Bull. Nov. Comp. Center, Comp. Science, 33 (2012), 59–68

Construction and simulation of Petri nets in the WinALT*

S.V. Piskunov, M. Ostapkevich

Introduction

Petri nets [1] are a research tool for systems consisting of interacting components. Petri nets are the most interesting in that they allow representation and studying the behavior of evolving parallel processes in a program or in a discrete device. Petri nets are used to describe an asynchronous composition of fine-grained algorithms [2, 3]. To this end a sample model of a Petri net based on a parallel flowchart [2, 3] is created in the form of a parallel substitution algorithm (PSA) [4]. A set of cells associated with places within a net is transformed by such an algorithm. Parallel substitutions are associated with transitions within a network. Each such substitution transforms the states of the cells that are set in correspondence with input and output places of the transition that is set. The resulting algorithm can be used to build a network of automata, on which is a control device for the operators included in the composition [2,4]. This transforms a Petri net into a working system, for example, a control unit of a special-purpose processor.

The WinALT system [5–7] can be used for the construction and research into the simulation models of such compositions on the computer. The distributive of the system and a collection of simulation models are available at the site [8]. The language of the system describes fine-grain structures and algorithms. It is based on the PSA. A cellular array is the main data object in the system. Its graphic image on the screen, for example, in a two-dimensional case is a rectangle of any finite size, composed of colored cells located along the vertical and horizontal axes. A color is a certain reflection of the state of a cell.

The objective of this paper is to describe the information technology for the representation of a Petri net in its conventional graphic form for the user within a flat cellular array. This technology allows user to observe transitions of markers between cells that correspond to places in a Petri net. The transformation of the states of such cells is performed by a model program that simulates the behavior of a Petri net.

*Supported by RAS under Grant 14.6.

1. Overview of the WinALT simulating system

The WinALT system was built and is being developed as a free access software suite. The software suite has an open architecture. New modules, which are `dll` files in Windows, can be added. These modules form libraries in the WinALT suite.

Another important feature of the system is that it is built as a system of visual programming. This means that it has a graphic mode providing developed tools of visual construction and debugging of simulation models. A simulation model is represented by a project in the system. The system main window contains sheets of the two kinds. The sheets of the first kind (kept in files with `.3do` extension) contain graphical objects—cellular arrays, templates of parallel substitutions. The second kind (kept in files with `.src` extension) is for simulation programs. Creating and editing graphic objects are performed by using menus, toolbars and dialogs activated by them. The dialogs have a recognizable appearance for the Windows users. Texts of simulation programs are created and edited with a text editor, whose functions are similar to those of a standard editor such as MS NotePad.

2. Constructing Petri nets in the WinALT

Let us describe two versions of constructing the Petri nets. The first one is fully based on using only graphic tools of the system. The second one uses external (as related to the WinALT) software tools as well.

2.1. The first version of constructing Petri nets. An example of a Petri net for constructing a representation as a cellular array in the WinALT is taken from [2]. The network is shown in Figure 1.

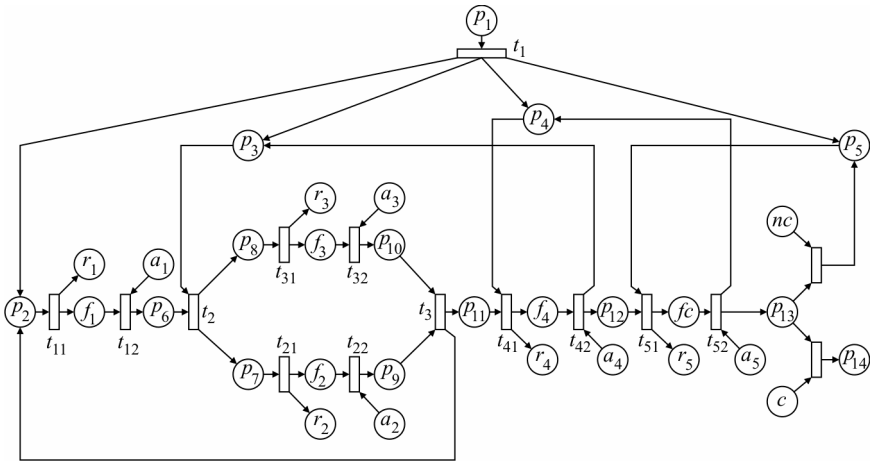

Figure 1. A source Petri net

Step 1. A flat cellular array with dimensions that according to the user's estimation are adequate to store graphic image of a Petri net is constructed. The array is created using the dialog `Create Object`. The type and the name of the cellular array, its size by the coordinate axes, its position on the sheet are set in this dialog.

