Applications of membrane systems in distributed systems*

Aneta Binder**, Rudolf Freund, Georg Lojka and Marion Oswald

(Institute of Computer Languages, Vienna University of Technology, Favoritenstr. 9, Wien, Austria)

Abstract Based on the biological model of cell-to-cell communication proposed by A. Rustom et al. (Science, 2004, 303: 1007—1010), we investigate the possibilities to apply P systems with dynamic channels transporting membrane vesicles for describing processes in distributed systems.

Keywords: membrane system, P system, communication process.

Membrane systems were introduced in 1998 by Păun[1] as a parallel distributed model of computation abstracted from cell functioning. In a membrane structure (that can be represented as a tree in P systems or as an arbitrary graph in the more general case of tissue P systems), multisets of objects can evolve according to given evolution rules. The monograph[2] gives an excellent introduction to the whole area of P systems. For the actual status of research we refer the reader to the P systems webpage[3].

Like in the biological model of cell-to-cell communication proposed by Rustom et al.[4], in [5] (like in tissue P systems, e.g. see [6]), cells able to dynamically form connections (channels or nanotubes in the sense of [4] and [7]) between them according to specific constraints, possibly relying on some attributes assigned to the membranes of the cells as well as on their contents, were considered. In contrast to other models of P systems, these multisets of elementary objects are transported in membrane vesicles through nanotubes (channels). The transport of a membrane vesicle through a nanotube between two cells may depend on the contents of the membrane vesicle itself as well as on the objects contained in the two cells connected by this nanotube, on the specific attributes assigned to the cell membranes and the membrane of the vesicle to be transported through the nanotube as well as on the specific state of the nanotube (channel).

To model these biological systems described by Rustom et al.[5] P systems with dynamic channels transporting membrane vesicles were considered to work in the asynchronous mode (an arbitrary number of rules that do not interfere with each other can be carried out in parallel in one derivation step) or in the sequential mode (exactly one rule is carried out in one derivation step); for an overview on P systems working in the asynchronous or in the sequential mode see [8]. On the other hand, for simulating models of distributed systems and mobile software agents we can also use P systems with dynamic channels transporting membrane vesicles working in the maximally parallel mode (which means that as many processes as possible are carried out in parallel). In this paper we continue the work started by Binder et al.[1)][9] and discuss the potentials of this model of P systems (membrane systems) for modelling and simulating (processes in) distributed systems. P systems for modelling algorithms in distributed computing had already been considered in [10] together with the description of a program for simulating some examples of that variant of P systems, and in [11] a variant of P systems called client-server P systems for modelling molecular processes was described.

After pointing out some features of distributed systems in the second section, in the third section we describe the model of P systems with dynamic channels transporting membrane vesicles (which were shown to be computationally complete in [5]). The potentials of this model of P systems (membrane systems) for modelling and simulating (processes in) distributed systems are discussed in the fourth section. A short summary concludes the paper.

* The work of Marion Oswald is supported by FWF-project T225-N04

** To whom correspondence should be addressed. E-mail: ani@emcc.at

1) Binder A and Lojka G. Diploma thesis, draft, 2006

1 Elements of distributed systems

A distributed system consists of a collection of autonomous, geographically dispersed computing entities (e.g. see [12]) which are equipped with suitable software and are connected by some suitable communication medium (LAN, WAN or the Internet); the software enables the single computers (components) of the network to coordinate their activities and share their resources of the system-hardware, software and data.

Various hardware and software architectures are commonly used for distributed computing: client-server, 3-tier architecture (the client intelligence is moved to a middle tier so that stateless clients can be used), n-tier architecture (typically used for web applications forwarding their requests to other enterprise services), tightly coupled (clustered, a set of highly integrated machines runs the same process in parallel, subdividing the task in parts finally put back together to make the final result), peer-to-peer (all responsibilities are uniformly distributed among all machines).

One of the key features of distributed systems is the sharing of resources[13]; there are two types of models of distributed systems—the client-server model (server processes act as a resource manager for a collection of resources of a given type and client processes perform a task requiring access to some shared hardware and software resources) and the object-based model (each shared resource is viewed as an object; objects are uniquely identified and may be moved anywhere in the network without changing their identities). In what follows we will only consider the client-server model.

The components of a distributed system are both logically and physically separated[13]; they must communicate in order to interact. Communication between a pair of processes involves operations in the sending and the receiving processes which together result in:

① the transfer of data from the sending process to the receiving process;

② the synchronization of the receiving process with the sending process in such a way that both processes are prevented from continuing until the other process makes an action that frees them both from this prevention.

The two basic programming primitives send and receive together perform message passing actions between a pair of processes. Each message passing action involves the transmission by the sending process of a set of data values (a message) through a specified communication mechanism (a channel or a port) and the acceptance by the receiving process of the message. The mechanism may be synchronous or blocking, meaning that the sender waits after transmitting the message until the receiver has performed a receive operation or it may be asynchronous or non-blocking meaning that the message is placed in a queue of messages waiting for the receiver to accept them and the sending process can proceed immediately.

