bullet R := C_{\bar{i}} \bullet R \,|\, (r, j)$
[] $C_j \bullet A = \sigma$
→
skip
fi

Informally handler h operates as follows :

If there are reactions in its buffer, then it takes the first message in the buffer and produces from it (with OP) a result and a new state. This implies that in its configuration some adjustments need to be made : there is a new state, the result produced must be added to the sequence of results produced, the number of reactions that have to be accepted can decrease by one and from the buffer the first element has been taken away.

If there are no reactions in its buffer, the handler is allowed to send actions to machines. So AP suggests actions that could be sent. If there are no actions that can be sent, because h is waiting for some reaction, then nothing is done. Otherwise one of the actions is sent to the machine that is specified. Again, this implies that there is a new state, there is less work to be done on the transaction currently being handled, the number of reactions that h has to accept increases by one and the message (containing the action and its own identification) is put in the buffer of the specified machine.

If there are no reactions in the buffer and there are no reactions expected any more, then the first element of the sequence of waiting transactions will become the transaction currently being handled.

If h is only waiting for some reactions, but they have not yet been received, then it will do nothing.

Machine m simply takes the first action in its buffer (if there is one), and executes it (with the aid of E), which means that from the action and the current state, a reaction and a new state are computed. This implies that there is a new state, the first action is taken from its buffer and the reaction (together with its identification) is put in the buffer of the

handler that sent the action.

We say that a handler or a machine performs a step, if it executes the if-statement that we just described for both handlers and machines. So every handler and every machine keeps on performing steps.
The only thing that has to happen exclusively is the manipulation of buffers, since buffers only are accessed by more than one object. This implies that whenever h performs something like " $C_j \bullet A := C_j \bullet A \mid (a, i)$ " with j the identification of some machine, then this should happen exclusively. A similar remark holds for the execution by m of " $C_i \bullet R := C_i \bullet R \mid (r, j)$ ".
Therefore, the communication between transaction handlers and machines independent of NTSS, is reduced to the execution of steps by handlers and machines. These steps can be performed locally, with the exception of the assignments that denote the adding of a message to a buffer. So, when we see a step as a sequence of assignments, we can then say that steps can happen in parallel, if assignments in which an element is added to a buffer do happen exclusively.

2.2 Bounded Buffer Information System

In a bounded buffer information system, which will be abbreviated by BBIS, the main difference with the SBIS is the fact that all buffers are bounded. This implies that in the definition of a system the size of each buffer must be included. Also, every handler and every machine must check before putting some message in a buffer, whether there is a free place in the buffer for the message.

A BBIS can be described as follows.

H is a set of transaction handlers.
M is a set of machines.

There is a function $id : H \cup M \rightarrow ID$, where :
id is a one-to-one function,
$HID \subset ID$ and $\mathbf{A}\, h : h \in H : id(h) \in HID$,
$MID \subset ID$ and $\mathbf{A}\, m : m \in M : id(m) \in MID$.

There is a function $BS : ID \rightarrow \mathbb{N} - \{0\}$.
BS (from : buffer size) assigns to each handler and each machine a positive natural number that represents the size of the buffer, i.e. the number of messages that can be put into the buffer.

Let h be a handler, so $h \in H$, with $id(h) = i$, say.
Then $h = (S, T, A, R, O, OP, AP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times MID \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times MID \times T \times S)$, is an action producer.

The configuration of h, denoted by C_i, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times (R \times MID)^*$.
We define : if $C_i = (s, t, o, p, n, r)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$.

Let m be a machine, so $m \in M$, with $id(m) = j$, say.
Then $m = (S, A, R, E)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \rightarrow R \times S$, is an executor.

The configuration of m, denoted by C_j, will be an element of
$S \times (A \times HID)^*$.
We define : if $C_j = (s, a)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$.

Handler h (with $id(h) = i$) continuously executes the following :

if $C_i \bullet R \neq \sigma$ **and** first$(C_i \bullet R) = (r, \bar{j})$
→
$(o, s) := OP(r, \bar{j}, C_i \bullet S)$;
$C_i \bullet S := s$; $C_i \bullet O := C_i \bullet O \,|\, o$; $C_i \bullet N := C_i \bullet N - 1$; $C_i \bullet R :=$ tail$(C_i \bullet R)$
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P \neq \eta$
→
$pa := AP(C_i \bullet P, C_i \bullet S)$;
do $pa \neq \varnothing$
→
$(a, \bar{j}, p, s) :=$ pick(pa);
if size$(C_{\bar{j}} \bullet A) < BS(\bar{j})$
→
$C_i \bullet S := s$; $C_i \bullet P := p$; $C_i \bullet N := C_i \bullet N + 1$; $C_{\bar{j}} \bullet A := C_{\bar{j}} \bullet A \,|\, (a, i)$;
$pa := \varnothing$
▯ size$(C_{\bar{j}} \bullet A) =$ size$(\bar{j})$
→
$pa := pa - (a, \bar{j}, p, s)$
fi
od
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N = 0$ **and** first$(C_i \bullet T) = p$
→
$C_i \bullet T :=$ tail$(C_i \bullet T)$; $C_i \bullet P := p$
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N \neq 0$
→
skip
fi

Informally :
If h is allowed to send an action to a machine, then it computes the set pa including the actions that could be sent. Then h checks whether for some action in pa the machine to which the action is supposed to go has a buffer, that is not full yet, so h can put the message in that buffer.
This checking is performed by running through the set pa until a action has been found that can be sent. If all buffers implied by pa are full, then there will be no action sending and in the next step h will possibly check again whether there is a free buffer, i.e. whether an action can be sent.

Machine m (with $id(m) = j$) continuously executes the following :

if $C_j \bullet A \neq \sigma$ **and** first$(C_j \bullet A) = (a, \bar{i})$

```
→
(r ,s ) := E(a ,C_j•S );
if size(C_ī•R ) < BS(ī)
   →
   C_j•S := s ; C_j•A := tail(C_j•A ); C_ī•R := C_ī•R | (r , j )
[] size(C_ī•R ) = BS(ī)
   →
   skip
fi
[] C_j•A = σ
   →
   skip
fi
```

Informally :
If there is an action in the buffer of m, then m will try to execute the first action in its buffer. Executing an action means not only applying the E function, but also putting the result in the buffer of the handler, that put the action in the buffer of m. Therefore, m has to check whether that buffer is not full. If that buffer is indeed full, then m does not do anything, i.e. it cannot (as in fact h does) choose some other message to work with, because machines work first-in-first-out. This first-in-first-out method is chosen, because it easily excludes the danger of starvation. If we had chosen to use no method at all, then the danger of starvation is also excluded (asymptotically), but there would not have been the obligation to take an element from the buffer.

As far as exclusion is concerned, the difference with the SBIS is that here the checking of a buffer's size and the putting into the buffer of some element if the buffer is not full, should happen in the same exclusive event. Of course, we don't want it to happen that in between the checking of a buffer's size and the putting of a message into the buffer, some other object has put something into the (last free place of the) buffer.

2.3 Sequential Information System

In a SeqIS (the abbreviation for sequential information system), the most important feature is that it could be simulated with a single processor system. This means that there is nothing happening in parallel. Everything happens sequentially. When we say that it could be simulated with a single processor system, we mean that always there is only one handler or machine performing a step. So obviously there must be something that controls who is allowed to perform a step. This controlling is in a SeqIS represented by a function called $ACTIVE$, that in fact is a sequence of identifications, where each $i \in ID$ occurs at least once.

We will now describe a SeqIS.

H is a set of transaction handlers.
M is a set of machines.

There is a function $id : H \cup M \rightarrow ID$, where :
id is a one-to-one function,
$HID \subset ID$ and $\mathbf{A}\, h : h \in H : id(h) \in HID$,
$MID \subset ID$ and $\mathbf{A}\, m : m \in M : id(m) \in MID$.

Let h be a handler, so $h \in H$, with $id(h) = i$, say.

Then $h = (S, T, A, R, O, OP, AP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times MID \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times MID \times T \times S)$, is an action producer.

The configuration of h, denoted by C_i, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times (R \times MID)^*$.
We define : if $C_i = (s, t, o, p, n, r)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$.