Remark 1. If in the process of designing a network it happens that the size of the array is not sufficient for the user, one can always adjust it using the dialogs `Add layer(s)`, `Remove layer(s)` from menu `Edit`. These tools let the user add or remove a specified number of rows, columns or layers of cells of the array.

Step 2. A graphic image of a Petri net is painted just the same way as source data entering a cellular array. To assign a value into a cell of an array, one has to select this array by the dashed frame by pressing `Magic Wand`, and then using the mouse to move the input focus (denoted by a contrasting frame around the cell within the focus) to the cell that has to be edited. Then the dialog `Set Cell` is to be called and the cell name and value have to be set in it. By selecting a value, one sets the color of a cell on the screen. It is up to the user to decide which primitives are to be used to compose a net. For example, let us consider the following case. The gray rectangular areas with cells of size 2×7 denote a transition in graphic image of a net, while squares of size 3×3 composed of gray, dark gray and black cells (except for central white cell) correspond to a place in a net. The places and transitions, connected by directed arcs, which are gray lines with a one-cell thickness, arrows are denoted by dark gray cells. The cellular array `NET01` containing a Petri net image composed of color cells is depicted in Figure 2. In order to avoid the ambiguity in the subsequent discussion, let us assume that an image is located in the sheet `NET1.3do` of the project `PetriNetConv.wap`.

Remark 2. Tools of the WinALT system allow to improve the visibility of an image of a network. One can assign a name for a place or for a transition into a cell or write a context comment associated with a cellular array.

After completion of the two above steps we have obtained only a picture of a Petri net in a cellular array. And now we have to extract cells that set in correspondence with places in a net in order to build an algorithm of the net functioning. The structure of the WinALT called *virtual array* can be used to accomplish this task. Any subarray of a cellular array can be declared as a virtual array. Such an array can be used independent of its parent array in the language operators in the WinALT. But the results of transformations performed in a virtual array are always reflected in its parent array as well.

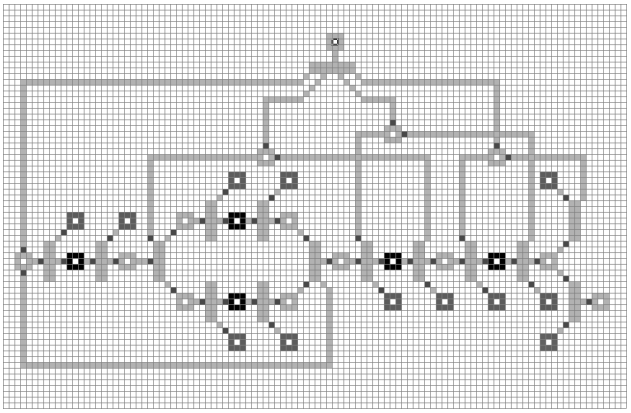

Figure 2. An image of a Petri net in the cellular array `NET01`

The construction of virtual arrays can be done via `Virtual` menu item. Its subitems provide operations for its constructing within a specified parent array, its deleting, showing and hiding its location in its parent array.

Step 3. Each cell with a place name is declared as a unicellular virtual array. As the notion of a virtual array is the principal one when describing the Petri net functioning, let us discuss the procedure of its construction in greater detail. The tools for its construction are depicted in Figure 3.

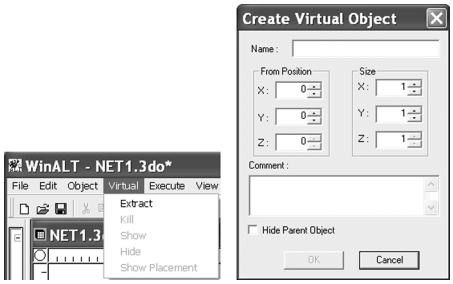

Figure 3. The tools for constructing virtual arrays

Clicking `Extract` item in `Virtual` menu activates the dialog `Create Virtual Object`. One has to determine the coordinates of the future virtual array in its parent array before the actual use of this dialog. For example, let the input focus be in the cell named `p1` (see Figures 1, 2). The coordinates of the cell in the focus are shown on the status line of the WinALT main window (Figure 4). The first coordinate corresponds to the horizontal axis (X), the second one is for the vertical one (Y), while the last one is for the axis that goes to the screen (Z).

