STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

HALF SYNCHRONIZED TRANSITION SYSTEMS

FLORIAN MIRCEA BOIAN AND CORINA FERDEAN

Abstract. In a distributed system, defined as a collection of interconnected nodes, the underlying role is taken by the communication subsystem. It functions using common protocols which addresses the problem of heterogeneity and homogeneity of the participants nodes, and contributes to the performance of the whole system. No matter what type: *synchronous* or *asynchronous*, the communication software has to be flexible, reusable and adaptable. The formal methods used to model these protocols should reflect these qualities. In this paper, we extend the classical, synchronous and asynchronous message passing models, proposing an intermediary class of models, starting from some limitations, in time and space, imposed upon the entities involved. We named the new model, *half-synchronized transition systems.*

Keywords: distributed systems, distributed algorithms, transition systems, synchronous, asynchronous message passing

1. Introduction

In a distributed system, defined as a collection of interconnected nodes, the underlying role is taken by the communication subsystem. We cannot have distributed computing if there is no communication. Also, the communication protocols addresses the problem of heterogeneity and homogeneity of the participants nodes, and together with the communication media, have a profound influence on the performance of the whole system. These observations lead to the conclusion that the communication software, used to design distributed applications, must have important qualities, among which we mention: flexibility, efficiency, reusability, adaptability.

Also, the communication medium and applications domain properties have a major influence on designing the communications protocols. These protocols can be selected as a compromise between some competing properties. In order to achieve its main purposes, as the underlying system for collaborating between distributed applications, the communication protocols take many forms and they can be modelled in different ways, using abstract languages.

Besides the introduction, this paper is structured in two main parts. The first part presents the two major ways *synchronous* and *asynchronous-,* of modeling communication, between distributed processes, using transition systems. In the second part, the paper extends the models mentioned above and proposes an

intermediary class of models, starting from some limitations, in time and space, imposed upon the entities involved. We named the new model, *half-synchronized transition systems*, and we will describe it both intuitively and formally.

We begin the presentation with the definitions of some background terms, that we will use in describing the models.

Definition 1.1. *A* distributed system *is an interconnected collection of autonomous nodes, which can be computers, processes or processors [7]. The nodes must at least be equipped with their own private control and they are capable of exchanging information. A more restrictive definition [6] considers a system to be distributed only if the existence of autonomous nodes is* transparent *to users of the system.*

We consider a sequence of processes $P = (p_1, p_2, ..., p_N), S_{p_i}$ the set of the possible states associated with the process p_i, I_{p_i} the initial states of the process $p_i (\forall i \in 1, ..., N)$ and a set of messages M, which can be transmitted (sent and received) between the processes.

In the following, the relations " $>^I$ ", " $>^S$ ", " $>^R$ " associated with the *internal, send*, respectively *receive* events, will be presented.

Definition 1.2. *The binary relation* " $>^I$ " *defined on* $S_{p_i} \times S_{p_i}$*, contains the following pairs of states of the process* p_i*:* (s_{p_i}, u_{p_i})*, which means that an internal event of the process* p_i *is produced, the process* p_i *changes its state from* s_{p_i} *to* u_{p_i}*, but the communication subsystem remains unchanged.*

Definition 1.3. *The ternary relation* " $>^S$ " *defined on* $S_{p_i} \times M \times S_{p_i}$*, contains the following triples:* (s_{p_i}, m, u_{p_i})*, where the process* p_i *executes a* send *event with the message* m *and changes its state from* s_{p_i} *to* u_{p_i}*.*

Definition 1.4. *The ternary relation* " $>^R$ " *defined on* $S_{p_i} \times M \times S_{p_i}$*, contains the following triples:* (s_{p_i}, m, u_{p_i})*, where the process* p_i *executes a* receive *event with the message* $m \in M$ *and changes its state from* s_{p_i} *to* u_{p_i} *.*

Definition 1.5. *The* local algorithm *of a process* p_i *is a quintuple* $(S_{p_i}, I_{p_i}, >^I, >^S, >^R)$*, where all the 5 elements were defined above .*

Definition 1.6. *A* distributed algorithm *is defined as a collection of local algorithms, for the sequence of processes* $P = (p_1, p_2, ..., p_N)$ *and the set of messages* M*. These messages can be exchanged between processes, using pairs of* send-receive *events with the same message* $m \in M$ *.*

Definition 1.7. *A* configuration *is an* $N + 1$*-uple of the following form:* $C = (s_{p0}, ..., s_{p_i}, ..., s_{p_N}, M')$*, where* $\forall i \in \{1, ..., N\}$ s_{p_i} *is a state from* S_{p_i}*, and* $M' \subseteq M$ *is a queue associated with the messages* sent*, but not* received *yet (messages which are in transit [1]).*