Let m be a machine, so $m \in M$, with $id(m) = j$, say.
Then $m = (S, A, R, E)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \rightarrow R \times S$, is an executor.

The configuration of m, denoted by C_j, will be an element of
$S \times (A \times HID)^*$.
We define : if $C_j = (s, a)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$.

There is a function $ACTIVE : [0..n-1] \rightarrow ID$, where every identification of ID occurs at least once as value of $ACTIVE$.
We will use q to run through $[0..n-1]$. Therefore $ACTIVE(q)$ will be the identification of the handler or the machine that is currently allowed to perform a step.

Handler h (with $id(h) = i$) continuously executes the following :

```
if ACTIVE(q) = i
   →
   [... the step as in an SBIS ...];
   q := q + 1 (mod n)
[] ACTIVE(q) ≠ i
   →
   skip
fi
```

Machine m (with $id(m) = j$) continuously executes the following :

```
if ACTIVE(q) = j
   →
   [... the step as in an SBIS ...];
   q := q + 1 (mod n)
[] ACTIVE(q) ≠ j
   →
   skip
fi
```

So every object when performing a step, checks whether it is the only object that has the

allowance of performing a real step, which means a step in which as in an SBIS some work on a transaction could be done. If it has the allowance it performs such a step, otherwise it will do nothing at all.

As far as exclusion is concerned, the only additional requirement, compared to the SBIS, is the exclusivity of the update of q, i.e. the objects, that check the value of q at almost the same time as q is updated, should have a consistent view of q.

Of course, at the initialization of the system $q \in [0..n-1]$ should hold.

2.4 Simple Information System

A simple information system (SimIS) is really a special case of an SBIS. In a SimIS the sets H and M consist both of only one element. In fact, this is the only difference with an SBIS in general. But, since in an SBIS, where both H and M are singletons, many things are redundant, we will now describe a SimIS as we would do it without the description of an SBIS.

A SimIS could be described by describing the transaction handler and the machine, that together with the scheduler belong to the system.

Handler $h = (S, T, A, R, O, OP, AP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times T \times S)$, is an action producer.

The configuration of h, denoted by C_h, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times R^*$.
We define : if $C_i = (s, t, o, p, n, r)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$.

Machine $m = (S, A, R, E)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \rightarrow R \times S$, is an executor.

The configuration of m, denoted by C_m, will be an element of
$S \times A^*$.
We define : if $C_j = (s, a)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$.

Handler h continuously executes the following :

if $C_h \bullet R \neq \sigma$ **and** first$(C_h \bullet R) = r$
$\rightarrow$
$(o, s) := OP(r, C_h \bullet S)$;
$C_h \bullet S := s$; $C_h \bullet O := C_h \bullet O \,|\, o$; $C_h \bullet N := C_h \bullet N - 1$; $C_h \bullet R :=$ tail$(C_h \bullet R)$
$\square$ $C_h \bullet R = \sigma$ **and** $C_h \bullet P \neq \eta$

→
$pa := AP(C_h \bullet P, C_h \bullet S)$;
if $pa \neq \varnothing$
→
$(a, p, s) :=$ pick(pa);
$C_h \bullet S := s$; $C_h \bullet P := p$; $C_h \bullet N := C_h \bullet N + 1$; $C_m \bullet A := C_m \bullet A \mid a$
▯ $pa = \varnothing$
→
skip
fi
▯ $C_h \bullet R = \sigma$ **and** $C_h \bullet P = \eta$ **and** $C_h \bullet N = 0$ **and** first$(C_h \bullet T) = p$
→
$C_h \bullet T :=$ tail$(C_h \bullet T)$; $C_h \bullet P := p$
▯ $C_h \bullet R = \sigma$ **and** $C_h \bullet P = \eta$ **and** $C_h \bullet N \neq 0$
→
skip
fi

Informally :
The difference with the step in an SBIS is that we have messages only containing an action or a reaction, but no identification, that has to identify which object has sent that action or reaction.

Machine m continuously executes the following :

if $C_m \bullet A \neq \sigma$ **and** first$(C_m \bullet A) = a$
→
$(r, s) := E(a, C_m \bullet S)$;
$C_m \bullet S := s$; $C_m \bullet A :=$ tail$(C_m \bullet A)$; $C_h \bullet R := C_h \bullet R \mid r$
▯ $C_j \bullet A = \sigma$
→
skip
fi

Note that in a SimIS we could also leave out the third condition for the scheduler (the serializability condition), since in a system where there is only one handler and one machine this condition is satisfied in a trivial way.

2.5 Multiple Buffer Information System

When we call the messages, that handlers send to machines or machines send to handlers, communication messages, then in an SBIS every handler and every machine has one buffer for communication messages and it takes the messages in a first-in-first-out manner out of that buffer. An MBIS (multiple buffer information system) has exactly one buffer for each object that can put a message in its buffer. So every handler has as many buffers as there are machines, and every machine has as many buffers as there are handlers. However we will represent this by having only one buffer that can be accessed in a first-in-first-out manner per sender. So first of all we choose of which sender we possibly want a message. Then we take from that sender the first message (the message that was put first in the buffer by this sender) if that exists.

An MBIS can be described as follows.

H is a set of transaction handlers.
M is a set of machines.

There is a function $id : H \cup M \rightarrow ID$, where :
id is a one-to-one function.
$HID \subset ID$ and $\mathbf{A}\, h : h \in H : id(h) \in HID$.
$MID \subset ID$ and $\mathbf{A}\, m : m \in M : id(m) \in MID$.

Define $W = \{wait, nowait\}$.

Let h be a handler, so $h \in H$, with $id(h) = i$, say.
Then $h = (S, T, A, R, O, OP, AP, BP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times MID \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times MID \times T \times S)$, is an action producer,
- $BP : S \times MID \times W \rightarrow MID \times W$, is a buffer priority rule.

The configuration of h, denoted by C_i, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times (R \times MID)^* \times MID \times W$.
We define : if $C_i = (s, t, o, p, n, r, b, w)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$, $C_i \bullet B = b$, $C_i \bullet W = w$.

Informally :
The function BP determines, given the current state and information about which sender was chosen and with which priority the last time we could take a message from the buffer, which sender is chosen and with which priority.
If a sender is chosen with priority *wait*, then we take the first message from that sender if there is such a message. Otherwise we do not do anything, i.e. we do definitely not choose another sender, which means that we will wait until that sender has sent a message. So we are able to wait for a particular machine to send a reaction independent of the time the machine takes for handling that reaction.
If a sender is chosen with priority *nowait*, then we take the first message if there is such a message, otherwise we choose again (probably an other sender).
The priority must be seen as a part of the state, with which we can control which message is to be handled, i.e. which sender is given some priority. Since in this way the possibility of the starvation of messages (as far as acceptation is concerned) is introduced, some additional assumptions are needed as we will see at the end of this description of the MBIS model.

In the configuration $C_i \bullet B$ determines the buffer (= sender) from which a message should be taken if we want to do so. $C_i \bullet W$ determines with which priority we want to do so.

Let m be a machine, so $m \in M$, with $id(m) = j$, say.
Then $m = (S, A, R, E, BP)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \rightarrow R \times S$, is an executor,
- $BP : S \times HID \times W \rightarrow HID \times W$, is a buffer priority rule.

The configuration of m, denoted by C_j, will be an element of
$S \times (A \times HID)^* \times HID \times W$.
We define : if $C_j = (s, a, b, w)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$, $C_j \bullet B = b$, $C_j \bullet W = w$.