The most commonly used patterns of communication in distributed systems are the client-server model for communication between pairs of processes and the group multi-cast communication model for communication between groups of cooperating processes. In what follows we restrict ourselves to the client-server model.

The client-server communication model is oriented towards service provision. An exchange consists of the transmission of a request from a client process to a server process, the execution of the request by the server and the transmission of a reply to the client.

Send and receive operations:

Single message passing can be supported by two message communication operations: send and receive, defined in terms of destination and messages. In order for one process to communicate with another one, the first process sends a message (a sequence of data items) to a destination and the second process at the destination receives the message. This activity involves the communication of data from the sending to the receiving process and may involve the synchronization of the two processes.

Synchronous and asynchronous communication:

A queue is associated with each message destination. Sending processes add messages to queues and receiving processes remove messages from them. The communication between the sending and the receiving process may be either synchronous or asynchronous.

In the synchronous form of communication, the sending and receiving processes synchronize at every message. In this case, both the send and receive are

blocking operations. Whenever send is issued, the sending process is blocked until the corresponding receive is issued. On the other hand, whenever receive is issued, the process blocks until a message arrives.

In the asynchronous mode of communication, the use of the send operation is non-blocking such that the sending process is allowed to proceed as soon as the message has been copied to a local buffer and the transmission of the message proceeds in parallel with the sending process. The receive operation can have blocking and non-blocking variants. In the non-blocking variant, the receiving process proceeds with its program after issuing a receive operation which provides a buffer to be filled in the background, but it must separately receive a notification that its buffer has been filled (either by polling or by interrupt).

2 P systems with dynamic channels transporting membrane vesicles

In this section we describe (a special variant of) the general model of P systems as introduced in [5]. Intuitively, we consider cells enclosed by a membrane which allows for communication with the environment as well as for dynamically forming connections (channels or nanotubes in the sense of Rustom et al. in [4]) between cells that transport (multisets of) objects enclosed in a membrane (vesicle). We assume the reader to be familiar with the basic elements of formal language theory (e.g. see [14]) and membrane systems (P systems, e.g. see [2]).

A P system with dynamic channels transporting membrane vesicles (in what follows we shall use the notion P system only) Π is a construct

$$(O, O_T, Q, F, I, R)$$

where O is the set of objects; $O_T \subseteq O$ is the set of terminal objects; Q is the set of states for the channels between cells; F is the set of attributes assigned to a cell or a vesicle; I specifies the initial configuration of the system; R is a set of different types of rules of one of the following forms: evolution rules for objects in a cell, evolution rules for objects in a vesicle, communication rules for exchanging (multisets of) objects between a vesicle in a cell and the surrounding cell, rules initializing (opening) a channel (nanotube) between two cells, rules transporting a vesicle through a channel (nanotube) between two cells, rules closing a channel (nanotube) between two cells, rules generating a new vesicle in a cell, rules for including a vesicle in another vesicle, rules eliminating a vesicle in a cell, rules eliminating the membrane of a vesicle and expelling its contents into the surrounding cell.

We do not go into the details of specifying rules allowing us to dynamically change the components and the structure of the network. Moreover, we should like to point out that all the sets in the general model described above need not be finite, e.g. the set of objects may be the set of strings over a given alphabet, and the set of attributes may be infinite, too. If the set of rules is infinite, we assume that given an instance of the P system, we are able to effectively list the rules applicable to this instance and to decide whether a set of rules chosen so far is maximal or not.

The P system Π may work in different derivation modes: in the maximally parallel mode, for each derivation step we choose a subset of the rules in R that cannot be extended any more; in the asynchronous mode, an arbitrary number of rules is applied in parallel; in the sequential mode, exactly one rule is applied.

The general model of P systems with dynamic channels transporting membrane vesicles introduced above is very powerful from a theoretical point of view, i.e. not all possible features are needed to obtain computational completeness[5]. Moreover, we may use these P systems for different tasks, e.g. as generating devices or as accepting devices (P automata), but also for modelling different processes carried out by other devices; here we will focus on this dynamic aspect. For our purposes, even the sequential derivation mode is suitable. In what follows, we shall give a more detailed description of the rules needed in the examples elaborated in the succeeding section.

2.1 Cells, vesicles, communication through channels

Here we explain the different objects and processes which will be used in our model. The objects can be ① cells, which can act as clients or servers, ② vesicles, which are the packages of data sent through channels between different cells, and ③ channels, which are the communication medium.

Cells can be described by their labels and attributes. The list of attributes of a cell can be defined in form of the context-free grammar G_1 with

$$G_1 = (V, T, P, CellAttributes),$$

V = {*CellAttributes*, *Name*, *Path*, *State*}
∪ {*Domain*, *LowerChar*, *UpperChar*, *Num*},
T = {busy, free, A, ⋯, Z, a, ⋯, z, 0, ⋯, 9, _, ., (,)},
P = {*CellAttributes* → (*Name*.*Path*.*State*),
State → busy | free,
Path → {.*Domain*},
Domain → *Name*,
Name → *LowerChar* | *UpperChar* {*LowerChar* | *UpperChar* | *Num*},
LowerChar → a | ⋯ | z,
UpperChar → A | ⋯ | Z,
Num → 0 | ⋯ | 9}.

