DEPÓSITO LEGAL ppi 201502ZU4666
Esta publicación científica en formato digital
es continuidad de la revista impresa
ISSN 0041-8811
DEPÓSITO LEGAL pp 76-654

Revista de la Universidad del Zulia

Fundada en 1947
por el Dr. Jesús Enrique Lossada

Ciencias Exactas Naturales y de la Salud

Año 11 N° 30
Mayo - Agosto 2020
Tercera Época
Maracaibo-Venezuela

A method for designing the logical structure of a distributed telecommunication environment

G. S. Vasilyev *
V. T. Eremenko **
O. R. Kuzichkin ***
A. V. Eremenko ****
D. I. Surzhik *****
S. V. Eremenko ******

ABSTRACT
The article presents a methodology for the development of information exchange tools based on the apparatus of hierarchical Petri nets, characterized by the possibility of simultaneous events and allows solving the problem of correctness analysis and verification of protocols. Protocol objects and access points are represented on the basis of the Petri nets apparatus: the simplest object of the reception/transmission protocol, the queue object, a single object and a trivial object. It describes the construction of complex structures of objects and operations on them, the management of timeouts based on the timer object, the specification of protocol objects, the specification of protocols and levels of logical structure. The proposed method is relevant for the analysis of complex communication systems of large size, which are mobile ad-hoc networks with a large number of nodes, characterized by topology instability due to unstable communication channel characteristics and high nodes mobility, and especially flying ad-hoc networks of aircraft (FANET) and flying sensor networks (FSN).

KEYWORDS: distributed system; transport protocol; Petri net; flying ad-hoc network; FANET; flying sensor network; FSN.

* Belgorod State University, Belgorod, 308015, Russia (E-mail: Belova-t@ores.su, https://orcid.org/0000-0003-1681-5223).

** Orel State University, Orel, 302026, Russia (E-mail: ssv@ores.su, https://orcid.org/0000-0002-5650-9151).

*** Belgorod State University, Belgorod, 308015, Russia (E-mail: eav@ores.su, https://orcid.org/0000-0003-0817-223X)

**** Orel State University, Orel, 302026, Russia (E-mail: info@ores.su, https://orcid.org/0000-0003-3119-9123).

***** Vladimir State University, Vladimir, 600000, Russia (E-mail: russia@prescopus.com, https://orcid.org/0000-0002-0101-3503).

****** Orel State University, Orel, 302026, Russia (E-mail: editor@ores.su, https://orcid.org/0000-0001-8766-3816).

Recibido: 23/03/2020 Aceptado: 27/05/2020

Método para diseñar la estructura lógica de un entorno de telecomunicaciones distribuida

RESUMEN

El artículo presenta una metodología para el desarrollo de herramientas de intercambio de información basada en el aparato de redes jerárquicas de Petri, que se caracteriza por la posibilidad de eventos simultáneos y permite resolver el problema del análisis de corrección y verificación de protocolos. Los objetos de protocolo y los puntos de acceso se representan sobre la base del aparato de redes de Petri: el objeto más simple del protocolo de recepción / transmisión, el objeto de cola, un único objeto y un objeto trivial. Describe la construcción de estructuras complejas de objetos y operaciones en ellos, la gestión de tiempos de espera basados en el objeto temporizador, la especificación de objetos de protocolo, la especificación de protocolos y niveles de estructura lógica. El método propuesto es relevante para el análisis de sistemas de comunicación complejos de gran tamaño, que son redes móviles ad-hoc con una gran cantidad de nodos, caracterizados por la inestabilidad de la topología debido a las características inestables del canal de comunicación y la alta movilidad de los nodos, y especialmente a Redes hoc de aeronaves (FANET) y redes de sensores de vuelo (FSN).

PALABRAS CLAVE: sistema distribuido; protocolo de transporte; red Petri; red ad-hoc voladora; FANET; red de sensores voladores; FSN.

Introduction

The most important characteristics of the telecommunication environment are largely determined by the properties of the protocols used – the rules of interaction between various objects, including remote ones. Here a special role is played by the properties of logical correctness, i.e. the absence of logical errors such as deadlocks, unproductive cycles, overflows.

This problem was dealt with by a number of scientists who made a significant contribution to the development of general theory and specific applications to the specific field of communication protocols. Works of R. Milner, K. Petri, C. Hoare are known (Hoare, 1989). A number of significant achievements in solving fundamental and applied problems in this area are associated with the names of Russian researchers: N. A. Anisimov (1989, 1990); O. L. Bandman, V. I. Varshavsky, Yu. G. Karpov (Karpov,

1987); V. E. Kotov (1984); V. G. Lazarev (Eremenko, 2018); E. I. Pilya, S. A. Yuditsky (Eremenko, 2019). The results obtained by them in the description and analysis of protocols of distributed environments are applicable to systems of small size. In a more complex environment based on the construction and analysis of a set of achievable states, the practical need involves further development of protocols. Examples of such a complex environment are mobile ad-hoc networks with a large number of nodes, characterized by topology instability due to unstable communication channel characteristics and high node mobility (Konstantinov et al., 2018, 2019). First of all, these are flying ad-hoc networks of aircraft (FANET) (Bekmezci et al., 2013; Karan, 2015) and flying sensor networks (FSN) (Purohit and Zhang, 2009; Ahmed and Kanhere, 2011).