Handler h (with $id(h) = i$) continuously executes the following :

if $C_i \bullet R \neq \sigma$ **and** first$(C_i \bullet R, C_i \bullet B) = (r, \bar{j})$
→
$(o, s) := OP(r, \bar{j}, C_i \bullet S)$;
$C_i \bullet S := s$; $C_i \bullet O := C_i \bullet O \,|\, o$; $C_i \bullet N := C_i \bullet N - 1$; $C_i \bullet R :=$ tail$(C_i \bullet R, C_i \bullet B)$;
$(C_i \bullet B, C_i \bullet W) := BP(C_i \bullet S, C_i \bullet B, C_i \bullet W)$
▯ $C_i \bullet R \neq \sigma$ **and** first$(C_i \bullet R, C_i \bullet B) = \sqcup$
→
$(C_i \bullet B, C_i \bullet W) := BP(C_i \bullet S, C_i \bullet B, C_i \bullet W)$
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P \neq \eta$
→
$pa := AP(C_i \bullet P, C_i \bullet S)$;
if $pa \neq \varnothing$
→
$(a, \bar{j}, p, s) :=$ pick(pa);
$C_i \bullet S := s$; $C_i \bullet P := p$; $C_i \bullet N := C_i \bullet N + 1$; $C_{\bar{j}} \bullet A := C_{\bar{j}} \bullet A \,|\, (a, i)$;
$(C_i \bullet B, C_i \bullet W) := BP(C_i \bullet S, C_i \bullet B, C_i \bullet W)$
▯ $pa = \varnothing$
→
skip
fi
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N = 0$ **and** first$(C_i \bullet T) = p$
→
$C_i \bullet T :=$ tail$(C_i \bullet T)$; $C_i \bullet P := p$
▯ $C_i \bullet R = \sigma$ **and** $C_i \bullet P = \eta$ **and** $C_i \bullet N \neq 0$
→
skip
fi

Informally :
Whenever a reaction is accepted from the buffer, the BP function is applied to choose which sender is next. When an action is sent, the state has changed, so the BP function has to be applied, as is necessary whenever there are indeed reactions in the buffer, but not any from the sender that has been chosen. If the priority was *wait* then applying the BP function will have no effect, since *wait* means that we definitely wait for a message to come from the chosen buffer.

Machine m (with $id(m) = j$) continuously executes the following :

if $C_j \bullet A \neq \sigma$ **and** first$(C_j \bullet A, C_i \bullet B) = (a, \bar{i})$
→
$(r, s) := E(a, C_j \bullet S)$;
$C_j \bullet S := s$; $C_j \bullet A :=$ tail$(C_j \bullet A, C_i \bullet B)$; $C_{\bar{i}} \bullet R := C_{\bar{i}} \bullet R \,|\, (r, j)$;
$(C_j \bullet B, C_j \bullet W) := BP(C_j \bullet S, C_j \bullet B, C_j \bullet W)$
▯ $C_j \bullet A \neq \sigma$ **and** first$(C_j \bullet A, C_j \bullet B) = \sqcup$
→
$(C_j \bullet B, C_j \bullet W) := BP(C_j \bullet S, C_j \bullet B, C_j \bullet W)$
▯ $C_j \bullet A = \sigma$

→
skip
fi

Of course, the use of BP by m is similar to that of BP by h.

Initially the following conditions should hold :
$C_i \bullet B \in MID$, $C_j \bullet B \in HID$, $C_i \bullet W = nowait$ and $C_j \bullet W = nowait$.

In order to avoid the possibility of starvation of messages in the buffers the BP functions should satisfy some conditions.

For handler h $(id(h) = i)$ holds :

$$
\begin{array}{l}
\mathbf{E}\, k : k \in I\!N : \\
\mathbf{A}\, j : j \in MID : \\
\mathbf{A}\, s,s_1,b,b_1,w,w_1 : \\
\quad s \in S \wedge s_1 \in S \wedge b \in MID \wedge b_1 \in MID \wedge w \in W \wedge w_1 \in W \\
\quad \wedge (s_1,b_1,w_1) \in R_k(s,b,w) : \\
\mathbf{E}\, s_2,w_2,l : s_2 \in S \wedge w_2 \in W \wedge l \in I\!N : \\
\quad (s_2,j,w_2) \in R_l(s,b,w) \wedge (s_1,b_1,w_1) \in R_{k-l}(s_2,j,w_2) ,
\end{array}
$$

with R defined by :

$$
\begin{array}{l}
R_0(s,b,w) = \{ (s,b,w) \}, \\
R_k(s,b,w) \\
= \\
\{ (s_1,b_1,w_1) \\
\mid \mathbf{E}\, s_2,b_2,w_2,r,m,o : \\
\qquad (s_2,b_2,w_2) \in R_{k-1}(s,b,w) \wedge r \in R \wedge m \in MID \wedge o \in O : \\
\qquad\quad OP(r,m,s_2) = (o,s_1) \wedge BP(s_1,b_2,w_2) = (b_1,w_1) \\
\vee\ \mathbf{E}\, b_2,w_2 : (s_1,b_2,w_2) \in R_{k-1}(s,b,w) : \\
\qquad BP(s_1,b_2,w_2) = (b_1,w_1) \\
\vee\ \mathbf{E}\, s_2,b_2,w_2,a,m,p,p_1 : \\
\qquad a \in A \wedge m \in MID \wedge p \in T \wedge p_1 \in T \wedge (s_2,b_2,w_2) \in R_{k-1}(s,b,w) : \\
\qquad\quad (a,m,p_1,s_1) \in AP(p,s_2) \wedge BP(s_1,b_2,w_2) = (b_1,w_1) \}.
\end{array}
$$

For machine m $(id(m) = j)$ holds :

$$
\begin{array}{l}
\mathbf{E}\, k : k \in I\!N : \\
\mathbf{A}\, i : i \in HID : \\
\mathbf{A}\, s,s_1,b,b_1,w,w_1 : \\
\quad s \in S \wedge s_1 \in S \wedge b \in HID \wedge b_1 \in HID \wedge w \in W \wedge w_1 \in W \\
\quad \wedge (s_1,b_1,w_1) \in R_k(s,b,w) : \\
\mathbf{E}\, s_2,w_2,l : s_2 \in S \wedge w_2 \in W \wedge l \in I\!N : \\
\quad (s_2,i,w_2) \in R_l(s,b,w) \wedge (s_1,b_1,w_1) \in R_{k-l}(s_2,i,w_2) ,
\end{array}
$$

with R defined by :

$$
\begin{array}{l}
R_0(s,b,w) = \{ (s,b,w) \}, \\
R_k(s,b,w) \\
= \\
\{ (s_1,b_1,w_1) \\
\mid \mathbf{E}\, s_2,b_2,w_2,a,r :
\end{array}
$$

$$(s_2,b_2,w_2) \in R_{k-1}(s,b,w) \wedge r \in R \wedge a \in A :$$
$$E(a,s_2) = (r,s_1) \wedge BP(s_1,b_2,w_2) = (b_1,w_1)$$
$$\vee \mathbf{E}\, b_2,w_2 : (s_1,b_2,w_2) \in R_{k-1}(s,b,w) :$$
$$BP(s_1,b_2,w_2) = (b_1,w_1) \}.$$

Informally :
$R_k(s,b,w)$ is the set of elements of $S \times MID \times W$ (in case of a handler), that can be reached from (s,b,w) (also an element of $S \times MID \times W$), by performing k times a step in which BP is applied. So, it is guaranteed that for each state s and each sender j holds that after at most k times of choosing a sender j is chosen.

As far as exclusion is concerned, there are almost no problems, if we assume that the buffer, as we described it, is indeed implemented as one buffer per sender. Then, we only have to assure that, when at almost the same time a message is added to the buffer (as a whole) and it is tried to take a message away and these two messages are in fact the same, then this should happen in a consistent way.

3 THE EQUIVALENCE OF THE MODELS

3.0 Equivalence

After describing a number of models for distributed information systems, we now demonstrate how a given system of a certain model can be simulated by systems of other models, that is, we show the equivalence of the set of the models that have just been presented.
We call a set of models equivalent if any model can be simulated by every other model in the set.

We write A $\rightarrow$ B, short for A simulates B, if for every system b of model B a system a of model A can be found, such that, given an initial state and an input, a produces a final state and an output that could have been produced by b.
Now we are going to give some of these simulations. Suppose we want to demonstrate that A $\rightarrow$ B. We always start with a system b of B and define a system a of A such that a operates as b possibly could do.

3.1 SBIS $\rightarrow$ SimIS

The simulation of SBIS by SimIS is very straightforward, since essentially the model that should be simulated (SimIS) is a special case of the simulating model (SBIS).