For better readability, terminal objects are underlined in the definitions of the context-free grammars G_1 and G_2.

The grammar specifies the attributes, which consist of *Name*, *Path*, and *State*. *Name* is the identifier of the cell and *Path* gives the current membership to a network indicated by its domain name. The attribute *State* defines whether a cell is free (free) for creating a new channel to another cell or not (busy).

In addition to attributes, each cell has a *label* describing its function in the network, which is taken from the following finite set of reserved strings:

CellLabels = {server, client}

Similar to cells, vesicles can be characterized by their attributes and labels. Attributes can be represented by the context-free grammar G_2 with

G_2 = (*V*, *T*, *P*, *V esicleAttributes*),
V = {*V esicleAttributes*, *V Name*, *V Path*}
∪ {*Domain*, *LowerChar*, *UpperChar*, *Num*},
T = {A, ⋯, Z, a, ⋯, z, 0, ⋯, 9, _, ., [,], (,)},
P = {*V esicleAttributes* → (*V Name*.*V Path*),
V Path → [*Name*{.*Domain*}] {.[*Name*{.*Domain*}]},
Domain → *Name*,
V Name → *Name*,
Name → *LowerChar* | *UpperChar* {*LowerChar* | *UpperChar* | *Num*},
LowerChar → a | ⋯ | z,
UpperChar → A | ⋯ | Z,
Num → 0 | ⋯ | 9}.

The grammars G_1 and G_2 differ in the definition of *Path*. A cell contains only the domain name in this attribute, but the *V Path* of a vesicle implies the whole name *Name* and the domain of the sending cell concatenated with a dot symbol such that the receiving cell knows where the vesicle is coming from.

The label of a vesicle describes the kind of content enclosed by its membrane and is taken from the following finite set of reserved strings:

V esicleLabels = {data, resdescr, invoc, result}

A vesicle with the label invoc contains two vesicles one containing data (labelled as data) and the other one containing a resource description (labelled as resdescr). This corresponds to an invocation from one computer to another one. If the invocation needs some computation, the result is transmitted back as a vesicle labelled result.

Channels are connections dynamically formed between cells (sender and receiver) allowing for the transport of vesicles. The sending cell tries to open a channel to the receiving cell but has to take care that only one uni-directional channel between a cell and another one can be established at the same time. Another important restriction is the limitation of the quantity of channels which can co-exist, given by the number n. Moreover, these n channels at the beginning are available being in the state closed.

$$Channel^{(ChannelState)}_{Name\_of\_Sender:Name\_of\_Receiver.i}$$

The state of a channel (*ChannelState*) can be set to: ① request: This state is set by the sending cell which has the intention to create a channel. ② open: The channel has been opened. ③ transport: The vesicle is already on its way through the channel. ④ sent: After the vesicle has reached the destination cell, the receiver sets the channel state to sent. ⑤ closed: At the end of the transportation process the channel is closed by the sending cell.

The label of a channel consists of three components: the name of the sender, the name of the receiver, and the number of the channel between sender and receiver (i).

2.2 Rule definitions

In this subsection we describe the rules which are needed for creating and deleting channels, generating and transporting vesicles, and dissolving the membranes of these vesicles to expel their contents. Moreover, for all the following rules we assume that

CellLabel ∈ *CellLabels* and *V esicleLabel* ∈ *V esicleLabels*.

2.2.1 Rules for communicating membrane vesicles

The process of communicating vesicles between two different cells in our model cannot be described with only one rule—we need three rules which define the opening of a channel, the transport of the vesicle through the open channel and finally the closing of this channel after the successful communication. The attribute *Path* contains the information about the network domain in which the cells are situated.

The first rule is called $R_{c,c,i}$; it consists of three subrules which are defined as follows:

$$R_{c,c,i.1}: channel_{x.y.i}^{(closed)}[\alpha]_{client}^{(Name,Path,free)} \rightarrow channel_{x.y.i}^{(request)}[\alpha']_{CellLabel}^{(Name,Path,busy)}$$

$$R_{c,c,i.2}: channel_{x.y.i}^{(request)}[\beta]_{CellLabel}^{(Name,Path,free)} \rightarrow channel_{x.y.i}^{(open)}[\beta']_{CellLabel}^{(Name,Path,free)}$$

$$R_{c,c,i.3}: channel_{x.y.i}^{(open)} \cdot [\alpha''[\varphi]_{V esicleLabel}^{(V Name,V Path)}]_{CellLabel}^{(Name,Path,busy)} \rightarrow channel_{x.y.i}^{(transport)}([\varphi]_{V esicleLabel}^{(V Name,V Path)}) \cdot [\alpha''']_{CellLabel}^{(Name,Path,free)}$$

where x is the name of the sender, y is the name of the receiver, i is the number of the channel, *Path* contains the domain name followed by a dot, *V Path* contains the name of the sending cell followed by a dot and the domain name.