Definition 1.8. *An* initial configuration *is a configuration of the following form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M)$*, where* $s_{p_i} \in I_{p_i} \forall i \in \{1, ..., N\}$ *and the messages queue* M *is empty.*

In the following, the *transition relations* "$\rightarrow_{p_i}$" and "$\rightarrow_{p_ip_j}$" on $C \times C$, will be defined:

Definition 1.9. *The transition relation "$\rightarrow_{p_i}$" is defined as the set of the following pairs of configurations:* $((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M_1)$, $(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M_2))$, *where one of the following conditions take place:*

- $(s_{p_i}, u_{p_i}) \in >^I$ *and* $M_1 = M_2$,
- $\exists m \in M$ *such that* $(s_{p_i}, m, u_{p_i}) \in >^S$ *and* $M_2 - m = M_1$.
- $\exists m \in M$ *such that* $(s_{p_i}, m, u_{p_i}) \in >^R$ *and* $M_1 - m = M_2$.

Definition 1.10. *The transition relation "$\rightarrow_{p_ip_j}$", is defined as the set of the following pairs of configurations:* $((s_{p_0}, ..., s_{p_i}, ..., s_{p_j}, ...)$, $(s_{p_0}, ..., u_{p_i}, ..., u_{p_j}, ...))$, *where* $\exists m \in M$, $(s_{p_i}, m, u_{p_i}) \in >^S \wedge (s_{p_j}, m, u_{p_j}) \in >^R$. *In this case, there is an exchange of messages from the process p_i to the process p_j. The process p_i produces the message m, initiating a* send *event, and the process p_j consumes the message, which leads to a* receive *event.*

2. Synchronous and asynchronous transitions systems

Definition 2.1. *The behavior of a distributed algorithm can be described using a* transition system*, defined as a triple $T = (C, \rightarrow, I)$, where:*

- *C is the set of the possible configurations.*
- *"$\rightarrow$" is a binary relation from $C \times C$, called* transition relation*, and it is defined as a reunion of the transition relations "$\rightarrow_{pi}$", associated with the N processes.*
- *I is the set of initial configurations.*

Definition 2.2. *We define a* terminal configuration*, a configuration $c \in C$ which doesn't have a successor configuration (there is no configuration $c_1 \in C$ which verify $c \rightarrow c_1$).*

Definition 2.3. *An* execution *of T is defined as a maximal sequence $E = (c_0, c_1, c_2, ...)$, where $c_0 \in I$ and $\forall i, c_i \rightarrow c_{i+1}$.*

Putting it into words, the execution of a transition system can be defined as a sequence of transitions between related configurations, sequence which can be infinite or ended with a terminal configuration. The transitions are the events associated with processes, events that don't affect only the states of the processes, but also they influence and can be influenced by the queue of messages.

The transitions systems allow modeling the transmission of messages between distributed processes, in two possible ways: synchronous and asynchronous transmissions. We will describe each of the two models textually and formally.

Transition systems with asynchronous message passing. In the asynchronous messages transmission, each *send* event and its corresponding *receive* event are independent. It means that a process initiates a *send* event, without waiting

for a process to receive the produced message. On the other hand, a process initiates a *receive* event, without knowing which process produced the message (see [1,3,7]).

We consider a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 2.4. *The* transition system *induced by the distributed algorithm, using a communication model based on* asynchronous message passing, *is the system* $T = (C, \rightarrow, I)$*, constructed according to the following steps:*

(0) *The local algorithms are:* $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$
(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M')$$

, where $\forall i \in \{1, ..., N\}, s_{p_i} \in S_{p_i}, M' \subseteq M$
(2) *The transition relation* " $\rightarrow$ " *is defined as in the general case of transition systems.*
(3) *The set of initial configurations I is composed of* $N + 1$*-tuples of the form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M')$*, where* $s_{p_I} \in I_{p_i} \forall i \in \{1, ..., N\}$*. The message queue* M' *is empty.*

In this model, only the process which initiates the *send* event, respectively the process which initiated the *receive* event changes its state.

Transition systems with synchronous message passing. In the synchronous message passing, each *send* event and its corresponding *receive* event are coordinated so as they form a single transaction of the system. In this scenario a process can transmit a message only if the destination process is ready to accept the message (see [4,5,7]).