We therefore define the SBIS system almost identical to the given SimIS. The only difference is implied by the fact that there are some definitions in the SBIS model that slightly differ from the corresponding definitions in the SimIS model. But it is trivial (as we mentioned, before describing the SimIS model), that these differences are not really differences in the case of a system that has exactly one handler and one machine.

Suppose in the SimIS holds :
the handler is $(S_h, T_h, A_h, R_h, O_h, OP_h, AP_h)$, and the machine is (S_m, A_m, R_m, E_m).
Then the SBIS is defined by :
$H = \{h\}$, $M = \{m\}$, $id(h) = \bar{h}$, $id(m) = \bar{m}$,
$h = (S, T, A, R, O, OP, AP)$, where :
$S = S_h$, $T = T_h$, $A = A_h$, $R = R_h$, $O = O_h$,
$OP(r, \bar{m}, s) = OP_h(r, s)$
$(r \in R$ and $s \in S)$,
$AP(p, s) = \{(a, \bar{m}, p', s') \mid (a, p', s') \in AP_h(p, s)\}$
$(p \in T$ and $s \in S)$,
$m = (S, A, R, E)$, where :
$S = S_m$, $A = A_m$, $R = R_m$, $E = E_m$.

3.2 SimIS $\rightarrow$ SBIS

Here we have the task to construct, given an SBIS, a SimIS, that operates as the SBIS could operate. This means that a SimIS should be constructed that, given an input and an initial state, produces an output and a final state that could have been produced by the SBIS.

We will use the fact that it is possible in the SBIS that each transaction is assigned to the same handler, since in an SBIS it is not defined how the input, being a sequence of transactions, is partitioned over the handlers.
Therefore our strategy for this simulation, is to choose the one handler of the SimIS almost identical to a handler of the SBIS. Furthermore, the machine of the SimIS should

be able to do as much as all machines of the SBIS could do, therefore we will combine all these machines into the one machine of the SimIS.

Suppose in the SBIS $h \in H$, with $id(h) = i$ and $h = (S_i, T_i, A_i, R_i, O_i, OP_i, AP_i)$.
For the SimIS we now define :
the handler is (S, T, A, R, O, OP, AP), where :
$S = S_i$, $T = T_i$, $O = O_i$, $A = A_i \times MID$, $R = R_i \times MID$,
$OP((r, j), s) = OP_i(r, j, s)$
$((r, j) \in R$ and $s \in S)$,
$AP(p, s) = \{((a, j), p', s') \mid (a, j, p', s') \in AP_i(p, s)\}$
$(p \in T$ and $s \in S)$.

Suppose in the SBIS $M = \{(S_j, A_j, R_j, E_j) \mid j \in MID\}$.
We define for the SimIS :
the machine is (S, A, R, E), where :
$S = \mathbf{C}\ j : j \in MID : S_j$, (i.e. the carthesian product of all S_j)
$A = \{(a, j) \mid j \in MID \wedge a \in A_j\}$,
$R = \{(r, j) \mid j \in MID \wedge r \in R_j\}$,
$E((a, j), \bar{s})) = ((r, j), (\hat{s}))$, where
$\bar{s} \in S$ (with the j-component of $\bar{s}$ is s) and
$\hat{s}$ is the state $\bar{s}$ except for the j-component which is s_j,
if $E_j(a, s_j) = (r, s)$,
$(a \in A_j$, $j \in MID$ and $\bar{s} \in S)$.

3.3 BBIS → SBIS

For this simulation we use the idea that a BBIS operates almost the same as an SBIS. The only difference is that in a BBIS there will be some waiting for buffers to become not full.

Therefore in the BBIS we define :
H = "H of the SBIS", M = "M of the SBIS",
id = "id of the SBIS" (so also ID = "ID of the SBIS"),
$\mathbf{A}\ i : i \in ID : BS(i) = 1$.

So we use the idea that waiting for some space in buffers has no disadvantages, i.e. it does not imply results that could not have been produced by the SBIS.
Whenever there is some waiting before sending an action, then possibly after the waiting the original action is sent. Since we do not consider time aspects, that does not imply any bad effects.
If the waiting implies the sending of some other action, then the NTSS schedule implies that this has no influence on the result, since otherwise the schedule would have prohibited this changing of the order of sending of both actions.
The acceptation of some reaction while waiting to send, is also allowed because of the independence of both operations.
Since the machines are only able to wait (i.e. they have not the possibility of executing some other action), there is no problem as far as the machines are concerned.
One must note that whenever two reactions of different machines arrive in some other order as they would in the SBIS, then that is not a problem (note that this order of arrival influences the result), since this is a question of time. Two reactions of the same machine can not change order.

3.4 SBIS → BBIS

This simulation is rather trivial. We choose for the SBIS :
H = "H of the BBIS", "M = M of the BBIS".

The SBIS will operate as the BBIS would, but without its limitations. This notion can indeed be used, since the BBIS can not use in any way its limitations (i.e. the bounds for the buffer sizes).

3.5 SeqIS → SBIS

We will simulate the SBIS by allowing in the SeqIS every object to perform a step. This allowance will be given to one object at a time, and every object will eventually get the allowance to perform a step.

So we define in the SeqIS :
H = "H of the SBIS", M = "M of the SBIS",
ID = "ID of the SBIS".
Furthermore, we define the *ACTIVE* function for the SeqIS :
ACTIVE : $[0..|ID|-1] \rightarrow ID$, with *ACTIVE* a one-to-one function.

That the SBIS can be simulated in this way is obvious, since essentially we only exclude parallelism.

3.6 MBIS → SeqIS

For this simulation we will use the fact that it is possible in the SeqIS that each transaction is assigned to the same handler, since in a SeqIS it is not defined how the input, being a sequence of transactions, is partitioned over the handlers. Therefore we choose in the MBIS the number of handlers to be one and the number of machines to be one higher than the number of machines in the SeqIS. This implies that we choose that the one handler in the MBIS is almost identical to a handler of the SeqIS.

One of the machines (α) will, together with the handler, control the steps to be performed as in the SeqIS. The other machines are almost identical to the machines of the SeqIS. The handler h will be a handler of the SeqIS, but with some special tasks. How is the controlling of the performing of steps organized ?

The handler will ask α which object is allowed to perform a step, since α will contain the information concerning the *ACTIVE* function of the SeqIS. Then h's task is to control that the object, that has the allowance, performs a step. Note that a machine can not ask itself whether it can perform a step. In order to run this controlling smoothly we define the operations of the objects in such way that the contents of their buffer will be in their internal buffer, which will be part of the state.

If h learns from a reaction, which it has got from α, that it is allowed to perform a step itself, then it performs that step and subsequently tells that it has finished with this step by asking α again who is next to perform a step. When performing a step, h can possibly send an action to some machine m (not α). Then m will act as follows. It stores the action in its internal buffer and sends to h a message, that represents some empty (= irrelevant) reaction. Subsequently h will ask α again which object is next.

If some machine m is the object allowed to perform a step, then h will send an (empty) action to m and m will then react by executing the first action in its internal buffer and

sending the reaction, implied by this execution, to h. h puts this message in its internal buffer in order to be able to send an action to α, asking who is next.

If h has the permission itself and it has some message in its internal buffer, then it will act as it would do in the SeqIS case whenever its buffer is not empty, and it will subsequently send an action to α again.

In this way we have programmed the MBIS to operate in such a way that the "real steps" are performed in the same sequence as in the SeqIS.

Formally :

Suppose in the SeqIS $(S,T,A,R,O,OP,AP) \in H$.
Define in the MBIS :

$H = \{h\}$ with $h = (S_h, T, A_h, R_h, O, OP_h, AP_h, BP_h)$ and $id(h) = \bar{h}$, where :

$S_h = S \times SS \times (R \times MID)^* \times \mathbb{N}$, where :
MID as in the SeqIS, $\bar{\alpha} \notin MID$ $(id(\alpha) = \bar{\alpha})$,
$SS = \{ask, ready, waitemp, change\} \cup \{all(x) \mid x \in MID \cup \{\bar{h}\}\} \cup$
$\{wait(m) \mid m \in MID\} \cup \{waitr(m) \mid m \in MID\}$.