First the sender cell wants to open a channel with number i and sets the state of the channel to request, such that the receiver cell knows about the intention of an incoming communication, and the sender sets its own state to busy. If the cell state of the receiving cell is free, it sets the state of the channel to open and at the same time changes its own state to busy. As long as the cells are busy, the cells are inhibited to request another channel at the same time. After the sending cell has recognized that the channel has been opened, it sets its cell state to free, whereafter the receiving cell sets back its state to free again, too.

Theoretically, the process steps of the receiving cell seeing the request for opening a specific channel, getting busy, setting the channel to the state open and to change back from state busy to free again, can be captured by only one rule, i. e. $R_{c,c,i.2}$. Moreover, the sending cell when seeing the channel to have become open, not only changes its state back from busy to free, but with the Rule $R_{c,c,i.3}$ also changes the state of the channel to transport at the same time putting a vesicle into the channel.

As in distributed systems, not all connections between cells (computers) are allowed. For example, in client-server systems, clients cannot communicate directly with each other. The communication is done via the server in the network. To apply the rules above, the label of the cells plays an important rôle because it indicates the function of the cells (client, server). There are three different types of connections: ① Client to Client: it is not allowed. ② Client to Server: In this case it is necessary that both, client and server, are in the same domain (same *Path*). ③ Server to Server: This type of connection is always possible. The attribute *Path* can contain the same or different domain names.

The next rule for communicating membrane vesicles is $R_{c,c,t}$ and describes the transport of the vesicle through the established channel and is defined as follows:

$$R_{c,c,t}: channel_{x.y.i}^{(transport)}([\varphi]_{V esicleLabel}^{(V Name,V Path)}), [\alpha]_{CellLabel}^{(Name,Path,State)} \rightarrow channel_{x.y.i}^{(sent)}[\beta'[\varphi']_{V esicleLabel}^{(V Name,V Path)}]_{CellLabel}^{(Name,Path,State)}$$

After the rule $R_{c,c,i.3}$ has been applied successfully, the state of the channel is transport which means that the vesicle now is inside the channel. When the vesicle arrives at the receiver, the receiver sets the channel state to sent so that the sending cell knows that the transmission has been successful. The channel is closed by applying the following rule:

$$R_{c,c,d}: channel_{x.y.i}^{(sent)}[\beta'[\varphi']_{V esicleLabel}^{(V Name,V Path)}]_{CellLabel}^{(Name,Path,State)} \rightarrow channel_{x.y.i}^{(closed)}[\beta'[\varphi']_{V esicleLabel}^{(V Name,V Path)}]_{CellLabel}^{(Name,Path,State)}$$

2.2.2 Rules to manipulate membrane vesicles

Generating membrane vesicles Before it is possible to send a vesicle to another cell, it is necessary to generate it by the application of the following rule:

$$R_g: [\alpha]_{CellLabel}^{(Name,Path,State)} \rightarrow [\alpha'[\varphi]_{V esicleLabel}^{(V Name,V Path)}]_{CellLabel}^{(Name,Path,State)}$$

There are three different cases of creating the vesicle content φ-eventually to-gether with the remaining

contents of the generating cell:

① $generate$: $[\alpha\ \alpha'']^{(Name, Path, State)}_{CellLabel}$
$\rightarrow [\alpha'\ \alpha''[\varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$

② $copy$: $[\alpha\ \alpha'']^{(Name, Path, State)}_{CellLabel}$
$\rightarrow [\alpha'\ \alpha''[\alpha''\ \varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$

③ $move$: $[\alpha\ \alpha'']^{(Name, Path, State)}_{CellLabel}$
$\rightarrow [\alpha'[\alpha''\ \varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$

These three variants of creating a vesicle are not sufficient to send the required part of the requested data with the aid of membrane vesicles. Therefore we extend our model with a special method to create the contents of a vesicle by using the computable functions f and g:

$$[\alpha]^{(Name, Path, State)}_{CellLabel} \xrightarrow{f,g} [\alpha'[\varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$$

with the effect of yielding

$$[\alpha' f_\alpha(\alpha'')[\varphi\ g_\alpha(\alpha'')]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$$

from $[\alpha\ \alpha'']^{(Name, Path, free)}_{CellLabel}$.

We should like to mention that we are not interested in the details of the representation of data being sent through channels to other cells.

Dissolving vesicles To expel the whole contents of a vesicle we dissolve its membrane. The rule to achieve this is defined as follows:

$$R_\delta: [\alpha[\varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel} \rightarrow [\alpha'\ \varphi']^{(Name, Path, State)}_{CellLabel}$$

Deleting membrane vesicles If the vesicle is not needed anymore, it can be deleted as a whole with the following rule:

$$R_d: [\alpha[\varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel} \rightarrow [\alpha']^{(Name, Path, State)}_{CellLabel}$$

2.2.3 Rule to communicate between a membrane vesicle and its surrounding cell

We have already defined the communication between two different cells via channels. Furthermore, there is another way of communication in our model where it is possible to exchange some contents between the membrane vesicle and its surrounding cell. This rule is defined as follows:

$$R_{c,v}: [\alpha[\varphi]^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel} \rightarrow [\alpha'[\varphi']^{(V\,Name, V\,Path)}_{VesicleLabel}]^{(Name, Path, State)}_{CellLabel}$$

3 Communication processes

In this section we show how we can specify communication processes in distributed systems within our framework of P systems with dynamic channels transporting membrane vesicles (P systems for short).

