THE DEVELOPMENT OF PARALLEL SOFTWARE NAMED VISUAL MEMBRANE PETRI NETS

Author: Bing-lin Yang
Scientific Adviser: Doctor Habilitat, Profesor Emilian Gutuleac,
Ph.D. Associate Professor Aurelia Profir

Mathematics and Informatics Department, State University of Moldova
E-mail: BinglinYang@Gmail.com, AureliaProfir@yahoo.com

Abstract: *We are developing an useful and powerful software named Visual Membrane Petri Nets which gives graphical interface to design a Petri net model. This software could execute the Petri net model and calculate it on a parallel machine and has ability to analyze the reachability of Petri nets. This application is a cross-platform application. In this article, the details about developing the client of Visual Membrane Petri Nets is elaborated.*

Keywords: *Petri Net, Parallel Computing, C++, Qt, cross-platform, MPI*

1. An Introduction to Visual Membrane Petri Nets System

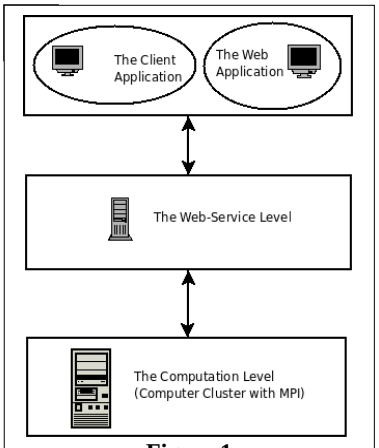

Figure 1

The Visual Membrane Petri Nets system consists of three parts, the client application, the web-service server and the computation server. The purpose of the client is offering a graphical user interface to let researchers designing Petri Net models and then sending the designed model to the web-service server. Web-service server is the middle level between the client level and the computation level. Web-service connects to the computation server (computer cluster) directly, thus, the model is sent to computation server though web-service, and the web-service return the result got from computation server to the client. The Computation level is as its name, which is responsible for computing the Petri net model in terms of the Petri net execution rules. A rocks computer cluster with 52 processors is used for calculating in our Petri Nets system, the parallel computing is implemented by using MPI. Figure 1 illustrates the three-level-structure of our Petri Nets system.

2. The Overview of The Client of Visual Membrane Petri Nets

We can see the client application is a very convenient tool for building Petri net models from figure 2. Researchers can conveniently use a mouse to build a Petri net model on a canvas and set the various properties of each node of the model in this software. We enriched the basic Petri net, therefor, the Visual Membrane Petri Nets can support high level and colored Petri net models. Besides these two points, we introduced an idea of membrane, which could divide a large Petri net model into plenty of subclasses. Based on this dividing, the computation in terms of Petri net execution rules of the model can be divided on various processes which run on different processor.

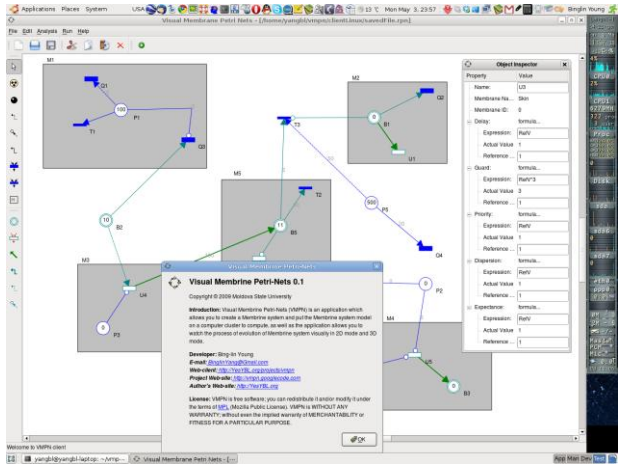

Figure 2

There are two main types of nodes offered in this tool. One is "discrete nodes", another is "continuous nodes". Just as their names imply, "discrete nodes" is a set of nodes that only can handle natural numbers, for example, tokens are only represented in discrete natural numbers.

Discrete places, Discrete arcs including test arc, inhibitor arc and normal arc, and discrete transitions including timed transition and instant transition, all of them build a discrete system for describing Petri net. "Continuous nodes" is a set of nodes that can handle real numbers, in our system, we use "level" instead of "token" as it can represent a continuous real number. Continuous nodes set contains continuous places, continuous transitions, continuous arcs, continuous test arcs and continuous inhibitor arcs. Those nodes build a continuous system for describing Petri net. One special arc must be pointed here — flow arc, which is used for connecting continuous place and discrete transitions such as timed transition or instant transition. We can connect discrete part of a Petri net to continuous part of the Petri net through flow arc. But the real number must be converted to a natural number before starting of the movement of tokens or level through this arc.

