Macroplaces in High Level Petri Nets: Application for Design Inbound Call Center*

Nikolay Anisimov**, Konstantin Kishinski, Alec Miloslavski,
Pavel Postupalski

Genesys Labs, 1100 Grundy Lane, Suite 125, San Bruno,
CA 94066, e-mail: alec@genesyslab.com
http://www.genesyslab.com

ABSTRACT

The given work is devoted to the use of the Petri net approach for the construction of formal models, intended as the basis for development of logical structure of inbound call centers. As a formal model for a specification of agents' scenarios we take a formalism called script-net that belongs to a class of high level Petri nets. It is emphasized that scripts describing real-world scenarios are usually extremely complicated and require some means of modularization. In this paper we suggest the extension of high level nets called high level macronets, intended for specification situations which are asynchronous to normal processing of scripts. The model is shown to be a compact notation of high level nets without macroplaces and a corresponding transformation procedure is presented. Some examples are used to illustrate the power of the formalism.

1. Introduction

This paper addresses the application of high-level Petri nets to design an architecture and software package for inbound call centers (ICC) that is a typical and sophisticated CTI-application [10].

Inbound call centers consist of a set of operators, called agents, who receive and process inbound calls from remote clients. Apart from using a telephone, call processing may involve the employment of computer systems and other devises, like faxes, as well as communication with other agents (e.g. forwarding of the call to a more qualified agent). Typically, an ICC operates with calls of different natures and an agent can be switched from processing calls of one type to processing calls of another type.

The behaviour of an agent processing the call is heavily regulated by a scenario referred to as "script", each such specification being specially designed for specific types of calls by the experts in telemarketing. It should be noted that an agent must not be a specialist in the matter of the call being processed, he or she simply follows the script with the aid of the computer display.

One of the major problems here is the design and implementation of such scripts. It is clear that to cope with this problem one needs special tools supporting this process: language and underlying formal model capturing all features of scripts and especially the aspect of communication.

In [2] we suggested a formal model called "Script System" intended for the design of the model of ICC as a set of disjointed scrip-nets, their tokens represent agents who can communicate with each other by message sending. Here, a script net is a formal representation of a script. The model is based on cooperative nets reported in [9] that have been slightly extended and modified. In

* Published in: Proceedings of the Int. Conference on Information Systems Analysis and Synthesis (ISAS'96), pp.153-160 (Orlando, Florida, USA, July 22-26, 1996).

** Also with the Russian Academy of Sciences

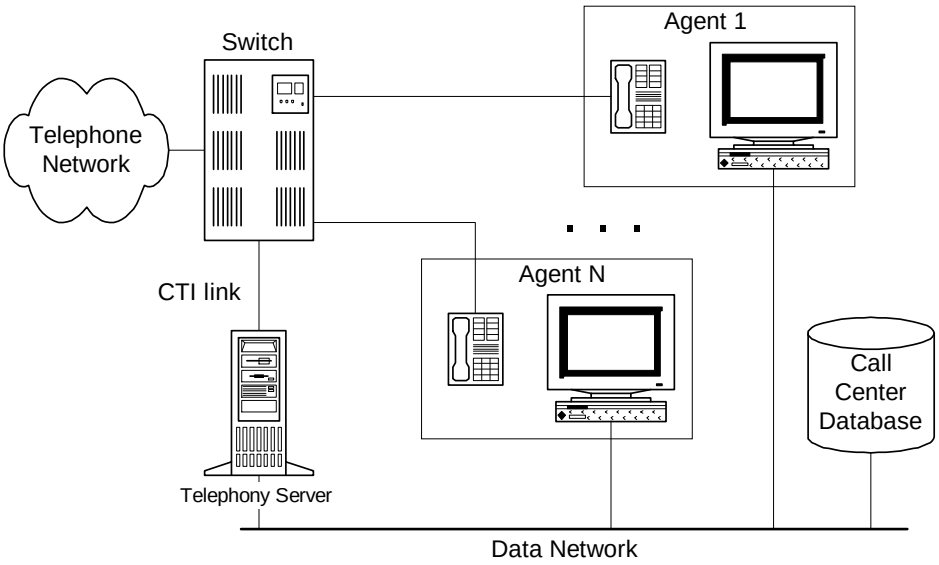

Figure 1. Call-Center structure

particular, we have added a transition merging scheme of communication between objects, allowed one to label transitions by a set of action expressions, and equipped nets with Merlin's time constraints.