3.1 Communication between cells

The communication in the model of P systems used in this paper is based on the scientific findings from the field of cell biology described in [4]. Vesicles can be seen as packages of information transported from one cell to another one. The communication medium is represented by channels created by one of the communicating cells (the sender). If such a channel exists, the information packed into a vesicle can be sent to the receiving cell. The sending process and the receiving process are controlled by the setting of channel states, and they are not happening in an exactly synchronous or asynchronous way, but as a mixture of both. Like in the synchronous mode there is some kind of "handshake" indicated by the states. Nevertheless, both cells can proceed their computations independently (like in the asynchronous mode).

In the following example, we describe in more detail how two cells can communicate with each other:

The client cell with the attribute *Name* A wants to send some information to the server cell with *Name* B. Thus cell A has to pack the data into a vesicle and has to name it by setting the vesicle attribute *V Name* and the label which identifies the type it should be. Moreover, it has to set the vesicle attribute *V Path* to its own *Name* and *Path*, concatenated with a dot. In what follows, we take Ves1 for *V Name* and A.Net1 for *V Path*.

To be able to transmit the data, a channel to cell B has to be requested by cell A. At first, cell A, being in the state free, has to request the opening of the channel by setting the state of the channel to request (Fig. 1). Cell B now permits the opening of the channel or refuses it (this means that the membranes of the cells can or cannot connect to each other). This permission depends on the state of cell B which can be free or busy. If the case B is free, B can permit the opening of the channel $channel_{A.B.i}$ and the channel gets the state open (Fig. 2).

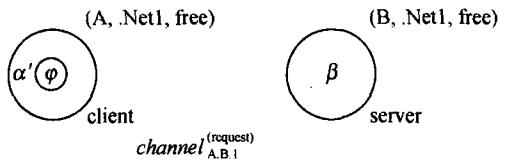

Fig. 1. Requesting the opening of a closed channel.

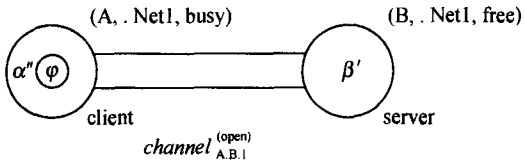

Fig. 2. The opening of a channel.

Cell A now can send the vesicle, and the state of the channel changes to transport (Fig. 3).

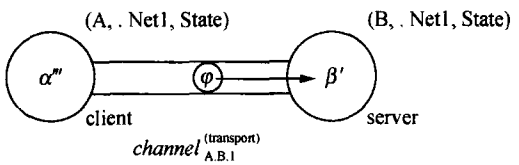

Fig. 3. Transporting the vesicle.

After the package of information—the vesicle—has reached the destination cell B (Fig. 4), B changes the channel state to sent. The channel now can be closed (the membranes of the cells disconnect), indicated by the channel state closed (Fig. 5).

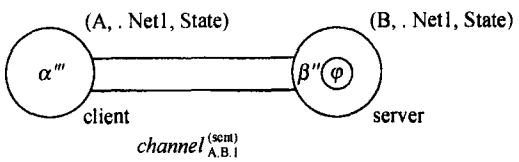

Fig. 4. Reaching the receiver cell.

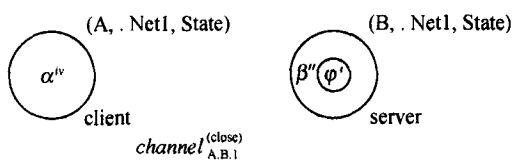

Fig. 5. Closing the channel.

At the same time, cell B can dissolve the membrane of the received vesicle to get the information contained in it.

This process of communication between cells can be described by applying the following sequence of rules:

① Creating the vesicle containing data

$$[\alpha]^{A,.Net1,free}_{client} \xrightarrow{f,g} [\alpha'[\varphi]^{Ves1,[A.Net1]}_{data}]^{A,.Net1,free}_{client}$$