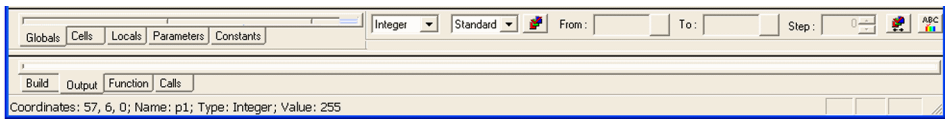

Figure 4. Showing coordinates $(57, 6, 0)$ of the cell named `p1` on the status line

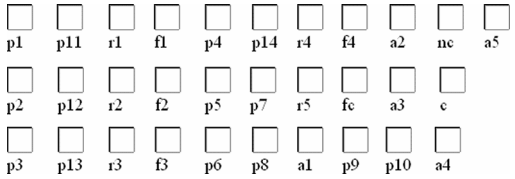

Figure 5. A set of unicellular arrays in correspondence with the positions in a Petri net

These coordinates are introduced into `From Position` fields of the dialog. As an array should to contain only one cell, all its dimensions are set to one. A name is assigned to the array. Here it is selected so as to coincide with that of a net place. The virtual array is extracted from its parent array and placed in the sheet `NET1.3do`. The unicellular arrays presented in Figure 5 are the result of the above procedure.

So, as a result of the procedure execution, we have obtained a cellular array which represents a Petri net in a user-friendly form and is the parent array containing virtual arrays that is set in correspondence with places in the Petri nets. These are only the states of these unicellular arrays, representing the presence or absence of markers, that are changed by the modeling program simulating the behavior of Petri nets. Simultaneously, all these changes are visible in the parent array.

2.2. The second version of the Petri net construction procedure. This version is interesting because it allows the user exploiting an external graphic editor for constructing a graphic image of a Petri net in the WinALT. Such external editors provide considerably more tools for image editing in comparison with what the WinALT does. Using them in combination with the WinALT extends its potential and attractiveness for the user. Let us select `MS Paint` as image editor.

The library of data formats [6, 7] was created by the developers of the system and is available in it. It includes a set of external modules that support certain subformats of `bmp` format. Images in these subformats can be used as ordinary cellular arrays in the WinALT. For example, the external module `bmp256u.od` provides the support for the subformat of images with 256 colors. With its help, image pixels turn into cells and their color codes become values of these cells. These cell values can be retrieved, modified. The cells can be used in parallel substitutions, i.e. using of `bmp` format does not essentially differ from using other formats available in the WinALT. The resulting cellular array has `image::` prefix in a sheet with graphical objects. However, the nature of a raster image imposes certain limitations on corresponding cellular arrays. First, one cannot create a cellular array with more than one layer (the size by Z axis is 1). Second, the cells can only have integer values in the range from 0 to 255 and cannot have names. Of course, all these limitations can be overcome if needed, as the image cells can be copied into an object of another kind that permits multiple layers

and cell naming. All the modules of the data format support are written in C and are available in the source texts. The sources are quite small and well commented, thus permitting the user to implement a support for own formats on their basis and to include them into the library of data formats.

Step 1. It substitutes Steps 1 and 2 of the previous procedure. A graphic image of a Petri net is created using `MS Paint` and saved into the directory of the Petri net model as a `bmp` file with 256 colors.

Step 2. This step coincides with the third step of the previous procedure.

Let us mention a possibility of using an external editor, which is often useful. In the case when an image of Petri net is available as a figure in an ebook, it can be captured from the screen (e.g. by `PrtScn` key), pasted into `MS Paint` and further transformed to a cellular array according to the above procedure. A sample of such a transformation is presented in Figure 6.

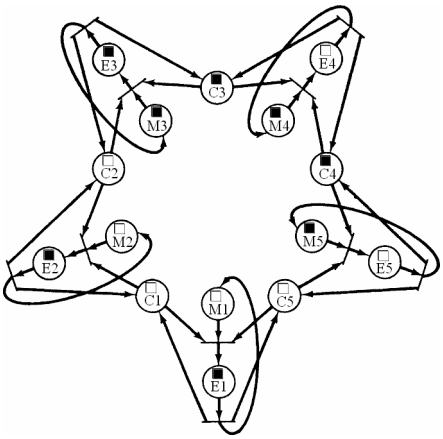

Figure 6. A graphic image of the Petri net built in WinALT using external tools

This cellular array is built on the basis of a picture of the Petri net [1] stored in `djvu` format. Virtual unicellular arrays that are set in correspondence with places of a net are located right in these very places. Their sizes are selected so that the user could conveniently observe the transitions of markers.