This transition systems model also uses a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 2.5. *The transition system induced by the distributed algorithm, using a communication model based on synchronous message passing, is the system* $T = (C, \rightarrow, I)$*, constructed according to the following steps:*

(0) *The local algorithms are:* $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$
(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}),$$

where $\forall i \in \{1, ..., N\}, s_{p_i} \in S_{p_i}$
(2) *The transition relation* "$\rightarrow$" *is defined as a reunion of the transition relations* "$\rightarrow_{p_i}$" *and* "$\rightarrow_{p_i,p_j}$"*,* $\forall i, j \in \{1, ..., N\}$.
(3) *The set of initial configurations I is composed of N-tuples of the form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N})$*, where* $s_{p_i} \in I_{p_i} \forall i \in \{1, ..., N\}$

In the configuration obtained after processing the transition of the system, both the process which initiated the *send* event and the process which got the *receive* event, change their states.

3. Transition systems with half-synchronous message passing

3.1. **Overview.** This paper proposes a new communication model, which represents an intermediary class between the synchronous and asynchronous communication models. We call this new model *half-synchronized transition system* and we will describe it using both a formal and a textual definition.

This new communication protocol also uses a transition system as the underlying modeling instrument, but the model imposes some constraints on the transition system's entities.

Thus, the message queue M is considered to be limited in space and time. The limitation in space means that the queue has a limited size, which coincides with the maximum number of messages that can be queued.

On the other hand, the limitation in time imposes that a message can be kept in the queue only for a certain period of time.

We consider that the process, which initiated the *send* event, establishes this period of time, because, as a "creator" of the message, it knows how important the message is. If the message is not consumed, during the given period of time, by another process, the message is automatically destroyed by a manager process, associated with the queue.

We represent the messages queue as a 4-tuple: (M, ds, dm, mq), where dm is the maximum size of the queue, ds is the current size of the queue ($dm \geq ds$), and M contains the messages: $M = \{m_1, m_2, ..., m_{ds}\}$. Each message is characterized by a "type" id, a "content" ct and a maximum period of life-time t, $\forall i \in \{1, ..., N\}$, $m_i = (id_i, ct_i, t_i)$. The receiving processes can retrieve the expected messages, using the type information, which could abstract different characteristics of a message, as well as its source or its destinations.

If the first three components of the queue are passive entities, the fourth element, noted with qm (queue manager) is an active process, which controls events like sending or receiving messages, as well as the life- time of the messages (destroying them, when their life-time period expires).

The queue manager have 3 possible states:

- $\mathbf{qm_w}$ waiting for a connection of client process (a sending or receiving process)
- $\mathbf{qm_r}$waken-up by a request of a client process
- $\mathbf{qm_e}$ processing a request of a process or destroying expired messages.

The main difference between the half-synchronized and the previous two models, is that the functionality of the new model comprises the interaction between

the distributed processes and the queue manager and the activity of the queue manager.

Another important difference of this new model, compared with the preceding two, concerns the events of the communication subsystem. Thus, the *send* and *receive* events presume new scenarios and new events like *connect, acknowledge, wait, store, retrieve* and *destroy* are added. These events concern either the interaction between the distributed processes and the queue manager or the activity of the queue manager. They are also associated with binary relations between related configurations of the system.

3.2. **Events describing the interaction between the distributed processes and the queue manager.**

Definition 3.1. *The relation "$\rightarrow^{C}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r))),$$

where the process p_i is executing a connect *event to the message queue: $\exists m \in M$ of type connect-send or connect-receive, $(s_{p_i}, m, u_{p_i}) \in >^S$ and $(qm_w, qm_r) \in >^I$.*

Definition 3.2. *The relation "$\rightarrow^{A}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e))),$$

where the queue manager qm acknowledges the process p_i that it can initiate a send *or* receive *event: $\exists m \in M$ of type acknowledge, $(qm_r, m, qm_e) \in >^S$ and $(s_{p_i}, m, u_{p_i}) \in >^R$.*

Definition 3.3. *The relation "$\rightarrow^{W} p_i$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_r)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w))),$$

where the queue manager qm execute a wait event, announcing the process p_i that the queue is full: $\exists m \in M$ of type wait, $(qm_r, m, qm_w) \in >^S$ and $(sp_i, m, up_i) \in >^R$.