At this point, we should note that scripts describing real-world scenarios are usually extremely complicated to work with and therefore require some means of modularization. In this paper we consider the problem of structural representation of script nets. In this respect, we can point out two techniques for modularization in Petri net based models we would like to employ – hierarchical transitions and macroplaces. The first technique is well elaborated within the framework of high-level Petri nets, e.g. see [3]. Generally, it consists of representing a hierarchical net as a set of disjoint subnets with links between transitions and subnets forming a hierarchical structure. Firing of such a hierarchical transition causes an execution of its internal net that consists of the firing of a transition (or step) sequence from initial marking to terminal one. So using this technique we can represent script net as a set of hierarchical organized script subnets.

At the same time in call processing we can face situations which are asynchronous to normal processing, and a reaction to such events should also be specified. For example, there may be situations when, during the dialogue between agent and client, the telephone line is disconnected (e.g. suddenly client puts down a receiver); as well as more sophisticated situations when the processing of current calls is interrupted and the agent is forwarded to process new calls with higher priority. Moreover, processing of such broken calls could be recommenced upon available agents. To specify such situations in script nets, special constructs are needed. To accomplish this, we suggest using the concept of macronets reported in [1] and generalized on high level Petri nets.

2. Structure of inbound call-center

In this section we present an abstract model of a typical ICC that will serve as a subject for the formalization process. From now on, a ICC will be referred to as a "System".

Typically, a system operates with a set of resources. These are: equipment (e.g. phones, fax machines, switches, a local area network, etc), software components (data base, text editor, etc) and personnel involved in system operation (agents, administration). All communications of the system is accessed through these resources. Typical Call-Center environment is shown on Figure 1.

At higher level, the system can be perceived as a collection of communicating objects, see Figure 2, which form the application level of the system.