3. Constructing a simulation program imitating the Petri net behavior in WinALT

Data objects in the simulation program are the virtual cellular arrays that correspond to the Petri net transitions, so that the process of building a simulation program does not depend on the procedure invoked to build a cellular representation of the Petri net. Let us construct a simulation program for the Petri net with image presented in Figure 2. Let us note that this net is built on the basis of the correct parallel flowchart [3] and therefore is alive and safe. Being safe means that none of its places can hold more than one marker. This feature considerably simplifies writing a substitution command describing the transition activation. For example, activation of transition `t1` is described by the following substitution command:

$$\{(1,p_1)(0,p_2)(0,p_3)(0,p_4)(0,p_5)\} \to \{(0,p_1)(1,p_2)(1,p_3)(1,p_4)(1,p_5)\}. \quad (1)$$

The semantics of the command is as follows: if there is a marker in place p_1 and no markers in places p_2, p_3, p_4, p_5, then the marker in place p_1 is erased and inserted in places p_2, p_3, p_4, p_5. The commands for the rest of transitions in the net are written in a similar way. A set of such commands forms a PSA. A bunch of operators `in-at-do` corresponds to one substitution command in a program written in the WinALT simulation language [7]. Operator `in` from this bunch takes, in general, a list of the names of processed cellular arrays as its parameter. The parameter list of operator `at` defines the left part of the substitution, while that of operator `do` sets its right part. Substitution (1) transforms to a bunch of operators of the simulation program in Figure 7.

```
in (p1, p2, p3, p4, p5)
   at (marker,empty, empty, empty, empty)
      do (empty, marker, marker, marker, marker)
```

Figure 7

In this figure, `marker` is a unicellular pattern whose color (black, the value is 1) denotes the presence of the marker in the place, `empty` is a unicellular pattern whose color (white, the value is 0) denotes the absence of the marker in the place. The simulation program is composed of the block containing the list of bunches written for all transitions in the net. This block is placed in operator brackets `ex-end`. According to the syntax of the simulating language this construction denotes that all the substitution commands are iteratively applied to the processed information. The termination occurs when none of the substitutions is applicable. The window of the project `PetriNetConv.wap` is presented in Figure 8. It contains a simulation model of a pipelined device. The device contains four stages. Operator `F1` is executed at the first one. Operators `F2` and `F3` are executed simultaneously at the second one, while operator `F4` is executed at the third stage. At the fourth stage, the counter functions in the operator `fc`, which stops the pipeline reaches a preassigned value.

The interaction of stages is described by a Petri net, whose image is represented in the cellular array `NET01`. The main unit of the simulation program contains a procedure call `init()` and block `ex-end`, which in addition to the simulation program of the Petri net includes the procedure calls `F1()`, `F2()`, `F3()`, `F4()`, `Fc()`, `Fch()`. The procedure `init()` prepares the input data for the simulation model. In particular, it sets the marker into the cell named `p1`, which corresponds to the place p_1. The procedures `F1()`, `F2()`, `F3()`, `F4()` imitate the execution of operators `F1`, `F2`, `F3`, `F4` of the pipeline. The procedures `Fc()`, `Fch()` imitate the work of counter `fc`. Along with the virtual single cell arrays, which correspond to the places of the Petri nets, the procedure uses cell objects on the sheet `NET1.3do` below the cell array `NET01`.