$AP(p,(s,ready,rm,nr)) = \{(who?, \alpha, p, (s, ask, rm, nr))\}$,
$OP(ok(x), \bar{\alpha}, (s, ask, rm, nr)) = (\sigma, (s, all(x), rm, nr))$.

$AP(p,(s,all(h),\sigma,nr)) = \{(a, m, \bar{p}, (s', wait(m), \sigma, nr+1)) \mid (a, m, p', s') \in AP(p,s) \wedge$
$(p' \neq \eta \Rightarrow \bar{p} = p' \wedge p' = \eta \Rightarrow \bar{p} = \omega)\}$
$(p \neq \omega)$,
$OP(\mu, m, (s, wait(m), rm, nr)) = (\sigma, (s, ready, rm, nr))$.

$AP(p,(s,all(\bar{h}),rm,nr)) = \{(\mu, \bar{\alpha}, p, (s, waitemp, rm, nr))\}$
$(rm \neq \sigma)$,
$OP(\mu, \bar{\alpha}, (s, waitemp, rm, nr)) = (o, (s', ready, \text{tail}(rm), nr))$
$(rm \neq \sigma$, $\text{first}(rm) = (r, m)$ and $OP(r, m, s) = (o, s'))$.

$AP(p,(s,all(m),rm,nr)) = \{(\sigma, m, p, (s, waitr(m), rm))\}$,
$OP(r, m, (s, waitr(m), rm, nr)) = (\sigma, (s, ready, rm \mid (r, m), nr-1))$.

$AP(\omega,(s,all(\bar{h}),\sigma,0)) = \{(\mu, \bar{\alpha}, \eta, (s, change, \sigma, 0))\}$,
$OP(\mu, \bar{\alpha}, (s, change, \sigma, 0)) = (\sigma, (s, ready, \rho, 0))$.

$AP(p,(s,all(\bar{h}),\sigma,nr)) = \{(who?, \bar{\alpha}, p, (s, ask, \sigma, nr))\}$
$(AP(p,s) = \varnothing$ and $nr > 0)$.

$OP(\mu, m, (s, waitr(m), rm, nr)) = (\sigma, (s, ready, rm, nr))$.

$A_h = A \cup \{who?, \mu\}$.
$R_h = R \cup \{ok(x) \mid x \in MID \cup \{\bar{h}\}\} \cup \{\mu\}$.

$BP_h((s, ask, rm, nr), m, w) = (\bar{\alpha}, wait)$,
$BP_h((s, ask, rm, nr), \bar{\alpha}, wait)) = (\bar{\alpha}, wait)$,
$BP_h((s, wait(m), rm, nr), m', w) = (m, wait)$,
$BP_h((s, wait(m), rm, nr), m, wait) = (m, wait)$,
$BP_h((s, waitemp, rm, nr), m, w) = (\bar{\alpha}, wait)$,

$BP_h((s, waitemp, rm, nr), \overline{\alpha}, wait)) = (\overline{\alpha}, wait)$.
$BP_h((s, waitr(m), rm, nr), m', w) = (m, wait)$.
$BP_h((s, waitr(m), rm, nr), m, wait) = (m, wait)$.
$BP_h((s, change, rm, nr), m, w) = (\overline{\alpha}, wait)$.
$BP_h((s, change, rm, nr), \overline{\alpha}, wait)) = (\overline{\alpha}, wait)$.
$(m \in MID$ and $w \in W)$.

For every $(S, A, R, E) \in M$ in the SeqIS, we define in the MBIS $(S_m, A_m, R_m, E_m, BP_m) \in M$ where :

$S_m = S \times A^*$.
$A_m = A \cup \{\mu\}$ $(\mu \notin A \cup R)$.
$R_m = R \cup \{\mu\}$.

$E_m(a, (s, ah)) = (\mu, (s, ah \mid a))$.
$E_m(\mu, (s, ah)) = (r, (s', \text{tail}(ah)))$
$(E(a, s) = (r, s')$ and $\text{first}(ah) = a)$.
$E_m(\mu, (s, \sigma)) = (\mu, (s, \sigma))$.

$\text{rng}(BP_m) = \{\overline{h}\}$.

Furthermore, for the MBIS are defined $ACTIVE'$ and $\alpha \in M$, where $\alpha = (S, A, R, E, BP)$, with :

if in the SeqIS $ACTIVE : [0..n-1] \rightarrow ID$, then $ACTIVE' : [0..n'-1] \rightarrow MID \cup \{\overline{h}\}$ is defined by :

$q := 0$; $q' := 0$;
do $q < n$
→
if $ACTIVE(q) \in MID$
→
$ACTIVE'(q') := ACTIVE(q)$; $q' := q' + 1$
[] $ACTIVE(q) = i$
→
$ACTIVE'(q') := \overline{h}$; $q' := q' + 1$
[] $ACTIVE(q) \notin MID \cup \{i\}$
→
skip
fi;
$q := q + 1$
od;
$n' := q'$

$S = [0..n'-1]$,
$A = \{who?, \mu\}$,
$R = \{\mu\} \cup \{ok(x) \mid x \in MID \cup \{\overline{h}\}\}$ (MID as in the SeqIS).

$E(who?, q') = (ok(ACTIVE(q')), q' + 1 (mod\ n'))$,
$E(\mu, q') = (\mu, q')$.

$\text{rng}(BP) = \{\overline{h}\}$.

3.7 SBIS → MBIS

A rather straightforward idea for this simulation is to make the buffers of the MBIS, which have the *BP* function that is not easy to simulate, internal buffers, i.e. to keep the contents of such a buffer in the state, and then to have every object ask a special machine β, which knows everything about the *BP* functions, which result should be produced given the internal buffer.

Because it is rather difficult for the machines to ask some other machine anything, we use again the fact that in the MBIS every transaction could be assigned to the same handler. So in the SBIS we will define one handler which will act almost the same as any handler of the MBIS.

This handler h will, when having messages in its buffer, produce first an empty (= irrelevant) result and change into some special state, that implies the sending of an action to β, which causes β to send a reaction that supplies the machine with the necessary information to be able to produce the right result and to change into the right state.

The definition of the machines is rather trivial, since the *BP* function has no real meaning, whenever there is only one sender.
Therefore we choose :

$M = \{(S,A,R,E) \mid (S,A,R,E,BP) \in$ "M of the MBIS"$\} \cup \{\beta\}$,
where $MID =$ "MID of the MBIS" $\cup \{\overline{\beta}\}$ and $id(\beta) = \overline{\beta}$.

If $(S,T,A,R,O,OP,AP,BP) \in H$ in the MBIS, then we define in the SBIS :
$H = \{h\}$, where $h = (S_h,T,A_h,R_h,O,OP_h,AP_h)$, $id(h) = \overline{h}$ and

$$S_h = S \times \{work,ask,wait\} \times (R \times MID)^* \times \mathbb{N},$$

$$OP_h(r,m,(s,work,rm,nr)) = (\sigma,(s,ask,rm \downarrow (r,m),nr-1)),$$
$$AP_h(p,(s,ask,rm,nr)) = \{(which(rm,s)?,\beta,p,(s,wait,rm,nr))\},$$
$$OP_h((r,m),\beta,(s,wait,rm,nr)) = (o,(s',work,rm',nr))$$
$$(OP(r,m,s) = (o,s') \text{ and } "rm' = rm - (r,m)"),$$
$$OP_h(\sqcup,\beta,(s,wait,rm,nr)) = (\sigma,(s,ask,rm,nr)).$$

$$AP_h(p,(s,work,\sigma,nr)) = \{(a,m,\overline{p},(s',work,\sigma,nr+1)) \mid (a,m,p',s') \in AP(p,s) \wedge (p' = \eta \Rightarrow \overline{p} = \omega \wedge p' \neq \eta \Rightarrow \overline{p} = p')\}$$
$$(p \neq \omega).$$

$$AP_h(\omega,(s,work,\sigma,0)) = \{(\mu,\overline{\beta},\eta,(s,wait,\sigma,0))\},$$
$$OP_h(\mu,\overline{\beta},(s,wait,\sigma,0)) = (\sigma,(s,work,\sigma,0)).$$

$$A_h = A \cup \{which(rm,s)? \mid rm \in (R \times MID)^* \wedge s \in S\} \cup \{\mu\}.$$
$$R_h = R \cup \{\mu,\sqcup\} \cup (R \times MID).$$

For machine β will be defined :
$$E(\mu,s) = (\mu,s),$$
$$E(which(rm,s)?,m',w') = ((r,m),m,w)$$
$$(BP(s,m',w') = (m,w) \text{ and } \text{first}(rm,m) = (r,m)),$$
$$E(which(rm,s)?,m',w') = (\sqcup,m,w)$$
$$(BP(s,m',w') = (m,w) \text{ and } \text{first}(rm,m) = \sqcup).$$

With this simulation we have seven simulations that can easily be seen to prove the

equivalence of the set of the models that we have introduced.