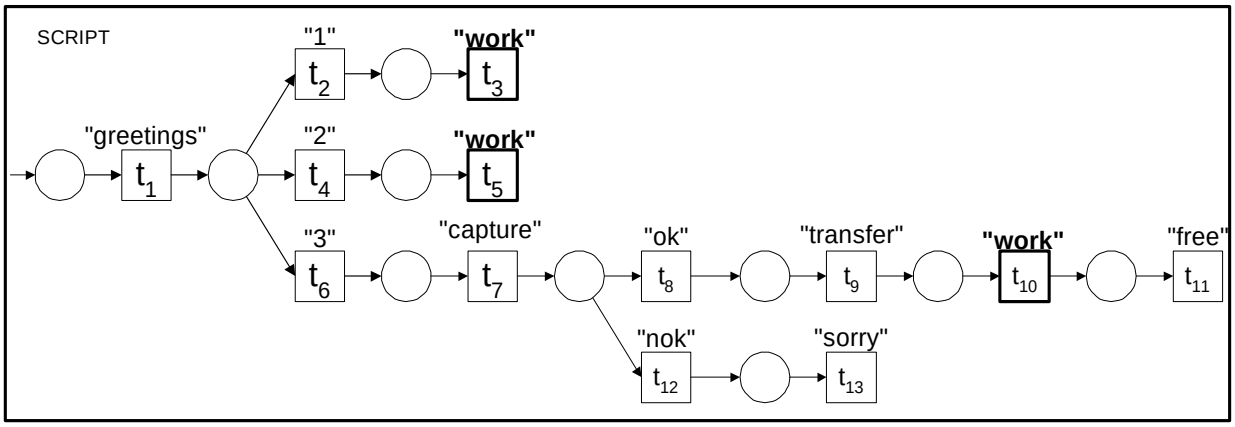

Figure 3. Example of script-net

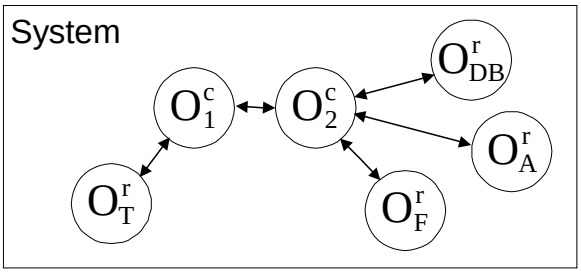

Figure 2: Structure of application level

We will divide all objects of the application level into two types: resource objects and call objects. The former represents objects corresponding physical resource of the system while the latter represents objects intended for call processing. In Figure 2, we can see two call objects O_1^c and O_2^c and four recourse objects O_T^r, O_F^r, O_A^r, O_{DB}^r, corresponding to telephone, fax machine, agent and data base, respectively. Communications between objects are depicted by two-directed arcs. We also see that the object O_2^c can use a fax machine O_F^r, data base O_{DB}^r, enlist an agent O_A^r and create a new call object O_1^c.

The behaviour of each object is regulated by a scenario specification, called 'script'. There may be several objects working in accordance with one and the same script. For example, for a script describing the behaviour of a telephone, there may be several objects corresponding to actual telephones in the system. On the other hand, for a script specifying the call processing, there may be several objects processing different calls of the same type.

We will assume that each script is identified by a unique name within the system. Moreover, we associate with each script a domain of object names, to identify each object within the script. This addressing scheme allows us to uniquely identify objects within the whole system.

3. Script-nets and script-systems

In this section we introduce a formal model comprising script-net and script-system [2] and based on object oriented model known as cooperative nets [9] equipped with additional facilities like multilabelling, macrostructure, and time constraints.

In the following we will assume the familiarity with the main concepts of Petri nets and high-level nets. The basic definitions are presented in Appendix A.

A Script system will be thought of as a construction having the following components:

1. A set of disjointed subnets called script-nets. Each net is annotated with standards for high-level inscriptions allowing manipulation of data. Each script net within the system is identified by a unique name.

2. A marking of a script-system is formed by a marking of script nets. Each marking of script net defines a set of objects. To accomplish this, each token has an object identifier.

3. Each transition in the script net can be labelled by a set of expressions of the forms:

a) sending the command: $s\left(\text{script}(v).\text{com}\left(v_1,\ldots,v_n\right)\right)$, where $\text{script}(v)$ points out a target object, $\text{com}\left(v_1,\ldots,v_n\right)$ is a command with parameters.

b) receiving the command: $r(\mathrm{com}(v_1,\ldots,v_n))$, that gives the command with parameters;

c) creation of a new object: $c(\mathrm{script}(v),v_1,\ldots,v_n)$, where the $\mathrm{script}(v)$ identifies a creating object and $v_1,\ldots,v_n$ its initial parameters.

4. Each transition is associated with a pair of real numbers $(t_{\min},t_{\max})$ giving a time interval enabling the transition, similar to Merlin's timed nets [6].

According to this definition one and the same transition can have several sending, receiving and creation expressions simultaneously.

At the application level of the system, we will use a transition fusion scheme as the semantics for sending and receiving. Note, that for more detailed levels, where the distributed nature of the system is taken into consideration we will use more sophisticated schemes of synchronization between transitions, e.g. "client-server" protocol.

Example 1: Script-net for catalogue sales.
In this section we build a script-net corresponding to a script for processing a call for catalogue sales from the area retail services, see [5, p.5-24]. A customer telephone a call center wishing to find out about availability of items in a catalogue, status of an order, delivery options, etc. All simple calls can be processed automatically by voice processing system. Other calls, especially those corresponding to dealing with new customers who need special assistance, are requested to be processed by the operator. In the Figure 3, we present a script net corresponding to processing this type of call.