Definition 3.4. *The relation "$\rightarrow^{S}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e))),$$

where the process p_i initiates a send *event with the message m and qm starts processing it: $\exists m \in M$, $(s_{p_i}, m, u_{p_i}) \in >^S$, $(qm_p, m, qm'_e) \in >^R$.*

Definition 3.5. *The relation "$\rightarrow^{R}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_2, ds_2, dm, qm'_e))),$$

where the process p_i initiates a receive *event with the message m and qm starts processing it: $\exists m \in M_1$, $(qm_p, m, qm'_e) \in >^S$ and $(s_{p_i}, m, u_{p_i}) \in >^R$.*

3.3. **Events describing the activity of the queue manager.**

Definition 3.6. *The relation "$\rightarrow^{S}_{qm}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{+}1,dm,qm_w))),$$

where qm store the received message m in the queue: $M_2 - \{m\} = M_1$ and $(qm_e, qm_w) \in >^I$.

Definition 3.7. *The relation " $\rightarrow^{R}_{qm}$ " is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{-}1,dm,qm_w))),$$

where qm retrieves the expected message m from the queue: $M_1 - \{m\} = M_2$ and $(qm_e, qm_w) \in >^I$.

Definition 3.8. *The relation " $\rightarrow^{D}_{qm}$ is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{-}1,dm,qm_w))),$$

where qm destroys a message $m \in M_1$, whose life-time expired ($m = (id, ct, t)$ and $t = 0$): $M_1 - \{m\} = M_2 and (qm_e, qm_w) \in >^I$.

Now, we are ready to define the half-synchronized transition systems.

3.4. **Definition of the half-synchronized transition system.** As in the previous models, we also consider a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 3.9. *The transition system induced by the distributed algorithm A, using a communication model based on half-synchronous message passing, called* half-synchronous transition system *is the system $S = (C, \rightarrow, I)$, constructed according to the following steps:*

(0) *The local algorithms are: $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$*

(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0},...,s_{p_i},...,s_{p_N},(M',ds,dm)),$$

where $\forall i \in \{1,...,N\}$, $s_{p_i} \in S_{p_i}$, $M' \subseteq M$, where the messages queue was defined above

(2) *The transition relation " $\rightarrow$ " is defined as a reunion of a transition relations "$\rightarrow^{C}_{p_i}$", "$\rightarrow^{A}_{p_i}$", "$\rightarrow^{W}_{p_i}$" "$\rightarrow^{S}_{P_i}$", "$\rightarrow^{R}_{p_i}$", "$\rightarrow^{S}_{qm}$", "$\rightarrow^{R}_{qm}$", "$\rightarrow^{D}_{qm}$", associated to the corresponding events, which comprise the functionality of the system.*

3.5. **Space and time limitations scenarios.** The following scenarios describe the half-communication model defined above:

Time limitation.

1.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1 \cup \{m = (id, ct, t)\}, ds + 1, dm, qm_w))$
and
$(s_{p_0}, ..., s_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_w)) \rightarrow^{C}_{p_j}$
$(s_{p_0}, ..., u_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_e)) \rightarrow^{R}_{p_i}$
$(s_{p_0}, ..., v_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm'_e)) \rightarrow^{R}_{qm}$
$(s_{p_0}, ..., z_{p_j}, ..., s_{p_N}, (M_2, ds_2 - 1, dm, qm_w))$ and $t' < t$,
or

2.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1 \cup \{m = (id, ct, t)\}, ds + 1, dm, qm_w))$
and
$(s_{p_0}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t'), t' = 0\}, ds_2, dm, qm_e)) \rightarrow^{D}_{qm}$
$(s_{p_0}, ..., s_{p_N}, (M_2, ds_2 - 1, dm, qm_w))$.

The first scenario refers to the case where a *send* event has a corresponding *receive* event with the same message $m \in M$.

In the following, we will enumerate the sequence of transitions, described formally above. The process p_i initiates a *connect* event to the queue, the queue manager returns an *acknowledge* that the queue can store new messages, the process p_i *sends* its message m (whose life-time is t) and continues its job and the queue manager *stores* the received message in the queue. Later ($t' < t$), a process p_j initiates a *connect* event, and after being *acknowledged* by the queue manager, it initiates a *receive* event with the message m, and the queue manager retrieves this message from the queue.

The second scenario differs from the first, in that the message m is not retrieved from the queue for a receiving process, hence the queue manger destroys the message, when its life-time period expires.

Space limitation.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, s_{p_j}, ..., s_{p_N}, (M_1, dm, dm, qm_r)) \rightarrow^{W}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w))$

The space limitation scenario is similar to the producer-consumer problem. A process p_i tries to *connect* to the queue in order to receive the message m, but the queue manager initiates a *wait* event, telling the process, that the queue is full. So, the process waits until some free space will be available in the queue.

3.6. **An example.** We give an example of a real problem, which can be solved using the half-synchronous model. The problem concerns the activity -supplying-storing-selling merchandise- of a store of perishable nutriments.