3. The Usage of Echo Node in This Software (Execution Rules)

Discrete place:

Function: For holding tokens, this kind of place can only hold natural number and its range must be in the interval between the value of its property of "CapacityMin" and the value of its property of "CapacityMax".

Connectible nodes: Test arc, Inhibitor arc and Normal arc.

Continuous place:

Function: For holding the value of level, this kind of place can hold real number and its range must be in the interval between the value of its property of "BoundMin" and the value of its property of "BoundMax".

Connectible nodes: Continuous arc, Continuous Test arc, Continuous Inhibitor arc.

Test arc:

Function: Test arc is used for controlling the status of a transition which is connected to the test arc. Test arc can not be used for carrying tokens. The status of a transition which is connected to a test arc can be enabled if and only if when the value of tokens of a discrete node which connected to the test arc is grater or equals the value of this test arc's weight value. Otherwise the status of the transition must be disabled.

Connectible nodes: Discrete place, Timed transition, Instant transition. (One way connection, place -> transition)

Normal arc:

Function: It is used for carrying tokens. Normal arcs can connect from a Discrete place to a Timed transition or Instant transition, also can connect to a Discrete place from a Timed transition or Instant transition.

- ,1In the case of connecting from a place to a transition, the Normal arc can carry tokens if and only if the value of weight property of the Normal arc is smaller or equals the value of tokens of the place which is connected to the Normal arc. In each step, the amount of tokens equaling the value of weight should be transported through this Normal arc.
- ,2In the case of connecting from a transition to a place, the Normal arc can carry tokens if and only if the status of the transition which is connected to the Normal arc, is enabled. In each step, the amount of tokens equaling the value of weight should be put into the Discrete place which is connected to this Normal arc on another side.

Connectible nodes: Discrete place, Timed transition, Instant transition. (Bothway connection, place <-> transition)

Inhibitor arc:

Function: It is used for controlling the status of a transition which is connected to the Inhibitor arc enabled or disabled. The status of a transition can be set enabled if and only if the value of weight property of the Inhibitor is grater than the value of tokens of a place which connected to the Inhibitor arc on another side.

Connectible nodes: Discrete place, Timed transition, Instant transition, Continuous transition. Inhibitor only can start from Discrete place, but can connect to any type of transitions. (One way connection, Discrete place -> any transition)

Continuous arc:

Function: Its function is similar to Normal arc, but It handles real number. Continuous arc may connect from Continuous place to Continuous transition or from Continuous transition to Continuous place.

Connectible nodes: Continuous place, Continuous transition. (Bothway connection, Continuous place <-> Continuous transition)

Flow arc:

Function: Flow arc is used for moving data (value of level) between Continuous place and Timed transition or Instant transition. Actually, this arc build a channel between Continuous nodes and Discrete nodes. Nevertheless, the type of data must be converted from real number to natural number before moving. This arc's execution rules are similar to Normal arcs'.

Connectible nodes: Continuous place, Timed transition, Instant transition. (Bothway connection, Continuous place <-> Discrete transition)

Continuous Test arc:

Function: Continuous Test arc is used for controlling the status of a Continuous transition which is connected to it, This arc can not be used for carrying data. The status of the Continuous transition which is connected to the arc can be set enabled if and only if the value of level property of the Continuous place which is connected to the arc is greater or equals to the value of weight property of the Continuous Test arc.

Connectible nodes: Continuous place, Continuous transition. (One way connection, Continuous place -> Continuous transition)

Continuous Inhibitor arc:

Function: Continuous Inhibitor arc is used for controlling the status of a Continuous transition which is connected to it. This arc can not be used for carrying data. When the value of level property of the Continuous place which is connected to the arc is lower than the value of weight property of this arc, then the status of the Continuous transition which is connected to this arc can be set enabled. Otherwise, the status should be set disabled.

Connectible nodes: Continuous place, Continuous transition. (One way connection, Continuous place -> Continuous transition)

Timed transition:

Function: Timed transition is used for controlling the moving of tokens from one place to another, or for introducing tokens into a Petri net system or leading out tokens from a Petri net system. Timed transition has "delay" and "guard" two properties. "Delay" represents the rate of speed of tokens' moving. For example, how many steps should be waited for one time transportation. "Guard" property is a boolean number, which holds the status of the Timed transition, enabled or disabled.