In entering the call into a script (a token appears in a head place s_0), the system plays a greeting to a calling customer and gives a choice: to press "1", "2", "3" (the transition t_1) that corresponds to call concerning availability of items, status of the order or other types of calls, respectively. In the first two cases ("1" and "2") the call is processed automatically (hierarchical transitions t_2 and t_4). The third case needs the intervention of an operator, who is captured by the expression `s(agent(), capt(id,y))`. Here `agent` is the name of the script-net for resources corresponding to operators, `capt` the name of capturing command, `id` an identifier of current object, `y` is the variable for the address of the operator. If in the system there is a free operator s/he is captured t_8 and his/her address is placed in the variable `y`. After which the call is transferred and the operator works with the customer (hierarchical transition t_{10}). At the end of the conversation, the operator is released (t_{11}). If there are no free operators (t_{12}) the system then plays the file with apologizes.

4. Macronets

In this section we introduce Petri nets with macroplaces and then on its base high-level macronets. In the following we will assume the familiarity with the main concepts of Petri nets [7,8] and high-level nets [3,4,9]. The basic definitions we need in this paper are presented in Appendix A.

4.1. Petri nets with macroplaces

Notions of macronets and macroplaces have been introduced in [1] for specification of such situations where execution of one procedure may be interrupted by starting the execution of another procedure. Syntactically, a macronet is defined similar to nets with hierarchical transitions, however we use macroplaces instead of transitions. In other words, a macronet could be perceived as a set of Petri nets equipped with hierarchical links of the type "place $\mapsto$ net". A place for which the function ρ is defined will be referred to as a macroplace. A net corresponding to a macroplace will be called an internal net. For each internal net we will distinguish a place called a head place that points out the initial state of the net.

More concisely, a macro net is defined as a tuple $MN=\langle R,\rho\rangle$ where

1. $R=\{N_1,N_2,\cdots,N_n\}$ is a set of disjointed Petri nets $N_i=\langle S_i,T_i,F_i\rangle$ such that $S_i\cap S_j=T_i\cap T_j=\varnothing$ for all $i\neq j$. We denote $S=\bigcup_i S_i$.

2. $\rho: S\to R$ is a partial function that for each macroplace assigns an internal net. The function ρ induces a tree structure, i.e., first, there are no recursive links (there are no chains of the type $(s^0,N^1)(s^1,N^2)\ldots(s^{k-1},N^k)$ such that $s^i\in S^i$ and $s^0\in S^k$) and, second, each N_i is an internal net of only one macroplace.

Graphically, a macronet can be represented as a set of included nets, each internal net being drawn within a circle of corresponding macroplace. The head place of an internal net is marked by an incoming extra arc.

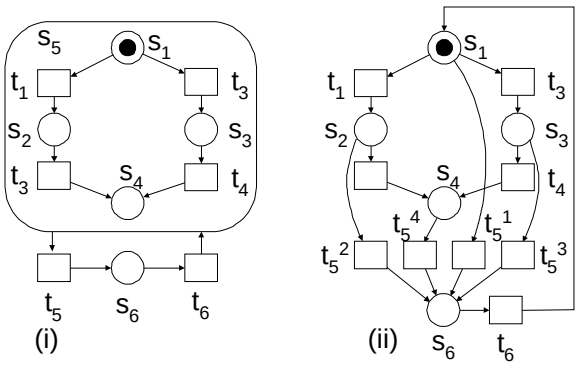

Figure 4: Example of a macronet.

On the Figure 4 (i) an example of a macronet is shown. This macronet is defined by two nets, one of them is an internal net of the macroplace s_5.

The firing rules of macronets are as follows[1]:

- A macroplace is considered to have a token if its internal net also has a token;
- adding a token to a macroplace results in adding a token to the head place of the internal net;
- removing a token from a macroplace results in removing a token from the internal net no matter what position it is in.

Returning to the example on the Figure 4, note that the transition t_5 is enabled while a token is in s_1, s_2, s_3 or s_4, in this case the macroplace s_5 is always marked. The firing of t_5 results in removing a token from an internal net, i.e. from the place s_1, s_2, s_3 or s_4. Then firing of a transition t_6 adds a token into the head place s_1.