4 GENERAL DISTRIBUTED INFORMATION SYSTEM

Here we present a model called GDIS (from : general distributed information system). The models previously presented will be special cases of the GDIS. Note, that each of these models is assumed to be identical with the special case of the GDIS, that is trivially equivalent to the model. This GDIS model covers distributed information systems, which operate according to the NTSS schedule but in which the communication (between handlers and machines) is performed in many different ways.

A GDIS can be described as follows.

H is a set of transaction handlers.
M is a set of machines.

There is a function $id : H \cup M \rightarrow ID$, where :
id is a one-to-one function,
$HID \subset ID$ and $\mathbf{A}\, h : h \in H : id(h) \in HID$,
$MID \subset ID$ and $\mathbf{A}\, m : m \in M : id(m) \in MID$.

Define $W = \{wait, nowait\}$.

Let h be a handler, so $h \in H$, with $id(h) = i$, say.
Then $h = (S, T, A, R, O, OP, AP, SP, BP)$, where :
- S is a set of states,
- T is a set of transactions,
- A is a set of actions,
- R is a set of reactions,
- O is a set of results (outputs),
- $OP : R \times MID \times S \rightarrow O \times S$, is a result producer,
- $AP : T \times S \rightarrow \mathbb{P}(A \times MID \times T \times S)$, is an action producer,
- $SP \in \mathbb{P}(\mathbb{P}(MID))$, is a sender partitioning, where
  - $\mathbf{A}\, m : m \in MID : \mathbf{E}(s : s \in SP : m \in s)$,
  - $\mathbf{A}\, s, \bar{s} : s \in SP \wedge \bar{s} \in SP \wedge s \neq \bar{s} : s \cap \bar{s} = \varnothing$,
- $BP : S \times SP \times W \rightarrow SP \times W$, is a buffer priority rule.

The configuration of h, denoted by C_i, is an element of
$S \times T^* \times O^* \times T \times \mathbb{N} \times (R \times MID)^* \times SP \times W$.
We define : if $C_i = (s, t, o, p, n, r, b, w)$, then
$C_i \bullet S = s$, $C_i \bullet T = t$, $C_i \bullet O = o$, $C_i \bullet P = p$, $C_i \bullet N = n$, $C_i \bullet R = r$, $C_i \bullet B = b$, $C_i \bullet W = w$.

Informally :
SP partitions the senders (machines) for this handler, thus modeling that every element of SP has its own buffer. So for every sender there is exactly one buffer, but it is possible that the sender has to share the buffer with some other senders. Note that the two extremes are the MBIS, where each sender has its own buffer, and the SBIS, where all senders share one buffer.
One should note that when we say that there is more than one buffer, we represent this by having just one buffer, which is accessed not first-in-first-out, but only first-in-first-out as far as some set of senders, as determined by BP, is concerned (cf. the description of the MBIS model).

As in an MBIS BP determines the buffer to access (the set of machines of which a message should be taken) and the priority with which to access that buffer. The only difference for BP is that in stead of one machine a set of machines is determined.

In the configuration $C_i \bullet B$ stands for the buffer (i.e. the set of machines) from which a message should be taken if we want to do so.

Let m be a machine, so $m \in M$, with $id(m) = j$, say.
Then $m = (S, A, R, E, SP, BP)$, where :
- S is a set of states,
- A is a set of actions,
- R is a set of reactions,
- $E : A \times S \to R \times S$, is an executor,
- $SP \in \mathbb{P}(\mathbb{P}(HID))$, is a sender partitioning, where
  - $\mathbf{A}\, h : h \in HID : \mathbf{E}(s : s \in SP : h \in s)$,
  - $\mathbf{A}\, s, \bar{s} : s \in SP \wedge \bar{s} \in SP \wedge s \neq \bar{s} : s \cap \bar{s} = \varnothing$,
- $BP : S \times SP \times W \to SP \times W$, is a buffer priority rule.

The configuration of m, denoted by C_j, will be an element of
$S \times (A \times HID)^* \times SP \times W$.
We define : if $C_j = (s, a, b, w)$, then
$C_j \bullet S = s$, $C_j \bullet A = a$, $C_j \bullet B = b$, $C_j \bullet W = w$.

There is a function $ACTIVE : [0..n-1] \to \mathbb{P}(ID)$, with every element of ID occurring in at least one set that acts as a value of $ACTIVE$.
We will use q to run through $[0..n-1]$. Informally, $ACTIVE(q)$ is the set of identifications of the handlers and the machines that are currently allowed to perform a step. The difference with the SeqIS is that in the SeqIS exactly one handler or machine has the permission, whereas in the GDIS a set of handlers or machines gets the permission, which means that every element of that set can perform a step.

There will be also a function $BS : ID \to (\mathbb{N} - \{0\}) \cup \{ \sqcup \}$, where BS stands for buffer size. For each $i \in ID$, $BS(i)$ denotes whether the object identified by i, say h, has a bounded buffer. If $BS(i) = \sqcup$, then h has an unbounded buffer. If $BS(i) \in \mathbb{N} - \{0\}$, then the size of h's buffer is $BS(i)$, which means that the buffer can hold at most $BS(i)$ messages at a time.

We assume that the functions $ACTIVE$ and BS, which are really central functions, are represented in the scheduler. So the scheduler executes the schedule and controls the use of these two functions.

Handler h (with $id(h) = i$) continuously executes the following :

if $i \in ACTIVE(q)$
→
  if $C_i \bullet R \neq \sigma$ **and** first-set$(C_i \bullet R, C_i \bullet B) = (r, \bar{j})$
  →
    $(o, \bar{s}) := OP(r, \bar{j}, C_i \bullet S)$;
    $C_i \bullet S := s$; $C_i \bullet O := C_i \bullet O \,|\, o$; $C_i \bullet N := C_i \bullet N - 1$; $C_i \bullet R :=$ tail-set $(C_i \bullet R, C_i \bullet B)$;
    $(C_i \bullet B, C_i \bullet W) := BP(C_i \bullet S, C_i \bullet B, C_i \bullet W)$
  [] $C_i \bullet R \neq \sigma$ **and** first-set$(C_i \bullet R, C_i \bullet B) = \sqcup$
  →
    $(C_i \bullet B, C_i \bullet W) := BP(C_i \bullet S, C_i \bullet B, C_i \bullet W)$
  [] $C_i \bullet R = \sigma$ **and** $C_i \bullet P \neq \eta$
  →
    $pa := AP(C_i \bullet P, C_i \bullet S)$;
    do $pa \neq \varnothing$
    →

```
            (a ,j̄ ,p ,s ) := pick(pa );
            if size(C_j̄•A ) < BS(j̄ )
               →
               C_i•S := s ; C_i•P := p ; C_i•N := C_i•N + 1; C_j̄•A := C_j̄•A |(a ,i );
               (C_i•B ,C_i•W ) := BP(C_i•S ,C_i•B ,C_i•W );
               pa := ∅
            [] size(C_j̄•A ) = BS(j̄ )
               →
               pa := pa - (a ,j̄ ,p ,s )
            fi
         od
      [] C_i•R = σ and C_i•P = η and C_i•N = 0 and first(C_i•T ) = p
         →
         C_i•T := tail(C_i•T ); C_i•P := p
      [] C_i•R = σ and C_i•P = η and C_i•N ≠ 0
         →
         skip
      fi;
      q := q + 1 (mod n )
   [] i ∉ ACTIVE(q )
      →
      skip
   fi
```

Informally :
First of all h checks (as in a SeqIS) whether it is allowed to perform a step.
If h has the permission, it possibly decides to receive a reaction. As in a MBIS BP will determine of which senders a message can be taken.
Before h sends an action, it must be sure that the message implied can be put in the buffer of the appropriate machine. Therefore the same strategy is applied as is applied when in a BBIS a handler wants to send an action.