Connectible nodes: Normal arc, Test arc, Inhibitor arc and Flow arc.

Instant transition:

Function: The function of Instant transition is similar to Timed transition, but just transporting tokens without waiting. In other words, in the case of when it's enabled, it should transport tokens in each step. The "guard" property of Instant transition is same as Timed transition's.

Connectible nodes: Normal arc, Test arc, Inhibitor arc and Flow arc.

Continuous transition:

Function: Continuous transition is used for transporting value of level (real number), or for leading in data (the value of level) into a Petri net system or leading out data from a Petri net system. Continuous transition contains "delay", "guard" and "expentance" three properties. The functionality of "Delay" and "Guard" is similar to Timed instant. "Expentance" is used for "renewing" the value of weight of arcs which arc connected to the Continuous transition. The new value of weights should be the results of old value of weights multiply the value of expentance, but the new value of weights is only used for calculating the status of the Continuous transition for next step, not kept in the arcs which are connected to the Continuous transition.

Connectible nodes: Continuous arc.

Membrane:

Function: It's used for dividing the current Petri net model or system. Echo portion will be calculated in an unique process on parallel machine. Here, we have to point out that the whole canvas is also counted as a membrane. Thus, the number of membranes should be the number of all membrane nodes on canvas plus one.

4. The Designing of The Client Software

4.1 The Choosing of Programming Library

By reason of the client application is a graphical editor and is designed as a cross-platform software, the choosing of a cross-platform GUI (Graphical User Interface) programming library becomes very important. A good GUI programming library can observably reduce the workload of transplanting program which has been developed for a certain operating system to for other operating systems. Encapsulation for low-level APIs (Application Programming Interfaces) of operating systems makes transplanting applications becoming possible. Qt is a platform independent (cross-platform) graphics library for the development of applications with/without UIs (user interfaces). Qt supports various operating systems such as GNU/Linux, Microsoft Windows™, Mac OS etc. The abundant APIs and powerful integrity as a programming framework assure that we can achieve our purpose. For transportability, the portions which has been implemented as much as possible using original Qt APIs, however there are still some functions in modules like Network Socket unit and Calculator class need support from third-party programming libraries. Because the C++ programming language and the C programming language are with wonderful transportability, using programming libraries which are written in pure C or C++ is the critical factor since the Qt library are written in C++. Based on above considerations, the muParser mathematics library which are written in pure C++ and gSoap SOAP protocol implementation library which are written in pure C are being used in the software. And the result is good. Now we can compile source code into binary file for different operating systems with small modifications on source code.

4.2 The Logical Structure of The Client Software

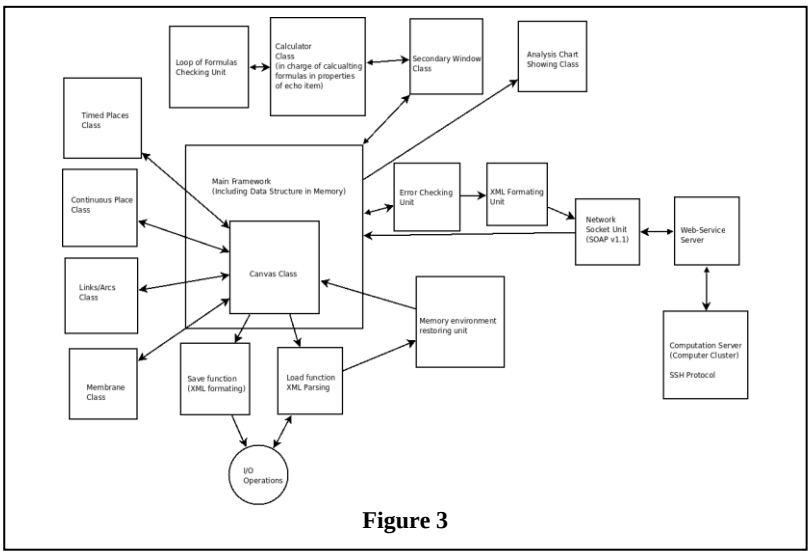

Figure 3

The logical structure of the client software is visualized depicted in figure 3. The client software consists of many program modules which are in charge of handling different functions for ensuring researchers building correct Petri net models. A common memory area in which the information related nodes are stored was designed in the application. The common memory area is very important in the software, because almost all program modules need to read and write data on this memory area. The directed arrows which are used for showing the connection between modules, and the direction represents data flow's orientation (interactions between modules).