It is clear that the concept of a macroplace is helpful for representing various situations where an interruption is involved.

It should be noted that a macronet can be transformed into an equivalent net without macroplaces, this, however, could result in an extremely bulky net. In this case, a macronet is nothing more than a compact and convenient notation of conventional Petri nets.

Let us consider the transformation procedure in more detail. From a graphical point of view, this procedure is to remove circles corresponding to macroplaces, splitting output transitions and drawing some additional arcs. Let m be a macroplace and $\rho(m) = N_m = \langle S_m, T_m, F_m \rangle$ its internal net with the head place $s_h \in S_m$. Then each outgoing transition of the macroplace $t \in m^\bullet$ is split into the set of transitions $\{t^s | s \in S_m\}$, each copy t^s corresponding to each place $s \in S_m$ of the internal net and a new arc (s, t^s) connecting a place with a corresponding copy of transition. Other arcs adjacent to the transition are inherited by its copies. Each incoming transition $t \in {}^\bullet m$ remains the same and connected with the head place s_h of the internal net, i.e. the arc (t,m) is a substitute by (t,s_h). On Figure 4(ii) an example of a transformation procedure is presented. The transition t_5 has been split into four copies t_5^1, t_5^2, t_5^3 and t_5^4 corresponding to places s_1, s_2, s_3 and s_4, respectively. The transition is connected with the head place s_1. From this example we can see that transformed nets behave absolutely like its initial macronet and can serve as semantics for the macronet. At the same time, it is clear that macronet notation is more compact and well-structured and therefore its structure and behaviour can be more easily understood.

4.2. High level macronets

Using standard possibilities of high-level nets we generalize a notion of macroplaces that will allow us to specify more general constructions. First, we will be able to build constructions where execution of internal net can be interrupted only in some specified regions. Second, we can specify a head place of internal nets dynamically that will help us to inject a token into any desired place.

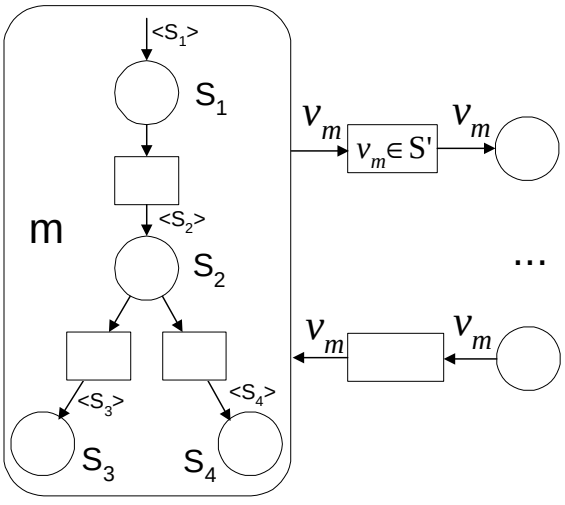

Figure 5: Macroplace and internal net

[1] In order to not overburden the paper, we consider a simplified case where internal nets belong to a class of state-machine nets