Machine m (with $id(m) = j$) continuously executes the following :

```
if j ∈ ACTIVE(q )
   →
   if C_j•A ≠ σ and first-set(C_j•A ,C_i•B ) = (a ,ī )
      →
      (r ,s ) := E(a ,C_j•S );
      if size(C_ī•R ) < BS(ī )
         →
         C_j•S := s ; C_j•A := tail-set(C_j•A ,C_i•B ); C_ī•R := C_ī•R |(r ,j );
         (C_j•B ,C_j•W ) := BP(C_j•S ,C_j•B ,C_j•W )
      [] size(C_ī•R ) = BS(ī )
         →
         skip
      fi
   [] C_j•A ≠ σ and first-set(C_j•A ,C_j•B ) = □
      →
      (C_j•B ,C_j•W ) := BP(C_j•S ,C_j•B ,C_j•W )
   [] C_j•A = σ
      →
      skip
   fi;
```

$q := q + 1 \pmod n$
$\square$ $j \notin ACTIVE(q)$
$\rightarrow$
skip
fi

Similar to the operation of h, m will check for the permission and if m has got the permission it chooses, with the help of BP (and SP), an action, after which it determines with BS whether a message implied by the execution of the action can be put in the right buffer.

Initially the following conditions should hold :
$C_i \bullet N = 0$, $C_i \bullet B \in SP$ (of i), $C_i \bullet W = nowait$.

In order to avoid the possibility of starvation of messages in the buffers the BP functions should satisfy some conditions.

For handler h $(id(h) = i)$ holds :

$$
\begin{aligned}
&\mathbf{E}\, k : k \in \mathbb{N} : \\
&\mathbf{A}\, j : j \in MID : \\
&\mathbf{A}\, s,s_1,b,b_1,w,w_1 : \\
&\quad s \in S \wedge s_1 \in S \wedge b \in SP \wedge b_1 \in SP \wedge w \in W \wedge w_1 \in W \\
&\quad \wedge (s_1,b_1,w_1) \in R_k(s,b,w) : \\
&\mathbf{E}\, s_2,b_2,w_2,l : s_2 \in S \wedge b_2 \in SP \wedge w_2 \in W \wedge l \in \mathbb{N} : \\
&\quad (s_2,b_2,w_2) \in R_l(s,b,w) \wedge (s_1,b_1,w_1) \in R_{k-l}(s_2,b_2,w_2) \wedge j \in b_2 ,
\end{aligned}
$$

with R defined by :

$$
\begin{aligned}
&R_0(s,b,w) = \{ (s,b,w) \}, \\
&R_k(s,b,w) \\
&= \\
&\{ (s_1,b_1,w_1) \\
&\mid \mathbf{E}\, s_2,b_2,w_2,r,m,o : \\
&\qquad (s_2,b_2,w_2) \in R_{k-1}(s,b,w) \wedge r \in R \wedge m \in MID \wedge o \in O : \\
&\qquad\quad OP(r,m,s_2) = (o,s_1) \wedge BP(s_1,b_2,w_2) = (b_1,w_1) \\
&\vee \mathbf{E}\, b_2,w_2 : (s_1,b_2,w_2) \in R_{k-1}(s,b,w) : \\
&\qquad BP(s_1,b_2,w_2) = (b_1,w_1) \\
&\vee \mathbf{E}\, s_2,b_2,w_2,a,m,p,p_1 : \\
&\qquad a \in A \wedge m \in MID \wedge p \in T \wedge p_1 \in T \wedge (s_2,b_2,w_2) \in R_{k-1}(s,b,w) : \\
&\qquad\quad (a,m,p_1,s_1) \in AP(p,s_2) \wedge BP(s_1,b_2,w_2) = (b_1,w_1) \}.
\end{aligned}
$$

For machine m $(id(m) = j)$ holds :

$$
\begin{aligned}
&\mathbf{E}\, k : k \in \mathbb{N} : \\
&\mathbf{A}\, i : i \in HID : \\
&\mathbf{A}\, s,s_1,b,b_1,w,w_1 : \\
&\quad s \in S \wedge s_1 \in S \wedge b \in SP \wedge b_1 \in SP \wedge w \in W \wedge w_1 \in W \\
&\quad \wedge (s_1,b_1,w_1) \in R_k(s,b,w) : \\
&\mathbf{E}\, s_2,b_2,w_2,l : s_2 \in S \wedge b_2 \in SP \wedge w_2 \in W \wedge l \in \mathbb{N} : \\
&\quad (s_2,b_2,w_2) \in R_l(s,b,w) \wedge (s_1,b_1,w_1) \in R_{k-l}(s_2,b_2,w_2) \wedge i \in b_2 ,
\end{aligned}
$$

with R defined by :

$$
\begin{aligned}
&R_0(s,b,w) = \{ (s,b,w) \},\\
&R_k(s,b,w)\\
&=\\
&\{ (s_1,b_1,w_1)\\
&\mid \mathbf{E}\, s_2,b_2,w_2,r,a :\\
&\qquad (s_2,b_2,w_2) \in R_{k-1}(s,b,w) \wedge r \in R \wedge a \in A :\\
&\qquad\quad E(a,s_2) = (r,s_1) \wedge BP(s_1,b_2,w_2) = (b_1,w_1)\\
&\vee \mathbf{E}\, b_2,w_2 : (s_1,b_2,w_2) \in R_{k-1}(s,b,w) :\\
&\qquad BP(s_1,b_2,w_2) = (b_1,w_1) \}.
\end{aligned}
$$

To avoid that handlers or machines that want to perform a step never know that they are allowed to do so (the condition we added to the definition of *ACTIVE* assures that they definitely will have the permission at some time, but it could be possible that they will not become aware of that fact), we presume that every handler or machine that does not have the allowance to perform a step, is waiting to get the allowance, i.e. immediately after q has got a value such that a handler or a machine has the permission, the handler or machine becomes aware of that and starts performing a step.
So in between two transitions of the value of q, every object that was waiting for the allowance becomes aware of having that allowance.

5 WHICH MODEL IS THE MOST PARALLEL MODEL ?

As we have seen, the GDIS model defines a large set of systems. We also could say, that the GDIS model implies a set of models, where every model is characterized by the way in which the communication between transaction handlers and machines, independent of NTSS, is organized. As we have seen, the five models that we presented earlier are elements of this set of models.

The question now is the following.
In which model is the communication organized in such a way that we could say that the model is the most parallel one in the set of all models implied by GDIS ?

Intuitively, the most parallel model is the model for which holds, that any transaction, being handled by some handler, is interfered in the least possible way by transactions, that are handled at the same time.
When we say, that a transaction is interfered by other transactions, we mean that the progress of its handling is slowed down, because the handling at the same time of the other transactions requires some arrangements to be made.
These arrangements are needed in order to guarantee that every transaction is handled in a correct way, which means that every handling of a transaction takes only a finite time and for each input and each state of the system, the output and the resulting state are computed in a consistent manner.

So we consider the GDIS model.
Note that the SimIS model obviously is not the most parallel model. We will only compare the models that allow systems to have a set of handlers and a set of machines.
If one wants to define a GDIS, then one has to specify the handlers, the machines and the scheduler. Of course, one also has to define the identification function id, but surely the definition of id has no effect on the parallelism.

Since a handler is defined to be a nine-tuple $(S, T, A, R, O, OP, AP, SP, BP)$, this means that for every handler we have to define what this nine-tuple is. Obviously, the definition of the sets S, T, A, R and O has no effect on the parallelism that is allowed in the system. The functions OP and AP are merely functions that determine some transformation, i.e. which result is implied by a reaction (OP) or which set of actions is implied by a transaction (AP). Therefore they do certainly not contribute to a higher degree of parallelism.
The function SP has an effect on the degree of parallelism, as we have learned from the description of the MBIS model. This implies that, when we want to define the most parallel model, we have to define SP in the best possible way. The definition of BP is of course dependent on the definition of SP. This implies that as far as the handlers are concerned, the definition of the most parallel model implies the best possible definition of SP and BP.