The main purpose of developing the logical structure of a distributed telecommunications environment at system level is the design of an open system, i.e. the logical structure of the communication module. In this case, network objects are used as elementary units – logical modules that have certain external characteristics (protocol object, auxiliary objects of system management, timer objects). The process of designing at the system level is based on the composition of objects to obtain the necessary configuration. Objects can interact with each other in various ways.

The element that characterizes the direction of interaction of the object is called an access point through which you can access the medium of information transmission. There are two types of access points – *P*-points (positions) and *t*-points (transitions) (Anisimov, 1990; Kotov, 1984; Eremenko, 2018).

For a formal description of the protocol level, it is necessary to build an N-level protocol model consisting of two copies of the *n*-object and (*N* – 1)-service, i.e. to build a logical implementation of the *N*-service. Next, it is necessary to compare it with the reference *N*-service and, in the case of equivalence of these objects, we can talk about the correctness of the *N*-level protocol. The problem statement requires the introduction of the concept of equivalence of objects and methods of its verification.

To implement the task requires the creation of a formal apparatus and requirements for the design methodology of the transport level.

1. Representation of protocol objects and access points

An object is a logical module that performs certain functions. The execution of functions is accompanied by interaction with other objects, for which there are access points in the object. An interaction of two objects is realized by connecting their respective access points. The interaction itself is an exchange of primitive commands, in which parameters and data can be transferred between objects.

This characteristic of the object allows us to formally define it as a Petri net with many *t*-access points, each of which formalizes the concept of an access point to the object.

A Petri net object is a set

$$E = \langle \sum, \Gamma \rangle,$$

where $\sum = (P, T, F, M_0)$ is a labeled network called *an object structure*; $\Gamma = \{\alpha_1,...,\alpha_n\}$ – a set of *t*-access points to the network $\sum$, where each point $\alpha_i = \langle tid_i, Alph_i, \sigma_i \rangle$ specifies an access point to the object. Each point α_i has its own name tid_i, by which one point will differ from another (Anisimov, 1989).

The Petri net tid_i defines the internal structure of an object and therefore its behavior. You can interact with an object and observe its behavior only through access points. Obviously, the triggering of some transition *t* can be "seen" from all access points, but under different names. If at some point the transition is marked with a τ-symbol, then at this point it is "not visible", it is impossible to interact through it at this point. If the transition is "not visible" from all access points, it is considered to be completely internal. The entire set of access point names of the object $E = \langle \sum, \Gamma \rangle$ is denoted as $Id(E) = \{tid_a \mid \alpha \in \Gamma\}$ (Kotov, 1984).

Network objects are depicted as a Petri net, where transitions can be labeled with multiple names. For each name, a colon-delimited access point is specified. These names are separated from each other by semicolons. For example, if the transition *t* at the point α in the object $E[\alpha, \beta]$ marked with name *a*, and at the point β with name *b*, then this corresponds to the expression "α: *a*; β: *b*". If at some access point the transition is "not visible", i.e. marked with a τ-symbol, then this entry is omitted.

The simplest object of the protocol of reception/transfer is shown on Fig. 1. An object that implements the simplest acknowledgement receive/transmit protocol has two access points, *U* and *L*, to connect to the user and the transmission line, respectively.

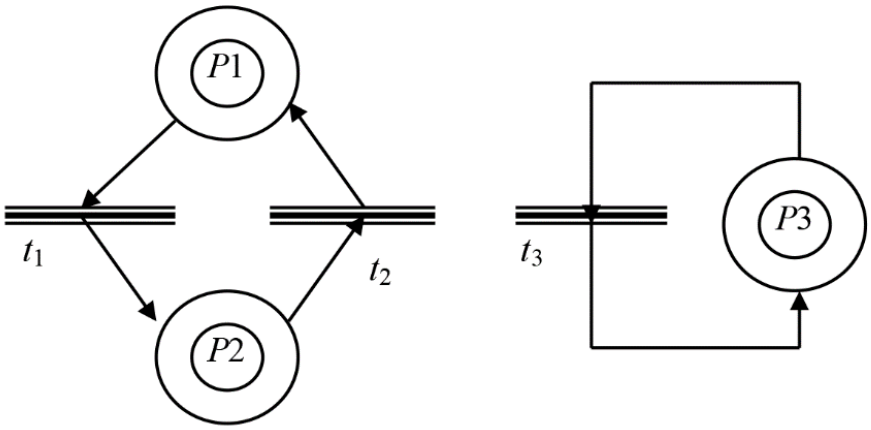

Fig. 1. Example of an object of the simplest protocol of reception / transfer $E_1[U, L]$

Transitions *t* have the following trigger conditions:

t_1 at point *U* – receiving a data request from the user (*DatReq*);

t_1 at point *U* – sending data block (*sDT*);

t_2 at point *L* – receiving acknowledgement on data block (*rACK*);

t_3 at point *L* – receiving a data block and sending confirmation to it (*rDT* + *sACK*);

t_3 at point *L* – indication of incoming data (*DatInd*).