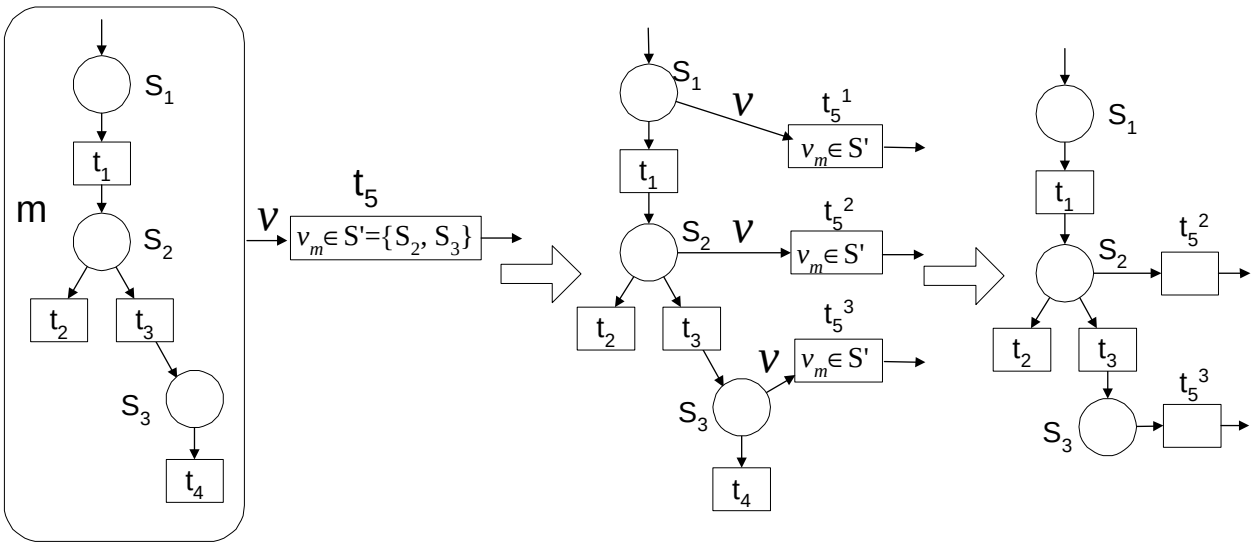

Figure 6: Splitting of an output transition

Let m be a macroplace and $N_m = \langle S_m, T_m, F_m \rangle$ be its internal subnet. For an internal net we introduce a data type $type_m$ with a domain equal to a set of internal places: $Dom(type_m) = S_m = \{s_1, \ldots, s_n\}$. Let us add to the token internal net an item of type $type_m$, the value of the item is exactly equal to the place where the token is situated. This can be easily implemented by assigning to tuples of incoming arcs with corresponding values from S_m, see Figure 5. Then we add to this a precondition of outgoing transition $t \in m^\bullet$ an expression of the type $v_m \in S'$, where $S' \subseteq S_m$. Firing of the transition t results in removing a token from a place of S'. Note that we can 'remember' the actual place where the token was before it has been removed. For an arc incoming to the macroplace m, the corresponding item of the tuple is stated in a suitable manner. For instance, if it is equal to a place $s_i \in S_m$ the adding of a token to macroplace m will result in injecting a token into s_i of the internal net. Apart from a constant, we can write a variable of the type $type_m$ that allows us to determine the incoming place dynamically. Specifically, we can place a token back to where it had been initially, see Figure 5.

Similar to conventional macronets, high level macronets can be transformed into equivalent high level nets without macroplaces. This also will lead to the splitting of transitions adjacent to macroplaces including input transitions. On Figure 6 the example of splitting output transition is depicted. Here $S' = \{s_2, s_3\}$ and the transition t_5