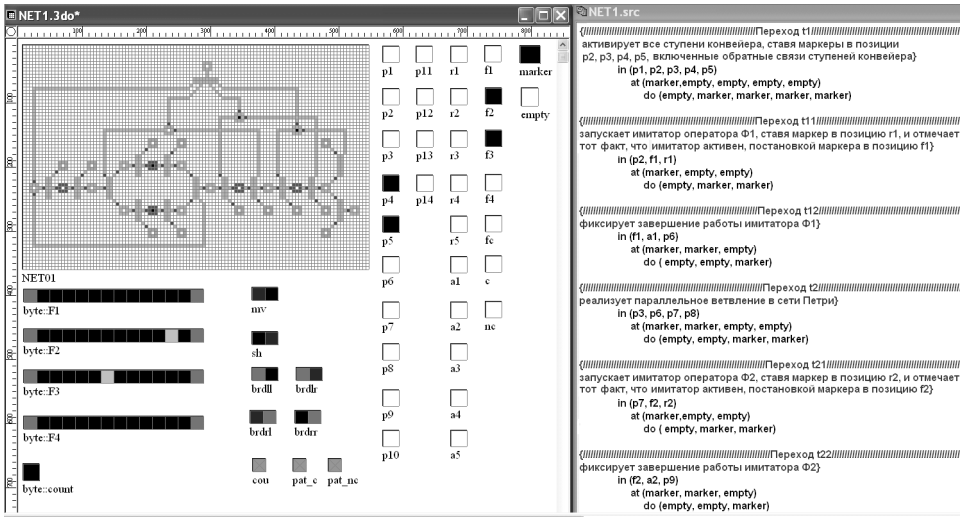

Figure 8. Project window of a model of a pipeline device

Let us demonstrate how the interaction between the control Petri net and the operators under its control is performed in a simulation model. In this model, the task of the simulator is to set a random number of steps of operator execution. The number of steps is in the range from 1 to k (k is a constant defined by the user). All simulators of operators have the same structure, so the interaction with only one of them, procedure `F1()`, will be described in detail. The procedure is presented in Figure 9. Let the marker be in the cell `p2` (its state is black). Then a bunch of `in-at-do` that corresponds to the transition `t11` removes the marker from the cell (casts it into the white state) and put markers in the cells of `f1` and `r1`. A bunch of operators `in-at-do` in the first `ch-end` block removes the marker from the cell `r1` of Petri nets and prepares to launch the simulator operator `F1` in its cell array `byte::F1`, setting the cell with coordinate 1 in the red state. Then the operator `if-end` using the condition `F1(1)=black` sets a randomly chosen cell in the cellular object `byte::F1` of simulator operator `F1` to the red state. The first bunch of operators `in-at-do` in the second block `ch-end` performs the motion of the red state in the working field till it coincides with the ending cell of the array `byte::F1`. The second bunch of operators places

```
procedure F1()
  begin
  show pause
    ch
      in (byte::F1, r1)
        at (brdll, marker)
          do (brdlr, empty)
    end {ch}
    if byte::F1(1) = red then
      byte::F1(1) = black; byte::F1((rand()) mod k + 1) :=red;
    end {if}
    ch
      in byte::F1
        at mv
          do sh
      in (byte::F1, a1)
        at (brdrl, empty)
          do (brdrr, marker)
    end {ch}
  end {procedure}
```

Figure 9

a marker in cell `a1` signaling the completion of the simulator. The bunch of `in-at-do` corresponding to the transition `t12` receives an opportunity to fire. It removes the markers from the cell `a1`, `f1`, and puts a marker to the cell `p6`.

Remark 3. As can be seen in Figure 8, the pipeline model imitates functioning of operators `F2` and `F3`. A cell in red state in cellular arrays `byte::F2` and `byte::F3` is visible as light gray in Figure 8.

Remark 4. The model is presented in section `Other models` of the model library at the site [8] and is available for download.

Conclusion

The technology of construction and simulation of the Petri nets in the WinALT system has been developed. It is assumed that it will be tested on nets of quite different types, and not just built on the parallel flowcharts, as it is shown in this paper. This means that in the general case, functional substitutions [4, 7] will be used to describe transitions. Currently, only two modes of simulation are available for the user: synchronous and asynchronous. In the synchronous mode all the applicable bunches of `in-at-do` are executed. In the asynchronous mode, only one applicable bunch is randomly selected. It is planned to provide a capability for the user to construct own modes of simulation and to include them into the library. A collection of simulation models of the Petri nets from a very wide range of applications should become the result of this study.

References

[1] Petersen J.L. Petri Net Theory and the Modeling of Systems. — Prentice-Hall, 1981.

[2] Custom Processor for High Performance Data Processing / V.E. Kotov, N.N. Mirenkov, eds. — Novosibirsk: Nauka, 1988 (In Russian).

[3] Achasova S.M., Bandman O.L. Correctness of Parallel Computation Processes. — Novosibirsk: Nauka, 1990 (In Russian).

[4] Achasova S.M., et al. Parallel Substitution Algorithm. Theory and Application. — Singapore: World Scientific, 1994.

[5] Piskunov S.V., Umrikhina E.V. Computer simulation of asynchronous composition of algorithms with fine-grain parallelism // Nauchny Vestnik NGTU. — 2007. — Iss. 3 (28). — P. 51–54 (In Russian).

[6] Piskunov S.V., Ostapkevich M.B. The simulating system of fine-grain algorithms and structures WinALT // Repository of Algorithms and Programs

SB RAS, Registration number PR11053, 2011-11-02. — http://fap.sbras.ru/node/2455.

[7] Ostapkevich M., Piskunov S.V. The construction of imitational models of algorithms and structures with fine-grain parallelism in WinALT // Lect. Notes in Comput. Sci. — Berlin; Heidelberg: Springer, 2011. — Vol. 6873. — P. 192–203.

[8] WinAlt Home Page. — http://winalt.sscc.ru/.