② Opening the channel and transporting the data vesicle

$$channel^{(closed)}_{A.B.1}[\alpha']^{(A,.Net1,free)}_{client} \rightarrow channel^{(request)}_{A.B.1}[\alpha'']^{(A,.Net1,busy)}_{client};$$

$$channel^{(request)}_{A.B.1}[\beta]^{(B,.Net1,free)}_{server} \rightarrow channel^{(open)}_{A.B.1}[\beta']^{(B,.Net1,free)}_{server};$$

$$channel^{(open)}_{A.B.1}[\alpha''[\varphi]^{(Ves1,[A.Net1])}_{data}]^{(A,.Net1,busy)}_{client} \rightarrow channel^{(transport)}_{A.B.1}([\varphi]^{(Ves1,[A.Net1])}_{data}), [\alpha''']^{(A,.Net1,free)}_{client};$$

$$channel^{(transport)}_{A.B.1}([\varphi]^{(Ves1,[A.Net1])}_{data})[\beta']^{(B,.Net1,free)}_{server} \rightarrow channel^{(sent)}_{A.B.1}[\beta''[\varphi]^{(Ves1,[A.Net1])}_{data}]^{(B,.Net1,free)}_{server};$$

$$channel^{(sent)}_{A.B.1}[\alpha''']^{(A,.Net1,free)}_{client} \rightarrow channel^{(closed)}_{A.B.1}[\alpha^{iv}]^{(A,.Net1,free)}_{client}.$$

③ Dissolving the membrane of the vesicle and expelling its contents

$$[\beta''[\varphi]^{(Ves1,[A.Net1])}_{data}]^{(B,.Net1,free)}_{server} \rightarrow [\beta'''\varphi']^{(B,.Net1,free)}_{server}.$$

3.2 Communication in a client-server system

In the following example we describe a communication in a client-server system using P systems as a model for distributed systems.

We here consider a client-server system which contains four clients (B, C, D, E), and the server A. All are members of the domain Net1. Let us assume that a client B has a task (a computation) to perform but has not got the resources needed for fulfilling the task. Hence, cell B sends an invocation to the server including the resource description and the data required for this computation (Fig. 6).

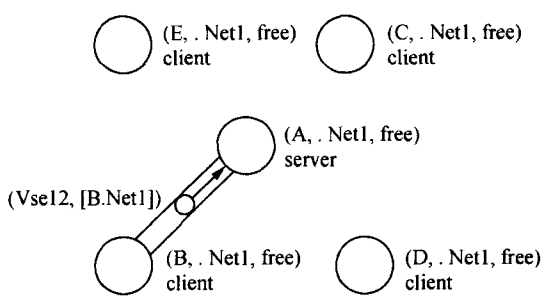

Fig. 6. Sending an invocation vesicle (labelled invoc) to the server.

Moreover, the required resources are in the pos-

session of two other clients: cell C and D. Because the direct communication between two clients in clientserver systems is not possible, the server has to take control of the following computation and communication process flow. Thus, it disposes of a special resource allocation table, in which all resources of the present clients in the network are listed. In what follows the server subdivides the task into two subtasks and sends the invocation including the data to each cell disposing of the needed resources for each subtask (Fig. 7).

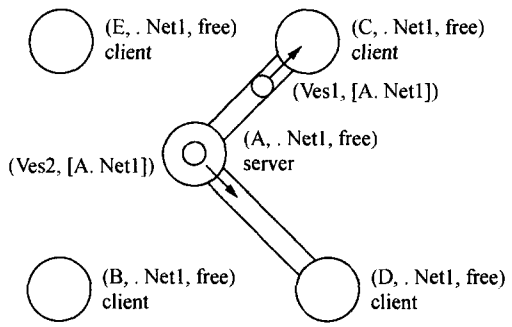

Fig. 7. Dividing the task into two subtasks and sending two vesicles labelled invoc to C and D.

After the partial results of the computation have become available for the server (Fig. 8), it combines them to the final result and sends it forward to cell B (Fig. 9).

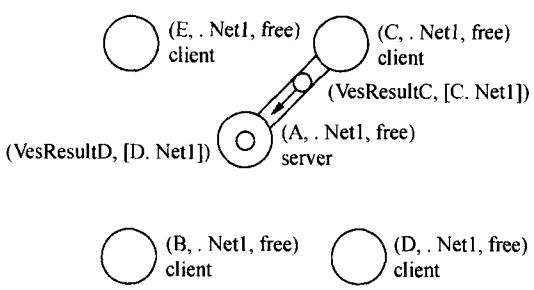

Fig. 8. Receiving the partial results from C and D, enclosed by the vesicles labelled result.

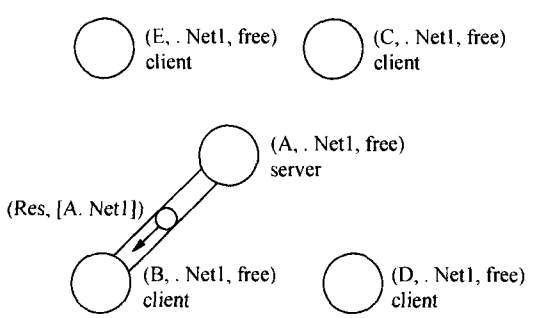

Fig. 9. Transmitting the result back to B.

Here we do not go into the details of the resource allocation, its management and the sharing of resources in a distributed system.