4.3 The Details of The Program

Up to the present moment, the project contains more than 20 source code files (*.cpp, 25,000 lines of code). Generally, each C plus plus file is used for implementing a class or a module. Important source files are explained here.

Main framework is the interface container of this application, which contains menu bar, tool bars, canvas, status bar and property window as showed in figure 2. The MainWindow class of this program inherits QMainWindow class. In mainWindow.cpp file which implements MainWindow class, all function of menu commands are written here. Qt uses the notation of "action" associating functions with menu commands. First, create a new "action", then use "connect" function to associate corresponding function. "Connect" function is based on the signals and slots mechanism. The signals and slots mechanism is fundamental to Qt programming. It enables the application programmer to bind objects together without the objects knowing anything about each other.

In addition, all functions which respond to the mouse clicking on main window are implemented in this file.

Canvas class inherits QGraphicalScene class, which is implemented in canvas.cpp file. Such functions as "add", "delete" nodes (discrete places, continuous transitions, flow arcs and membrane etc) are written here. When a user click mouse on the canvas after selected a node on tool bar, those functions will be called. The principle is that canvas received a "clicked" signal, then goes to check the status of current environment (give an example, to check which node user has selected), after that, corresponding function will be called.

Files "node.cpp", "link.cpp", "trans.cpp" and "membrane.cpp" are similar, all of them inherit QGraphicalItem class. The shape of nodes and properties are defined in those files.

File "formulaCalculator.cpp" encapsulates APIs of muParser library, and generates a new class named "FormulaCalculator". This class can accept a formula in QString structure, and parse it, calculate it recursively, cause one formula could contain other formulas. FormulaCalculator must work with an instance of "FormulaChecker" class which is implemented in file "formulaChecker.cpp". The main purpose of "FormulaChecker" class is that checks whether the current formula contains a sub-formula (maybe in many levels) which contains the current formula or not. The algorithm of this feature is that the program dynamically builds a temporary multiway tree in memory, this multiway tree consists of items which have relations in formula strings. Then use depth-first traversal algorithm to verify whether the tree is valid or not. Figure 4 represents this idea. In Figure 4, each English letter represents a formula. The mathematic definition is:

A: {B, C, D}
B: {E, F, G}
C: {H, I}
D: {J}

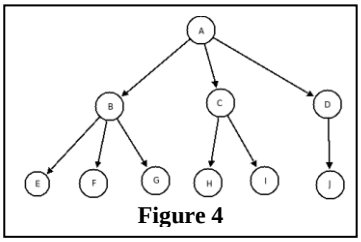

Figure 4

which means that the expression of formula A contains formula B, formula C and formula D, and the expression of formula B contains formula E, formula F and formula G, the rest may be deduced by analogy. This tree shows a valid tree. However, if a researcher input a wrong formula expression in formula G, as a result, formula G contains formula A, then the mathematic definition will be like:

A: {B, C, D}
B: {E, F, G}
C: {H, I}
D: {J}
G: {A}

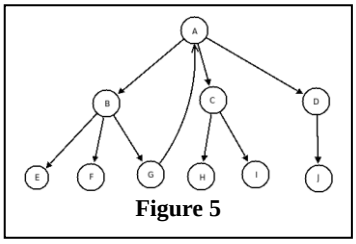

Figure 5

In this case, an endless loop has been created for recursive calculating formulas. Figure 5 depicts this situation.

The code statement for checking multiway tree is below:

```
void FormulaChecker::travelFormulaTree(QString formulaName)
{
     QList<QString> formulaNames;
     formulaNames = getSubFormulaNames(formulaName);
     formulaStack.push(formulaName);
     if (formulaNames.size() > 0)
     {
          for (int i = 0; i < formulaStack.size(); ++i)
          {
               for (int j = 0; j < formulaNames.size(); ++j)
               {
                    if (formulaStack.at(i) == formulaNames.at(j))
                    {
                         flag = false;
                         errorMsg =
                        "You built an endless loop in " + formulaStack.at(i);
                         return;
                    }
               }
          }
          for (int i = 0; i < formulaNames.size(); ++i)
          {
               travelFormulaTree(formulaNames.at(i));
          }
          // finishing at leaf node
          formulaStack.pop();
          return;
     }
     else
     {
          formulaStack.pop();
          return;
     }
}
```

File "io.cpp" is written for saving Petri net model which researchers have done on hard-disk and reading the saved file from hard-disk. Extensible Markup Language (XML) is used as the format of save-file. All information, no matter user input or program needs, about each nodes will be saved, in this way, when user opened a saved file, the program can rebuild common memory area by using the information saved in XML file.

File "network.cpp" encapsulates the APIs of soap protocol instance which is generated by gSoap. Therefore, the instance of MainWindow class can pass Petri net model in QString data type to network socket module, cause the soap protocol instance only accepts the C string. The encapsulation simplifies the complexity of programming.

Network socket module is in charge of sending model to web-service server and receiving result from web-service.

All functions which are used in properties window (secondary window) are implemented in file "properties.cpp". File "propertiesWin.cpp" defines the interface of properties window.

The program uses Model–View–Controller (MVC) software architecture to display and edit properties of nodes. But Qt combines the view and controller objects, thus, programming in Qt for programmer is like using Model/View architecture. And Qt introduces a notation of delegate, a delegate renders the items of data. The default number editor in Model/View architecture of Qt doesn't allow users input real number with more than two digits of fractional part. Hence, in the file "propEdit.cpp", we create a new delegate which inherits QItemDelegate class, so that this new input box can determine the type of a property, if a property belongs to a discrete node, then, the input box can only accept natural number, but if a property belongs to a continuous node, then, input box can accept real number even with 15 digits of fractional part.

File "runDialog.cpp" defines an interface of a dialog box which allows users to send a Petri net model to web-service server for computing. Researchers can specify how man steps should be simulated on computer cluster in this dialog box.

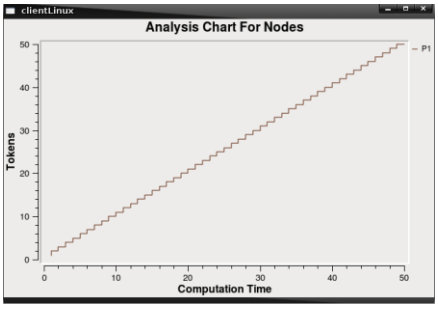

Figure 6

After computing the Petri net model, researcher can use simulation dialog box to get the result. Simulation dialog box is implemented in file "simulationDialog.cpp". The instance of SimulationDialog class calls the functions in network socket module and ChartWindow module to give users a final analyzed marking changing chart of places. Figures 6 shows a marking changing graph.

5. Parallel Computing on Computer Cluster Using MPI

The membrane is used for dividing Petri net models into many portions which are running as unique processes on different processor, in this way, we utilize parallel computing to calculate large Petri net models. Based on execution rules of Petri net theory, checking the status of every transitions must be done before going ahead into next step, wherefore Petri net models are flow dependent and anti dependent. In this situation, we have to synchronize data among all processes in each step. Due to above reasons, MPI_Allgather and MPI_Barrier programming interfaces are critical functional calls. A code snippet of parallel computing with MPI is below:

```
MPI_Allgather(sendStruct, mv.numberLocation, inewtype, receiveStruct,
               mv.numberLocation, inewtype, MPI_COMM_WORLD);
for(int i = 0; i < mv.numberMembrane; i++)
       if((mv.numberMembrane > rank) && (rank == i))
           beginExchange(mv, rank, receiveStruct, loc);
for(int i = 0; i < mv.numberLocation; i++)
     sendStruct[i].tokenCount = loc[i].TokenCount;
MPI_Allgather(sendStruct, mv.numberLocation, inewtype, receiveStruct,
               mv.numberLocation, inewtype, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
for(int i = 0; i < mv.numberMembrane; i++)
     if((mv.numberMembrane > rank) && (rank == i))
         finalExchange(mv, rank, receiveStruct, loc);
```

7. Conclusion

To develop a complex cross-platform software is not an easy job, especially when the software allows users input arbitrary content. Errors checking and exception handling is very important for robustness of an application. Making a good scheme in the beginning of development is significant. The choosing of programming libraries and encapsulation also could greatly improve the efficiency of development. The message format between different parts of a whole project must be defined well. Parallel computing requires more attention to dividing a task and the data exchanging among tasks.

Bibliography

1. James L. Peterson. *Petri Nets*. Computing Surveys, Vol 9, No. 3, September 1977.
2. Cheng G. L., An H, Cheng L, Zheng Q. L., Dan J. L. *The Practice of Parallel Computing Algorithm*. Higher Education Press, Jan 2004, p. 58-97, p. 107-127