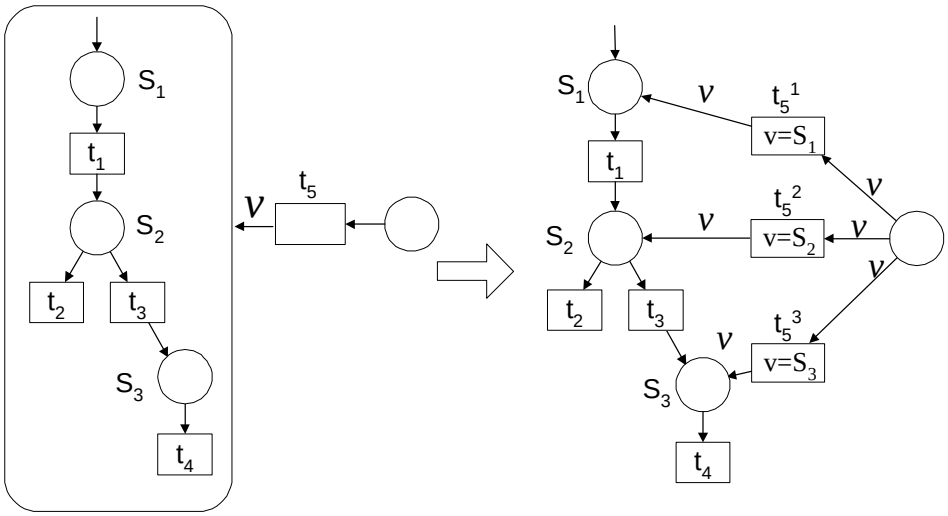

Figure 7: Splitting of an input transition

with a precondition $v \in S'$ can interrupt only execution of a script in place s_2 or s_3. It is clear that the preconditions of transitions t_5^2 and t_5^3 are always 'true' but for transition t_5^1, the precondition is always 'false' and this transition can be painlessly removed, see Figure 6.

On Figure 7 the example of transformation of high level macroplaces with input transitions is given. Similar to the first case, the transition t_5 is split into three copies t_5^1, t_5^2, and t_5^3. Here, however, for each new transition t_5^i we add new precondition $v = s_i$ that forwards the token into a desired place.

With the aid of a macroplace, we can easily specify the next situation in an agent's scenario:

- Interruption of a script execution (naturally with an apology to a client) at any stage with subsequent return to an initial state. In this case the processing of the interrupted call is cancelled ($S' = S_m$, $s_i = s_0$ where s_0 is a head place of an internal net.)

- Interruption of a script execution only if it is in special regions of the script $\left(S' \subset S_m\right)$.

- Interruption of a script execution while noting the place of interruption and possible current parameters of the call processing. This information can be used for future recommencing of the processing of the call ($S' \subseteq S_m$, in a variable v_m we document the place of interruption and a tuple of incoming arcs is equipped with the variable.)

5. Examples

In the next example we extend our script net shown on Figure 3 with macro constructions allowing us to specify two asynchronous procedure, see Figure 8.

First, imagine that for some reason an agent involved in call processing presses a button "not ready" on her/his telephone and become unavailable. This event is corresponded to firing of the transition t_{14}. After that the script tries to find another agent (t_{15}). If such available agent exists, the control of the script is returned to the same place where script was interrupted and call processing is continued.

Second, imagine that a client suddenly hang on a receiver. This event is corresponded to firing of the transition t_{19}. Firing of this transition 'disrupts' the execution of the script including the construction defined above, releases its agent (if any) and then terminates the call processing.

6. Conclusion

In this paper we generalize a notion of macroplace onto a class of high-level Petri nets. It allows designers to specify situations which are asynchronous to a normal execution of systems. In the nearest future we plan to insert this construction into a GUI-based language for script specification.

Acknowledgements. The authors are grateful to Evgeny Petrovikh, Alexey Kovalenko, Dmitry Kharitonov for they helpful discussions. Special thanks to Linda Hein for technical help. The first author has also benefit from a grant from the Russian Fund for Basic Research No. 96-01-00177.