This process of performing the given task and the communication in the client-server system can be described by applying the following sequence of rules:

① Creating the vesicle containing data and resource description by the client B At first, cell B has to create two vesicles containing the data and the information about the needed resources for the computation.

$$[\beta]^{(B,.\,Net1,\,free)}_{client} \rightarrow [\beta'[\delta]^{(Ves1,\,[B.\,Net1])}_{data}]^{(B,.\,Net1,\,free)}_{client};$$

$$[\beta']^{(B,.\,Net1,\,free)}_{client} \rightarrow [\beta''[\rho]^{(Ves2,\,[B.\,Net1])}_{resdescr}]^{(B,.\,Net1,\,free)}_{client};$$

$$[\beta'']^{(B,.\,Net1,\,free)}_{client} \rightarrow [\beta'''[\varphi]^{(Ves12,\,[B.\,Net1])}_{invoc}]^{(B,.\,Net1,\,free)}_{client}.$$

After this, the generated vesicles, $Ves1$ and $Ves2$, have to be packed into the invocation vesicle $Ves12$.

$$[\varphi]^{(Ves12,\,[B.\,Net1])}_{invoc}[\delta]^{(Ves1,\,[B.\,Net1])}_{data} \rightarrow [\varphi'[\delta]^{(Ves1,\,[B.\,Net1])}_{data}]^{(Ves12,\,[B.\,Net1])}_{invoc};$$

$$[\varphi']^{(Ves12,\,[B.\,Net1])}_{invoc}[\rho]^{(Ves2,\,[B.\,Net1])}_{resdescr} \rightarrow [\varphi''[\rho]^{(Ves2,\,[B.\,Net1])}_{resdescr}]^{(Ves12,\,[B.\,Net1])}_{invoc}.$$

We here use a rule more complex than the rules used so far, where a vesicle includes another one.

② Opening the channel and transporting the invocation vesicle to the server A The client B opens a free channel (in this case with number 1) to send the invocation vesicle to the server cell A.

$$channel^{(close)}_{B.A.1}[\beta''']^{(B,.\,Net1,\,free)}_{client} \rightarrow channel^{(request)}_{B.A.1}[\beta^{iv}]^{(B,.\,Net1,\,busy)}_{client};$$

$$channel^{(request)}_{B.A.1}[\alpha]^{(A,.\,Net1,\,free)}_{server} \rightarrow channel^{(open)}_{B.A.1}[\alpha']^{(A,.\,Net1,\,free)}_{server};$$

$$channel^{(open)}_{B.A.1}[\beta^{iv}[\varphi'']^{(Ves12,\,[B.\,Net1])}_{invoc}]^{(B,.\,Net1,\,busy)}_{client} \rightarrow channel^{(transport)}_{B.A.1}([\varphi'']^{(Ves12,\,[B.\,Net1])}_{invoc})[\beta^{v}]^{(B,.\,Net1,\,free)}_{client};$$

$$channel^{(transport)}_{B.A.1}([\varphi'']^{(Ves12,\,[B.\,Net1])}_{invoc})[\alpha']^{(A,.\,Net1,\,free)}_{server} \rightarrow channel^{(sent)}_{B.A.1}[\alpha''[\varphi''']^{(Ves12,\,[B.\,Net1])}_{invoc}]^{(A,.\,Net1,\,free)}_{server};$$

$$channel^{(sent)}_{B.A.1}[\beta^{v}]^{(B,.\,Net1,\,free)}_{client} \rightarrow channel^{(close)}_{B.A.1}[\beta^{vi}]^{(B,.\,Net1,\,free)}_{client}.$$

③ Dissolving the membrane of the vesicle and expelling its contents After the successful communication with the client B, the server A dissolves the

membrane of the invocation vesicle and in the following the membrane of the vesicle containing the resource description.

$$[\alpha''[\varphi''']_{\text{invoc}}^{(\text{Ves12},[\text{B. Net1}])}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})} \rightarrow [\alpha'''\tilde{\varphi}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})}.$$

④ Creating the vesicles containing the data and resource description by the server A for subtasks to be performed by the clients C and D The server divides the invoked task into two subtasks and creates two new invocation vesicles. Both contain, as a copy, the original data sent from client B and the information about the needed resources for each part of the computation. We should like to mention that the attribute *V Path* for the data vesicle changes its value to [B. Net1]. [A. Net1].

$$[\delta]_{\text{data}}^{(\text{Ves1},[\text{B. Net1}])} \rightarrow [\delta_1]_{\text{data}}^{(\text{Ves1},[\text{B. Net1}].[\text{A. Net1}])} [\delta_2]_{\text{data}}^{(\text{Ves1},[\text{B. Net1}].[\text{A. Net1}])};$$

$$[\tilde{\varphi}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})} \rightarrow [\tilde{\varphi}'[\mu_1]_{\text{invoc}}^{(\text{Ves1},[\text{A. Net1}])}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})};$$

$$[\tilde{\varphi}']_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})} \rightarrow [\tilde{\varphi}''[\mu_2]_{\text{invoc}}^{(\text{Ves2},[\text{A. Net1}])}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})};$$

$$[\tilde{\varphi}'']_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})} \rightarrow [\varphi[\nu_1]_{\text{resdescr}}^{(\text{Rs1},[\text{A. Net1}])}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})};$$