Thus, triggering the t_1 transition means receiving a data request from the user and simultaneously sending it to the line. Receiving confirmation of this data corresponds to the operation of the transition t_2. Note that in this case the user is not informed in any way, because at point *U* this transition is "not visible". Data reception consists in triggering the transition t_3, which corresponds to receiving data from the line (*rDT*) with immediate return of acknowledgement (*sACK*) and transfer to the user of the received block (*DatInd*). Marking the transition t_3 is a good illustration of the specification of simultaneous logical events. Simultaneity can be implemented both

within the same access point (*rDT* and *sACK*) and at different points (*rDT*, *sACK* and *DatInd*).

The queue object. The E_2 object shown as an example in Fig. 2 describes a queue of capacity three. The object has two access points *in* and *out*, through which you can respectively put and retrieve items from the queue. Transition t_1 (*in*: x) is "visible" only from point *in* and corresponds to the statement operation, and transition t_4 (*out*: x) is "visible" from point *out* and corresponds to the extraction operation. The transitions t_2 and t_3 corresponding to the internal promotion of messages are completely internal and invisible from the outside.

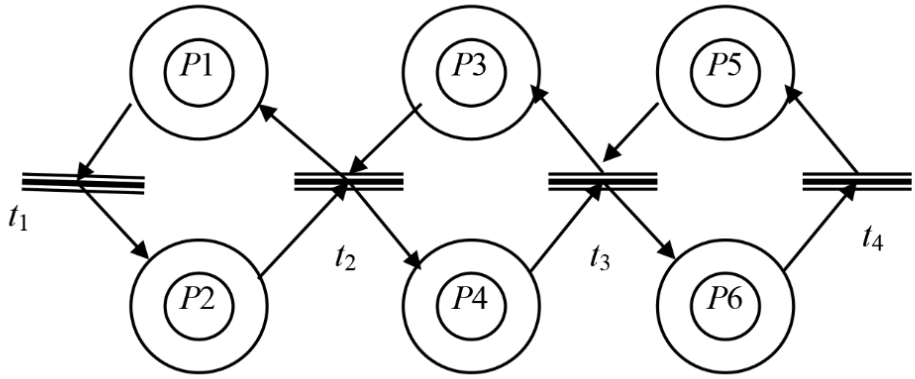

Fig. 2. An example of the queue object E_2[*in*, *out*]

A single object. Let $A = \{\alpha_1,...,\alpha_n\}$ be some set of names. If we construct an object with two points α and β so that its network is equal to

$(\varnothing, A, F, 0)$, where $F = \{\langle 0, a, 0\rangle \mid a \in A\}$, and the access points are equal to $\langle \alpha, A, \sigma_\alpha\rangle$ and $\langle \alpha, A, \sigma_\alpha\rangle$ with $\sigma_\beta(a) = \sigma_\beta(a) = a$ for all $a \in A$, then this object will only consist of transitions, and for each name $a \in A$ one transition starts, visible from α and β under the name *a*. Since transitions do not have preconditions, then at any time any shift (a multiset of transitions) can work, which is visible equally from the points α and β. Such an object is denoted as $1_A[\alpha, \beta]$ and is called a *repeater object*.

A trivial object. An object may have no access points at all, i.e. $\Gamma = \varnothing$. In this case, the object does not manifest itself in the external world. From the point of view of an external observer, i.e. from the point of view of compositionality, it does not seem to

exist, because it is impossible to interact with it. It is intuitively clear that such an object $E_1 = \langle \sum_1, \varnothing \rangle$ is equivalent to any other object $E_2 = \langle \sum_2, \varnothing \rangle$. Such an object is denoted as 0.

If some object $E = \langle \sum, \Gamma \rangle$ has two different access points $\alpha, \beta \in \Gamma$ and $\alpha \neq \beta$ have the same identifiers $tid_\alpha = tid_\beta$, then such an object does not allow to uniquely identify access points by identifiers. Therefore, a special normalization procedure called α-normalization is necessary (Karpov, 1987).

2. Construction of complex structures of objects and operations on them

For manipulation with objects, building more complex structures from them, appropriate operations are necessary.

Separate unification of objects. We will call an object

$$E = E_1 \oplus E_2 = \langle \Sigma_1 \oplus \Sigma_2, \ \widetilde{\Gamma}_1 \cup \widetilde{\Gamma}_2 \rangle$$

as a separate union of the two objects $E_1 = \langle \sum_1, \Gamma_1 \rangle$ and $E_2 = \langle \sum_2, \Gamma_2 \rangle$.

Separate unification simply allows two objects to be represented as a single object. In this case, the result object may not be α-normalized.

The operation of abstraction of the object. Let $E = \langle \sum, \Gamma \rangle$ be some object and $H \subseteq \Gamma$ be some subset of access points. Then the abstraction of the object E with respect to H is called the new object $E' = \tau_H(E) = \langle \sum, \Gamma \setminus H \rangle$.

The abstraction operation removes a subset of access points from the object definition without changing its structure (network $\sum$) and therefore without changing the behavior of the object in the remaining access points.