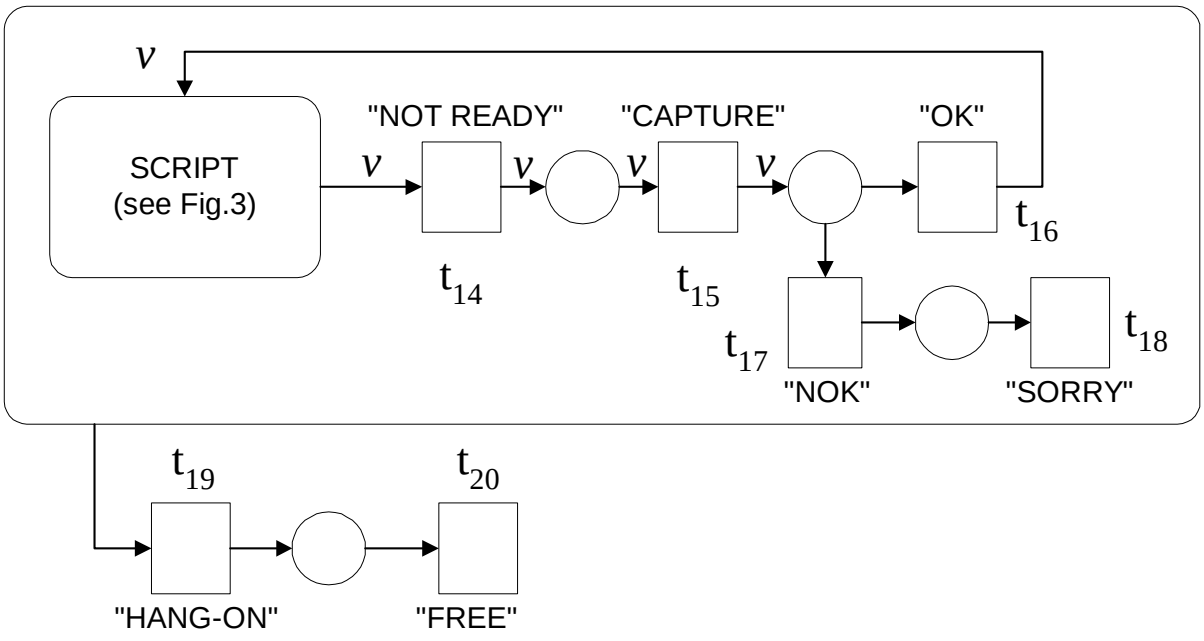

Figure 8: Script net with macroplaces

References

[1] N.A.Anisimov. *An Algebra of Regular Macronets for Formal Specification of Communication Protocols.* Computers and Artificial Intelligence, Vol.10, No.1 (1991), pp. 541–560.

[2] N.Anisimov, K.Kishinski, A.Miloslavski. *Petri Net Based Model for Design of CTI-applications,* to appear in Proc. of the conference "Computational Engineering on Systems Application: CESA'96", July 9-12, 1996, Lille, France.

[3] K.Jensen, *Coloured Petri Nets. Basic concepts, analysis methods and practical use,* Vol.1: Basic concepts, EATCS Monograph on Theoretical Computer Science, Springer Verlag 1992.

[4] K.Jensen, G.Rozenberg (Eds.), *High-level Petri Nets: Theory and Application*, Springer Verlag, 1992.

[5] E. Margulies. *Voice Processing Applications* (Flatiron Publishing, 1995)

[6] P.M.Merlin, D.J.Farber. *Recoverability of Communication Protocols — Implication of a Theoretical Study*. IEEE Trans. Commun. COM–24 (1976) 1036–1043.

[7] J.L.Peterson. *Petri Net Theory and the Modelling of Systems,* (Prentice–Hall Inc., 1981)

[8] W.Reisig. *Petri Nets: An Introduction.* EATCS Monograph on Theoretical Computer Science (Springer–Verlag, 1985).

[9] C. Sibertin-Blanc, *Cooperative Nets,* Lecture Notes in Computer Science (Springer Verlag, 1994), pp.471-490

[10] R.Walters. *Computer Telephone Integration* (Artech House, 1993)

A Basic notions

A net is a tuple $N = \langle S, T, F \rangle$ where $S = \{s_1, s_2, \ldots, s_n\}$ is a set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is a set of transitions such that $S \cap T = \varnothing$, $F \subseteq S \times T \cup T \times S$ is a flow relation.

For each $t \in T$ define its a pre-set of places as ${}^\bullet t = \{s | (s,t) \in F\}$ and a post-set $t^\bullet = \{s | (t,s) \in F\}$. Analogously, ${}^\bullet s = \{t | (t,s) \in F\}$ and $s^\bullet = \{t | (s,t) \in F\}$.

A marking of a net N is a function $M : S \to \{0,1,2,\ldots\}$.
Place/transition net or Petri net is a tuple $\Sigma = \langle N, M_0 \rangle$
where N is a net and M_0 is an initial marking.