$$[\varphi]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})} \rightarrow [\varphi'[\nu_2]_{\text{resdescr}}^{(\text{Rs2},[\text{A. Net1}])}]_{\text{server}}^{(\text{A},.\,\text{Net1},\text{free})};$$

$$[\mu_k]_{\text{invoc}}^{(\text{Ves}k,[\text{A. Net1}])}[\delta_k]_{\text{data}}^{(\text{Ves}k,[\text{B. Net1}].[\text{A. Net1}])} \rightarrow [\mu_k'[\delta_k]_{\text{data}}^{(\text{Ves}k,[\text{B. Net1}].[\text{A. Net1}])}]_{\text{invoc}}^{(\text{Ves}k,[\text{A. Net1}])}, \quad k \in \{1,2\};$$

$$[\mu_k']_{\text{invoc}}^{(\text{Ves}k,[\text{A. Net1}])}[\nu_k]_{\text{resdescr}}^{(\text{Rs}k,[\text{B. Net1}].[\text{A. Net1}])} \rightarrow [\mu_k''[\nu_k]_{\text{resdescr}}^{(\text{Rs}k,[\text{B. Net1}].[\text{A. Net1}])}]_{\text{invoc}}^{(\text{Ves}k,[\text{A. Net1}])}, \quad k \in \{1,2\}.$$

As the following steps mainly include only repetitions of processes already described by sequences of rules as above, we only give a short description of the remaining processes to finish the task and omit the corresponding sequences of rules.

⑤ Opening of the channel and transporting the invocation vesicles to the clients C and D expecting the partial results of the computation After the server has created the invocation vesicles Ves1 and Ves2, both can be transported through the channels to the clients C and D in the way already described above.

⑥ Dissolving the membrane of the vesicle, expelling its contents and performing of subtasks by the clients C and D After the successful communication, both clients dissolve the membranes of the vesicles and perform the computation of the subtasks. If all the partial results are available, the clients send them back to the server A applying the rules for communication described above. The partial results have to be sent back from the two clients to server A, each of them packed into a vesicle labelled result.

⑦ Creating the vesicle containing the result of the whole computation After the server has got the two partial resulting from the two clients, it assembles them to the final result of the computation and creates a vesicle containing this final result.

⑧ Opening of the channel and transporting the result vesicle to the client B The final result of the computation is sent back to cell B, which was the initiator of the task.

4 Conclusion

In this paper we have used a biologically motivated model of P systems to specify and describe communication processes in distributed systems. P systems with dynamic channels transporting membrane vesicles can model the initialization of a communication channel between two computers and the transfer of data by simulating these processes in our model of P systems as the dynamic opening of a channel between two cells and the transport of membrane vesicles through the channel. In the future, we will not only use this model of P systems to describe communication processes in distributed systems, but also to investigate strategies for resource allocation in grids etc.

References

1 Păun Gh. Computing with membranes. Journal of Computer and System Sciences, 2000, 61(1): 108—114

2 Păun Gh. Membrane Computing. An Introduction. Springer-Verlag, Berlin, 2002

3 The P Systems Web Page: http://psystems.disco.unimib.it

4 Rustom A, Saffrich R, Markovic I, et al. Nanotubular highways for intercellular organelle transport. Science 2004, 303: 1007—1010

5 Freund R and Oswald M. P systems with dynamic channels transporting membrane vesicles. Proc. AROB'05 (International Symposium on Artificial Life and Robotics 2005), February 4—6, 2005, Oita University, Beppu, Oita, Japan, 2005

6 Martin-Vide C, Păun Gh, Pazos J, et al. Tissue P systems. Theor Comp Sci, 2003, 296(2): 295—326

7 Salter RD and Watkins SC. Functional connectivity between immune cells mediated by tunneling nanotubes. Immunity, 2005, 23 (3): 309—318

8 Freund R. Asynchronous P systems. In: (eds: Mauri G, Păun Gh, Zandron C) Proceedings WMC5, Milano, Italy, 2004, 12—28

9 Binder A, Freund R, Lojka G, et al. Membrane systems as a model for distributed computing. In: (ed. Fernau H) Proc. Theorietag, 2005

10 Ciobanu G, Desai R and Kumar A. Membrane systems and distributed computing. In: Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19—23, 2002. Revised Papers, Lecture Notes in Computer Science 2597, Springer, Berlin, 2003, 187—202

11 Ciobanu G, Dumitriu D, Huzum D, et al. Client-server P systems in modelling molecular interaction. In: Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19—23, 2002. Revised Papers, Lecture Notes in Computer Science 2597, Springer, Berlin, 2003, 203—218

12 Berman F, Fox GC and Hey AJG. Grid Computing—Making the Global Infrastructure a Reality. Wiley, 2003

13 Coulouris G, Dollimore J and Kindberg T. Distributed Systems-Concepts and Design. Addison-Wesley, 2002

14 Salomaa A and Rozenberg G. Handbook of Formal Languages. Springer, Berlin, 1997