The operation of composition of objects. Let $E_1 = \langle \sum_1, \Gamma_1 \rangle$ and $E_2 = E = \langle \sum_2, \Gamma_2 \rangle$ be two objects in normal form, $\alpha \in \Gamma_1$ and $\beta \in \Gamma_2$ are their access points. Then, composition E_1 and E_2 in relation to α and β is called a new object $E' = (E_1 {}_\alpha||_\beta E_2) = norm(E)$, where an object $E = \langle \sum, \Gamma \rangle$ is constructed as follows:

1. $\sum = \langle (N_1 {}_\alpha|_\beta N_2), M_{01} \cup M_{02} \rangle$;
2. $\Gamma = \{ \tilde{\gamma} \mid \gamma \in \Gamma_1 \cup \Gamma_2 \setminus \{\alpha, \beta\}\}$;

The structure (network) of the resulting object is formed by a parallel composition of the source networks through the access points α and β. In this case, the original transitions intended for synchronization (T_{out}) are converted into synchronization transitions (T_{sync}) by a linear combination. The set of access points of this object is formed as a union of transformed access points of the original objects without points α and β as having played a role. Obviously, the transformation is necessary, because the structure of the network changes with the composition. After that, the resulting object is normalized, i.e. consistently be subject to the procedures α– и λ-normalization. This is really necessary because, at first, E_1 и E_2 can have access points with the same identifiers. Secondly, as a result of the operation, transitions that are λ-redundant can be formed.

Renaming of access points. Let $E = \langle \Sigma, \Gamma \rangle$ be an object and $\alpha = \langle tid, Alph, \sigma \rangle \in \Gamma$ its access point. Then renaming the access point α from the name *tid* to the name *tid'* gives a new object

$E[tid'/tid] \stackrel{def}{=} norm(E')$, where $E' = \langle \Sigma, \Gamma \setminus \{\alpha\} \cup \{\alpha'\} \rangle$, $\alpha' = \langle tid', Alph, \alpha \rangle$. Renaming multiple points at once will be denoted as $E[tid'_1 / tid_1, tid'_2 / tid_2, ..., tid'_k / tid_k]$.

To solve a number of problems related to the correctness of protocols, it is necessary to have an equivalence relation between objects, which allows comparing them from the external, behavioral side. The isomorphism relation is not suitable for this, because it is too discriminatory. The most convenient for these purposes is to use bisimulation equivalence of Petri nets.

Equivalence of objects in the access points. Let the objects $E_1 = \langle \Sigma_1, \Gamma_1 \rangle$ and $E_2 = \langle \Sigma_2, \Gamma_2 \rangle$ with access points $\alpha \in \Gamma_1$ and $\beta \in \Gamma_2$ be given in normal form. These objects will be called interleavingly equivalent at points α and β, written as $E_1 \ {}_{\alpha}\approx^{i}_{\beta} \ E_2$ if

1. $tid_\alpha = tid_\beta$;

2. $\Sigma_1 \ {}_{\alpha}\approx^{i}_{\beta} \ \Sigma_2$

These objects will be called stepwise equivalent, written as $E_1 \ {}_{\alpha}\approx^{s}_{\beta} \ E_2$, if in condition 2 the superscript *i* is replaced by *s*.

In other words, objects are equivalent at access points if, firstly, these points have the same identifier, and, secondly, labeled networks of objects are bisimulatively equivalent at these points. Like bisimulation, the object equivalence relation has two variants – interleaving and stepwise.

In addition to the equivalence at certain points, we will be interested in the equivalence of objects as a whole – simultaneously at all access points.

Equivalence of objects. The objects of $E_1 = \langle \Sigma_1, \Gamma_1 \rangle$ and $E_2 = \langle \Sigma_2, \Gamma_2 \rangle$ in normal form are called interleavingly equivalent, denoted as $E_1 \approx^i E_2$, if there exists a relation of bisimulation *R* such that $\forall \alpha \in \Gamma_1 \exists \beta \in \Gamma_2 : E_{1\,\alpha} \approx^i_\beta E_2$ with respect to bisimulation *R* and vice versa.

The step equivalence of objects denoted as $E_1 \approx^i E_2$ is defined similarly, only the superscript *i* in the condition is replaced by *s*.

Thus, full object equivalence is defined as equivalence at all access points *at the same time*. This simultaneity is provided by the uniform bisimulation relation *R* for all equivalences. Obviously, equivalent objects have the same number of access points, and for each point of one object there is a point of another object with the same identifier.

Objects equivalence, based on bisimulation, allows identification of objects that have different internal structure, but the same external behavior. However, in order to make active use of this equivalence, such as replacing one object in a model with an equivalent one without disrupting the behavior of the whole system, it is necessary that this equivalence relation be *congruent* with respect to operations on objects. If you take the operation of objects composition, the congruence property says that if $E_1 \approx E_2$, then $E_\alpha||_\beta E_1 \approx E_\alpha||_\beta E_2$.

3. Specification of enterprise telecommunication environment

With the help of the Petri net object concept, the concept of service and its architectural components is formalized. A service specification can be defined using a network object whose access points are interpreted as service access points. The alphabet of each access point is a set of service primitives (PS), where the names of the

transitions correspond to the names of the service primitives, and the set of terms – the parameters of the primitive. In particular, the connection endpoints specification is implemented using a special parameter, the connection endpoint identifier, assigned to each service primitive.

Thus, the specification of the service in this case is nothing but a Petri net, in which the transitions are mapped to the PS, and their triggering is treated as the execution of these primitives. It is easy to see that the rules of functioning of Petri nets completely satisfy the rules of execution of PS – their operation is an indivisible instantaneous operation. In general, the behavior of such a system specifies the behavioral part of the service, from which, in particular, you can derive information about possible valid sequences of execution of service primitives, including the values of their parameters. An object in this interpretation is called *a service object*.

Data transfer in violation of the order. One of the simplest services is the transfer of data block from one point to another, in which the order of messages is not preserved. In this case, two service access points are defined to send (*s*) and receive (*r*) blocks of data. Sending is done by executing *DatReq*(*x*) at point *s*, and receiving is done by executing *DatInd*(*x*) at point *r*. Parameter x contains the transmitted data. If there are labels in position P_1, the transition t_2 can work by deriving one of the arbitrary labels from position P_1, whose parameter *x* is "visible" from point *r*. It is obvious that the order of the data blocks is not respected here. In addition, this service does not limit the number of data blocks in it.

Data block transfer service. This service is provided by the transport protocol (parametric version). It is initially point-to-point, i.e. assumes only two users sending and receiving users, denoted by access points *s* and *r*.

Fig. 1 shows a service object that specifies a data block transfer service. The service object has two access points *s* and *r*. At point *s* the primitive *DatReq*(*x*) may be executed corresponding to the transmission request data block *x*. At the point *r* the primitive *DatInd*(*x*) is executed, the corresponding to indication for the user about the receipt of data block *x*. From the network view, we can draw the following conclusions concerning this service. First, the execution of query primitives and data indication at

points *s* and *r* alternate. Secondly, the block of data transmitted from point *s* is delivered to point *r* without changes, which is provided by the parameter *x*. Indeed, after the transition t_1 is triggered, the variable *x* takes one of the valid values, which is assigned to the label at position P_2. Next, the same value appears when the transition t_2 is triggered, which corresponds to the execution of the *DatInd*(*x*) primitive.

The final capacity queue. The data block transfer service and the T-service specify a buffer of the capacity queue type equal to one. It is not difficult to specify a transmission service with an arbitrary finite capacity. Fig. 2 shows an object describing a service that transfers three blocks of data at the same time (queue capacity three).

Here the transitions t_2 and t_3 are completely internal and correspond to the "advance" of the data block *x* inside the queue.

The queue with loss and duplication (Fig. 3). The previous examples described a fairly high quality service, i.e. a service that provides error-free data transmission. However, in practice, for a number of reasons, a lower quality service with errors is often used. One such type of error, associated with a violation of the order of transmission of blocks, was considered earlier. Other typical errors of such services are loss and duplication of data blocks. All such properties can be expressed in terms of the Petri net apparatus.

Triggering of the internal transition t_4 corresponds to loss of the data unit in the queue, and triggering of the internal transitions t_2 and t_5 corresponds to duplication of block *x*, number *z*, resulting in an additional unit *x* with the number one more than the [$y = z + 1$]. A service with only one kind of error is easily obtained by removing the corresponding transition along with the surrounding arcs (t_4 or t_5).

4. The timer object

Most protocols use a time service called a *timer* in their operation. Different protocols impose approximately the same requirements on the timer. Therefore, it is advisable to allocate the timer in an independent object and use it in composition with protocol objects both individually and in the separation mode (Fig. 4).

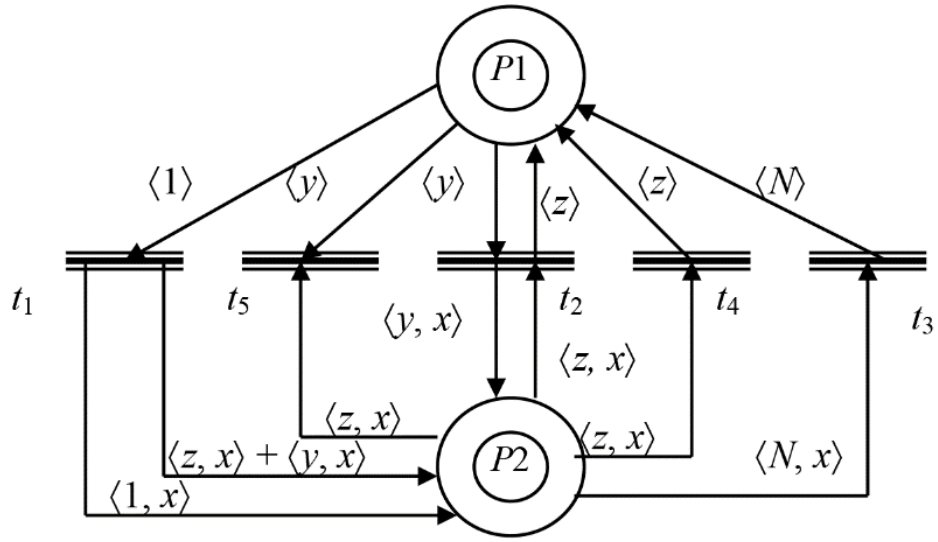

Fig. 3. Service object with loss and duplication *FIFO N-LD* [*s*, *r*]

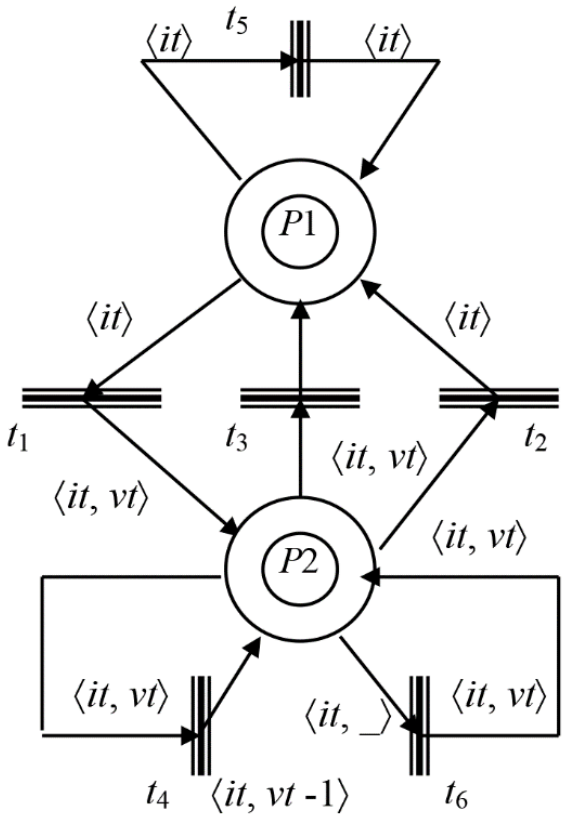

Fig. 4. The timer object

Based on the analysis of the existing ways of using the timer protocols, we can make the following conclusions about its functions. The timer should provide management of timeouts: start of a timeout with a certain value of an interval; its

shutdown before the moment of the expiration and alarm to the user about its expiration. Moreover, it is often necessary to restart the timeout, i.e. to set the enabled timeout to a new value of the time interval (Eremenko, 2015).

The timer object has one access point *u* for communication with users. A timer can manage multiple timeouts at the same time, in this case *n* instances. For each timeout, a label with a unique identifier it_i is allocated. Initially, all timeouts are disabled (all labels (it_i) are in position P_1). The timeouts are controlled by four commands that define the conditions for triggering transitions:

t_1 – start of timeout *ON* (*it*, *vt*);

t_2, t_5 – timeout *OFF* (*it*);

t_3 – end of time (expiration) timeout *EXP*(*it*), [*vt* = 0];

t_4 – timeout execution [*vt* › 0];

t_6 – restart timeout *RS*(*it*, *vt*)

The *vt* parameter in *ON* and *RS* commands specifies the timeout interval.

If the user needs to run a timeout of the value *vt*, he issues the command *ON* (*it*, *vt*), where the parameter *it* can be free (i.e. not defined in advance). After executing this command (t_1 transition triggered), one of the free it_i identifiers is moved from position P_1 to position P_2, and its value is returned to the user for further work with this timeout. Further, the triggering of the internal transition t_4, gradually decreases the value of *vt* and when it becomes zero, the command *EXP*(*it*) is executed that indicates expiration of the timeout with identifier it_i. The user may, before the expiration of the specified interval, to turn off the timeout command *OFF*(*it*) that translates the tag it_i from position P_2 to P_1. It may happen that this command will be issued by the user after the *EXP* command. In this case, the t_5 transition will be triggered which does not change the state of the timer objects. The user may need to restart the pending timeout by setting its interval to a new value. To do this, it can use the *RS*(*it*, *vt*) command, which does not change the timeout state, but only changes its value.

5. Specification of protocol objects

A protocol object is a logical module that interacts with various objects, among which there are necessarily protocol objects of the upper and lower neighboring levels and, possibly, some auxiliary objects. A protocol object implements a protocol of corresponding level.

Using the apparatus of objects of Petri nets, it is possible to represent schematically the protocol object in the most general form. The protocol object of *N*-level is represented as a network object $PE_N[S_N, S_{N-1}, P_N, T_m, C]$ having five access points. Here S_N and S_{N-1} are access points to protocol objects of the upper and lower neighboring levels, respectively. The alphabet of these access points are the service primitives of the *N* - and (*N*–1)-levels, respectively. The P_N access point describes the interaction of a protocol object over a "clean" protocol with a remote protocol object of this level. The alphabet of this access point is a set of protocol commands and the behavior of the object from this access point is "seen" as the execution of protocol rules for the exchange of protocol commands. Between points P_N and S_{N-1} there is a certain communication since protocol commands from P_N are actually forwarded through an access point S_{N-1} in the form of data of service primitives. For this reason, the P_N point is redundant, because protocol interaction can be restored from the behavior of the object at the S_{N-1} point.

In addition, the protocol object includes two auxiliary access points *Tm* and *C*, designed to interact with the timer object (in cases where it is convenient to have a common time service for all objects) and the control object. It is possible that the protocol object may have other auxiliary access points.

A protocol object specification in its most general form is called *a complete protocol object specification*. To solve specific problems of protocol engineering, simplified specifications, which are called *problem-oriented* specifications, are enough. For example, for protocol implementation purposes, a complete specification is required except for the P_N point, which can be obtained by the abstraction operation: $\tau_{\{PN\}}(PE_N)$. To analyze protocols for the presence/absence of deadlocks, as will be seen later, a specification with one P_N point is sufficient: P_N: $\tau_{\{SN, SN-1, Tm, C\}}(PE_N)$.

Transport protocol (T-protocol, Fig. 1). The T-protocol is asymmetric, i.e. it has two types of protocol objects – receiving *TPEr*[*r*, *b*] and transmitting *TPEs* [*s*, *a*] objects.

The transmitting TPES object[*s*, *a*] has two access points: point *s* for the user and point *a* for the service being used. At point *s*, the object receives a block of data *x* from the user (PS *UDatReq*(*x*)) and immediately passes it to the protocol command *DT* (*x*). This protocol command is actually transmitted through the service used by the *LDatReq*(*DT*(*x*)) primitive, after which the object goes into the state of waiting for the protocol command acknowledgement ACK, which comes in the *LDatInd*(ACK) primitive and then the object goes to the initial state. In this version of the T-protocol, two logical actions of receiving a block of data from the user and its transmission in the service primitive are performed simultaneously as a result of performing a single transition. Indeed, as common sense dictates, there is no need to perform these operations separately. This possibility of specifying several logical actions simultaneously in one physical action is one of the most important advantages of the introduced apparatus, which increases its expressive capabilities.

The receiving object *TPEr*[*r*, *b*] has two access points *r* and *b*, designed respectively for the user and the service used. Its behavior consists in repeated triggering of one transition that, however, includes performance of several logical actions – reception of the protocol command *DT*(*x*) with data *x* in PS *LDatInd* from the lower level; transfer to the user of the received block *x* in PS *UDatInd*(*x*); return of the protocol command of confirmation ACK in PS *LDatReq*. This object also has significant differences from the original version.

In addition to simultaneous interaction at different access points, there is simultaneous execution of several primitives at one point. Namely, at point *b*, the transition is labeled with the *MultiSet LDatInd*(*DT* (*x*)) + *LDatReq* (ACK), which specifies the simultaneous execution of these primitives. This mechanism is also very useful and powerful, because it avoids unnecessary intermediate states and, in the end, leads to a greater degree of compactness.

From this description it is possible to allocate "pure" protocol, having projected behavior of object on a set of protocol commands *DT* and ACK. The sending *TPEs* object

sends PS *DT*(*x*) (data) and waits for confirmation. The receiving *TPEr* object, having received *DT*(*x*), returns a confirmation ACK. After that, *TPEs*, having received ACK, goes to the initial state.

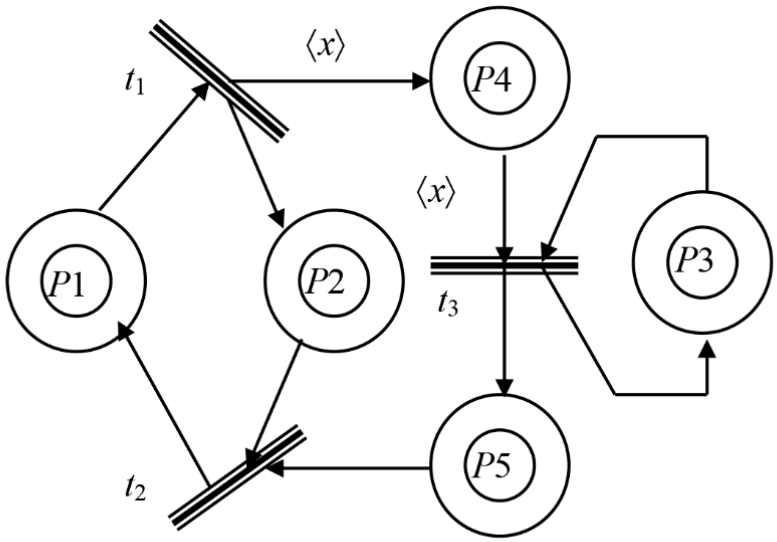

Fig. 5. Specification of means of information exchange
(three-level T-protocol)

6. Specification of protocols and logical structure levels

As a rule, work on the formal specification of protocols and levels of the logical structure of the telecommunication environment is performed at different levels of compositionality. At the system level, the final architectural actions are performed when objects specified at other levels of compositionality are combined into the final configuration.

Thus, depending on the protocol model, the protocol specification is completed by the composition of protocol objects and the object of the underlying service or data medium (Fig. 5). The last three-level model of the protocol is a specification of the entire level of the logical structure, which includes the specification of protocol objects, objects of the underlying and provided service.

As an example, here is the specification of the T-protocol. To do this, we take two protocol objects *TPEs* [*s*, *a*] and *TPEr*[*r*, *b*] (Fig. 1), the simplest duplex service *SE*1 and combine them as follows: *TL*[*s*, *r*] = (*TPEs* || *SE*1 || *TPEr*).

Conclusions

The presented technique allows to create a formal basis for the development of the logical structure of the distributed telecommunication environment at the system level. Its implementation allows:

- formally define a concept of object as a certain logical module with access points specifying directions of interaction with object;

- define rules for composing objects through access points, allowing to build complex configurations for determining the final means of information exchange, and for subsequent analysis of protocol implementations;

- provide the ability to determine the equivalence relationship between objects and the availability of key means of its verification;

- introduce a new concept of a network object, defined as a Petri net with several marks – access points to the object. The most important advantage of the objects is the possibility of specification of simultaneous events, which significantly increases the expressive and analytical capabilities of the device and compactness of the resulting descriptions. Graphical representation of network objects in architectural and network forms is introduced;

- introduce operations on objects that allow to design complex configurations of objects, the central of which is the operation of the composition of objects. The standard properties of operations that allow efficient manipulation of objects are shown;

- introduce the concept of objects equivalence based on the bisimulation equivalence of Petri nets. It is shown that this equivalence is a congruence with respect to all operations on objects.

- to interpret key concepts of the logical structure of the telecommunication environment, such as protocol, service, data transmission medium in terms of network

objects. In terms of the apparatus, the main tasks of development of the system level of logical structure, including the task of specification, are strictly formulated and solved.

Acknowledgments

The work was supported by RFBR grant 19-29-06030-MK "Research and development of wireless ad-hoc network technology between UAVs and control centers of "smart city" on the basis of adaptation of transmission mode parameters at different levels of network interaction".The theory was prepared in the framework of the state task FZWG -2020-0017 "Development of theoretical foundations for building information and analytical support for telecommunication systems for geo-ecological monitoring of natural resources in agriculture".

References

Ahmed, S., Kanhere, S. (2011). Jha, Link characterization for aerial wireless sensor networks, in: GLOBECOM Wi-UAV Workshop, 2011, pp. 1274–1279.

Anisimov, N. A. (1989). Recursive definition of the hierarchy of protocols based on Petri nets. Fourteenth all-Union school-seminar on computer networks. (Minsk, 1989). – Moscow; Minsk: VINITI, 1989, CH. 2, pp. 101-106.

Anisimov, N. A. (1990). Hierarchical composition of protocols. Automation and computer engineering, 1990, No. 1, pp. 3-10.

Bekmezci, L., Sahingoz, O.K., Temel, S. (2013). Flying Ad-Hoc Networks (FANETs): A Survey, Ad Hoc Networks, 2013, Vol. 11, № 3, pp. 1254–1270;

Eremenko, V. T. (2018). Mathematical models and methods for solving problems of optimization of reliability of systems with complex structure / D. S. Mishin, V. T. Eremenko, M. Yu. Rytov / / Bulletin of the Bryansk state technical University. 2018-No. 4 (65). - Pp. 88-95

Eremenko, V. T. (2019). Mathematical models of information exchange algorithms of ad-hoc networks of geographically distributed construction objects. / V. A. Eremenko, V. T. Eremenko, V. M. Haramohan. Information systems and technology. 2019. - No. 2 (112). - Pp. 41-47

Eremenko, V.T. (2015). Method of forming test kits for security protocols in data processing systems. V. T. Eremenko, V. M. Haramohan. Information systems and technology. - 2015. - No. 2 (88). Pp. 131-137.

Hoare, H. (1989). Communicating sequential processes. – Moscow: Mir, 1989. - 264 pp.

Karan Palan, Priyanka Sharma. (2015). FANET Communication protocols: A Survey, Volume 7, Number 1, Sept 2015 - March 2016, pp. 219-223. DOI: 10.090592/IJCSC.2016.034.

Karpov, Yu. G. (1987). On the property of protocol coherence. Automatics and computer engineering, 1987, No. 4, pp. 38-40.

Konstantinov, I.S., Vasilyev, G.S., Kuzichkin, O.R., Surzhik, D.I., Lazarev, S.A. (2018). Numerical and Analytical Modeling of Wireless UV Communication Channels for the Organization of Wireless Ad-Hoc Network. IJCSNS - International Journal of Computer Science and Network Security - 2018. - Vol. 18, No. 8, pp. 98-104. Open access: http://paper.ijcsns.org/07_book/201808/20180815.pdf.

Konstantinov, I.S., Vasyliev, G.S., Kuzichkin, O.R., Surzhik, D.I., Kurilov, I.A., Lazarev, S.A. (2019). Development Of UV Communication Channels Characteristics Modeling Algorithm In A Mobile Ad-Hoc Network / Journal of Advanced Research in Dynamical and Control Systems (JARDCS) / ISSN: 1943-023X / Volume 11 | 08-Special Issue, 2019. Pages: 1920-1928. http://www.jardcs.org/abstract.php?id=2543

Kotov, V. E. (1984). Petri Nets. – Moscow: Nauka, 1984. - 160 pp.

Polshchykov, K.; Lazarev, S.; Kiseleva, E. (2019). Decision-making supporting algorithm for choosing the duration of the audio communication session in a mobile ad-hoc network, *Revista de la Universidad del Zulia*, 10 (27), 101-107.

Purohit, A., Zhang, P. (2009). Sensor Fly: a controlled-mobile aerial sensor network, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys '09, ACM, New York, NY, USA, 2009, pp. 327–328.