In order to give a formal model to this problem, we consider only two processes: $P = (p_1, p_2)$, where p_1 is a "supplier" and p_2 is a "buyer". The messages set M contains the products $prod_1$, $prod_2$, ... (for simplicity, we consider that $prod_i$ includes a given quantity of the product i) and a subset M' of internal, protocol messages. Hence, $M = \{prod_1, prod_2, ...\} \cup M'$. The configurations, used in this particular model, take the following form: $(s_{p_1}, s_{p_2}, (M_1, dc, dm, qm))$, where s_{p_1} and s_{p_2} are the current states of the "supplier", respectively "buyer" processes, M_1 contains the products from the store, dc is the number of the products and dm is the maximum capacity of the store and qm is the manager & seller of the store.

Now, let's detail the two scenarios starting from the general case, for the "store model":

Time limitation

1. $(s_{p_1}, s_{p_2}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(z_{p_1}, s_{p_2}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(z_{p_1}, s_{p_2}(M_1 \cup \{m = (id, ct, 20d)\}, ds + 1, dm, qm_w))$
and $(s_{p_1}, s_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_w)) \rightarrow^{C}_{p_j}$
$(s_{p_1}, u_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_1}, v_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_e)) \rightarrow^{R}_{p_i}$
$(s_{p_1}, v_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm'_e)) \rightarrow^{R}_{qm}$
$(s_{p_1}, z_{p_2}, (M_2, ds_2 - 1, dm, qm_w))$ and $t' < t$,
or

2. $(s_{p_1}, s_{p_2}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(z_{p_1}, s_{p_2}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(z_{p_1}, s_{p_2}, (M_1 \cup \{m = (id, ct, 20d)\}, ds + 1, dm, qm_w))$
and
$(z_{p_1}, s_{p_2}, (M_2 \cup \{m = (id, ct, 0))\}, ds_2, dm, qm_e)) \rightarrow^{D}_{qm}$
$(z_{p_1}, s_{p_2}, (M_2, ds_2 - 1, dm, qm_w))$.

The first scenario corresponds to the situation when a provided product m is bought, before its life-time period expires. The supplier offers a product m to the store, which expires after $t = 20$ days. The communication protocol (connect-acknowledge-send-store, respectively connect-acknowledge-receive- retrieve), between the supplier p_1 or the buyer p_2 and the store manager qm, conforms to the general case, but with particular participants.

The second scenario models the situation when the availability period for the product m expires before anyone have bought it. So, the store manager throws away this product.

Space limitation
$(s_{p_1}, s_{p_2}, (M_1, dm, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, dm, dm, qm_r)) \rightarrow^{W}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, dm, dm, qm_w))$

Finally, if there is no more free space in the store, the supplier p_1 should wait until someone buys something from the store, in order to make its own offer.

As a final remark, we point out that, this particular model can be enhanced, in order to filter the products offered by the suppliers. In this case, the store manager decides if a provided product suits its needs.

4. Conclusion

The communication subsystem constitutes the main part of any distributed system. The protocols models, used to describe the functionality of the communication software, should be flexible, reusable, adaptable. In the first part, the paper presents the two most important possibilities *synchronous* and *asynchronous-*, of modeling communication between distributed processes, using transitions systems. In the second part, the paper extends the models mentioned above and proposes an intermediary class of models, called *half-synchronized transition systems*, starting from some limitations, in time and space, imposed on the entities involved.

References

[1] Andrews G.R. *Synchronizing Resources*, ACM Transactions on Computer Systems, Number 4, October, 1981, pp. 305-330
[2] Boian F.M., *Programare distribuită în Internet*, Editura Albastra, 1998
[3] Fred B. Schneider, *Synchronization in Distributed Programs*, ACM Transactions on Computer Systems, Volume 4, Number 2, April, 1992, pp. 125-148
[4] Lamport L., *Time, clocks, and the ordering of events in a distributed system*, Communications of the ACM, 21, 7, july 1978, pp. 125-133
[5] Spirakis G.P. *Real-Time Synchronization of Interprocess Communications*, ACM Transactions on Computer Systems, Volume 6, Number 2, April, 1994 pp. 215-238
[6] Tannenbaum A.S. *Distributed Operating Systems*, Prentice Hall, 1995
[7] Tel G. *Introduction to Distributed Algorithms*, Cambridge Press, 1994

"Babeş-Bolyai" University of Cluj-Napoca, Department of Computer Science
E-mail address: {florin,cori}@cs.ubbcluj.ro