A machine is a six-tuple (S, A, R, E, SP, BP). Again, the definition of the sets S, A and R has no effect at all on the degree of parallelism. The function E is a function, that merely determines which reaction and state are implied by an action and a state. So, as far as the machines are concerned, we only have to define SP and BP in the best possible way.

Furthermore, the definition of a GDIS implies the definition of the scheduler. Since we have already defined that the main task of the scheduler is to execute the NTSS schedule, we only have to define the functions $ACTIVE$ and BS.
As we have learned from the description of the SeqIS, the definition of $ACTIVE$ has an effect on the parallelism. From the description of the BBIS we know that BS also has its effect on the parallelism.

This means that defining the most parallel model can be reduced to defining *ACTIVE*, *BS* and for every handler and for every machine *SP* and *BP* in the best possible way.

What is the effect of *ACTIVE* on the handling of transactions by a distributed information system ? This function *ACTIVE* determines, independently of the handling of transactions, which objects (handlers or machines) are allowed to be active. If *ACTIVE* is a function from $[0..n]$ to $\mathbb{P}(ID)$ and for some $k \in [0..n]$ holds $ACTIVE(k) \neq ID$, i.e. not always every object is allowed to be active, then obviously the progress of the handling of a transaction can be slowed down, because some object has to wait for the allowance to be active (cf. the introduction of the SeqIS model).
One must note, that, whenever some object has to wait for the allowance, this is due to the fact that it is decided (by *ACTIVE*) that other objects are allowed to be active without this object being active. The reason for this could be, that only a limited number (e.g. 1 in the case of a SeqIS) of objects can be active at the same moment. Surely, this is in contradiction with our notion of the most parallel model. Therefore, in the most parallel model every object should be able to be active at any moment.
So, in the GDIS model *ACTIVE* should be defined as follows :

$ACTIVE : [0] \rightarrow ID$.

If for some $i \in ID$ holds $BS(i) \in \mathbb{N}$, i.e. the buffer of the object identified by i has a bounded size, then it could happen that the handling of a transaction is stopped, because a bounded buffer is full (cf. the introduction of the BBIS model). In general, this is also due to the fact that the handling of other transactions has implied that messages have been put in the buffer.
Therefore, it is much more convenient to have unbounded buffers, because then the actual state of the buffers cannot imply the handling of a transaction to be slowed down.
For the most parallel model, we should define :

$BS : ID \rightarrow \{ \sqcup \}$.

That leaves us to define *SP* and *BP* for every handler and every machine in the system.
From the description of the MBIS model we know, that the *SP* function can be used to partition the senders (i.e. objects that can put messages in the buffer) and that then *BP* can be used to take messages from the buffer in an order, that is not first-in-first-out, but that is determined by *BP*, i.e. *BP* determines a set of senders and then the first element, that is sent by one of the senders in that set, is taken away.
First of all, one must note that if one has the possibility to give some sender some priority (i.e. for a handler $SP \neq \{MID\}$ and for a machine $SP \neq \{HID\}$), then one has an additional possibility to execute a specific problem in a more natural way. Consider for example, the possibility that a handler is expecting a reaction from some machine, where the reaction could imply, that one wants to stop the transaction, because one is not interested in any results in that case (e.g. an error). Then it would be convenient to have this reaction accepted as soon as possible, i.e. without having to wait for the reaction to become the first in the buffer, which is also dependent on the speed of the machines.

There is also another aspect to the buffer definition.
In the case where some senders share a buffer, i.e. the owner of the buffer takes the messages from these senders simply first-in-first-out, one must take care of the situation where two objects want to manipulate a buffer at almost the same time. Of course the SBIS, where for every handler holds $SP = \{MID\}$ and for every machine $SP = \{HID\}$, is an example of such a system. This situation can occur, when two senders both want to add a message to some buffer.
In the case where every sender has its own buffer, so $SP = \{\{j\} \mid j \in MID\}$ for every handler and $SP = \{\{i\} \mid i \in HID\}$ for every machine, this problem can not occur. Of course the MBIS model is the only model for which this condition holds. So then we only have

to deal with the possibility of adding a message and at the same time trying to take the same message away. This implies that the problem of exclusion can easier be solved for the MBIS model.

Therefore the most parallel model should have for every object a single buffer per sender, which means that in the GDIS model one should define :

for every $i \in HID : SP = \{\{j\} \mid j \in MID\}$,
for every $j \in MID : SP = \{\{i\} \mid i \in HID\}$.

Now we can conclude that in the most parallel model the *ACTIVE* function should be defined in a trivial way (i.e. it does not constrain the system at all). Furthermore, every object must have an unbounded buffer and that buffer should be organized in such a way that every sender of messages has really its own (part of the) buffer.
So when we take these conditions and add them to the GDIS model, we have the definition of the most parallel model. Clearly, this implies that the MBIS model is the most parallel one of the models implied by the GDIS model.

Transactions that are handled in an MBIS can be handled almost completely in parallel. There are only two kinds of situations, in which a transaction that is handled in the system is interfered by other transactions.
The first kind of situations are situations that occur, when the schedule takes some measurements that are needed because several transactions want to manipulate with one part of the database and every transaction should have a consistent view of the database. These measurements are of course those described in the NTSS schedule.
The second kind of situations are situations that occur, when some buffer is empty and at (almost) the same time an object wants to add a message to the buffer and the object that owns the buffer wants to take a message away. As we have seen, this should be organized in some consistent way.
Since obviously every model implied by GDIS has to deal with this same schedule, the MBIS has as its advantage that it has organized the communication in such a way that the need of exclusion is the smallest of all models.

6 CONCLUSION

In this paper we describe distributed information systems, which consist of transaction handlers, machines and a scheduler. The machines are responsible for the information in their local database component. The information system executes transactions, which want to manipulate the information in the database. Therefore, transaction handlers get the responsibility for transactions. This implies that transaction handlers and machines have to communicate with each other. The main task of the scheduler is to execute a schedule, that implies that the transactions get a consistent view of the database.

We present here a number of schedules, for which we prove that they are correct schedules. One of those schedules we define to be the schedule for a distributed information system. This schedule NTSS has as its key issue, that it does not use time stamps and can therefore be run in a distributed way.
Furthermore, we present a number of models for (the communication in) distributed information systems. It has been proven that all of these models are equivalent.
Then we describe a general model for distributed information systems. We use this model to show which definitions have to be made, in order to have the most parallel model. It is shown that the MBIS model is the most parallel model, since it has organized the communication between handlers and machines in the best possible way. In the MBIS model transactions can be handled in the most parallel way, mainly due to the fact that each object (handler or machine) has a single buffer for each sender that supplies the object with messages.

7 REFERENCES

K.P. Eswaran, J.N. Gray, R.A. Lorie and I.L. Traiger,
The Notions of Consistency and Predicate Locks in a Relational Database System,
Communications of the ACM, Vol. 19, No. 11, November 1976, pages 624-633.

R.C. Hansdah and L.M. Patnaik,
Update Serializability in Locking,
Proceedings of ICDT '86, Rome, September 1986.

G.J. Houben, J. Paredaens and K.M. van Hee,
The Partition of an Information System in Several Parallel Systems,
Computing Science Notes 86/04, Eindhoven University of Technology, October 1986.

G.J. Houben and J. Paredaens,
A Formal Model for Distributed Information Systems,
Proceedings of MFDBS '87, Dresden, January 1987.

G. Schlageter,
Process Synchronization in Database Systems,
ACM TODS, Vol. 3, No. 3, September 1978, pages 248-271.

H.F. Korth and A. Silberschatz,
Database System Concepts,
McGraw-Hill Book Company, 1986.

COMPUTING SCIENCE NOTES

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language

86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failure semantics for communicating processes
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems

COMPUTING SCIENCE NOTES

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers

86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failure semantics for communicating processes
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview