Software Development Tool with Petri net and Object-oriented programming language: Net oriented System Description Language

Han Yong Gil*, Min Hyok, Kim Song Hyok, Choe Yong Su, Song Kwang Hyok,
Choe Tae Hyok and Kim Jing Yong
Faculty of Mechanical Engineering, Kim Chaek University of Technology, Yonggwang St, Kyougu-Dong No.60, Pyongyang, Democratic People's Republic of Korea

Abstract

This paper aims to describe the development and application of Net-oriented System Description Language (NSDL), a new and independent tool of software development combined with advantages of Petri net and object-oriented programming language VB.

Unlike previous tools, the transitions of custom controls such as Textbox, Table, Graph, Button and Checkbox, etc. and the extension (or restriction) of place, transition and arc were introduced to improve modeling capability of Petri net.

NSDL was enhanced the flexibility, convenience and extensibility of the software development by Visual Basic(VB) language support based on Microsoft Net Framework 4.0 library that is easy to use and learning.

Approximation of BP neural net was carried out to validate NSDL's effectiveness in three manners.

NSDL can be used in the development of software or modelling of complex information systems with its great modeling capability.

Keywords: Software development tool, Petri net, Object-oriented programming language, Complex information system, Neural net

1. Introduction

It is of a great significance in the software development and modeling of the complex information system to project new software development tools of convenience and practicality.

The software and complex information system in view of modeling can be regarded as a time/event driven system with the complicated structure of objects. The Object-oriented programming language is fitted to the information system development by ability of the encapsulation of state and procedures to manipulate the state, inheritance, polymorphism and code reuse. But in case of the complex systems, lots of time and efforts are needed to describe their structural and logical relations exactly. It is, therefore, a matter of great importance to realize the diagramming on logical action process along with the visualization of programming.

On the other hand, Petri Nets, the diagrammatic and mathematical tool to model discrete event driven system, represent the static structure of system by places, transitions and arcs, and also do the dynamic behavior of system by firing and moving tokens. The ordinary Petri nets are truly simple in structure, but complexity and state explosion may occur when modeling the complex systems. [1] Many scientists have proposed lots of different extensions and restrictions based on the ordinary Petri net. [2]

Also other are proposed software development tools and methodology of the complex information system modeling combined with Petri nets and object-oriented programming language. [3-14]

* Corresponding author
Email addresses: YG.HAN@star-co.net.kp (Han Yong Gil)

The modeling capability of such tools, however, needs to improve in order to solve problems of system modeling in practice.

Hence, we projected the NSDL by making effective combination of the advantage Petri net and object-oriented programming language VB.

Capability of modeling of our tool was improved significantly through the integration of diagrammatic model, functional code model, and interface model thanks to the introduction of the transition of custom controls.

Table 1 below shows the differences between the proposed NSDL and the other tools.

Table 1. Comparison of tools combined with Petri net and object-oriented language

Name of tool	object-oriented language	Diagram Editor	Code Editor	Model Compiler	Hierarchical Structure	Custom Control	Net Analysis
THORNs	C++	✓	✓	✓	✓	-	-
GPenSIM	MATLAB	-	-	-	-	-	✓
RENEW	Java	✓	✓	✓	✓	-	✓
SNAKES	Pathon	-	-	-	✓	-	✓
PNlib	Modelica	-	-	-	✓	-	✓
SimHPN	MATLAB	-	-	-	✓	-	✓
NSDL	VB	✓	✓	✓	✓	✓	-

✓ supported, - not supported

2. Net-oriented System Description Language NSDL

Net-oriented System Description Language NSDL is a new software development tool which combines Petri net with object-oriented programming language VB based on Microsoft. NET Framework 4.0.

Its formal notation is as follows.

$$NSDL = [P,\ T,\ A,\ M,\ F,\ \theta,\ O] \quad (1)$$

Where,

- P is the finite set of places,
- T is the finite set of transitions (includes the transitions of custom controls),
- A is the finite set of arcs,
- M is the finite set of markings (includes the object tokens),
- F is the finite set of functional code,
- θ is the finite set of attributes (delay time after firing, firing rate, priority, weight, capacity, type of elements, competition extraction setting and color, etc.) and
- O is the finite set of standard and user defined objects modeled with functional codes.

The NSDL models the system as follows.

- In the upper level of system,
  - Hierarchical diagrammatic models according to the mutuality and the logical action sequence of system components are constructed.
  - Graphical User Interface models can be configured according to the demands of the user. (optional)
- In the lower level of system,
  - Standard and user defined functional code model of upper level model elements are edited.
  - Properties of diagrammatic and interface model elements are explained by functional codes during the simulation run.

In NSDL, modeling elements including short in/out place, equal place, partial system and some capabilities for model library management and error detecting, model compiling were introduced to improve its flexibility and convenience, extensibility, productivity.

2.1. Main configuration

The main interface of NSDL is composed of title bar(①), menu bar(②), tool bar(③), heading line(④), modeling element window(⑤), diagram editing window(⑥), executable code editing window(⑦), project explorer window(⑧), property window(⑨) and setting bar(⑩) for diagram edition.

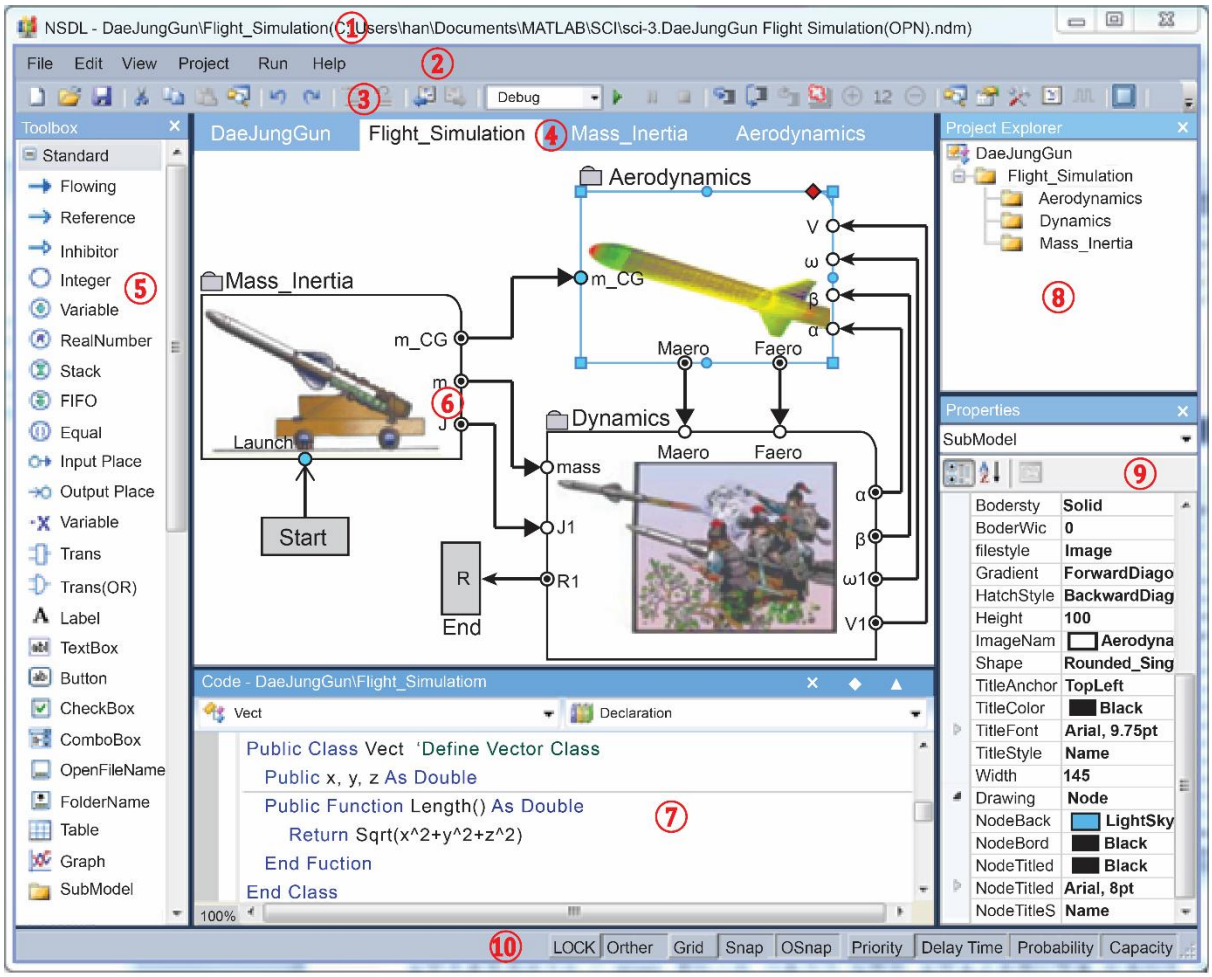

Fig.1. Main graphical user interface of NSDL

Use case Diagram, Class Connection Diagram, Algorithm Diagram are shown in following figures.

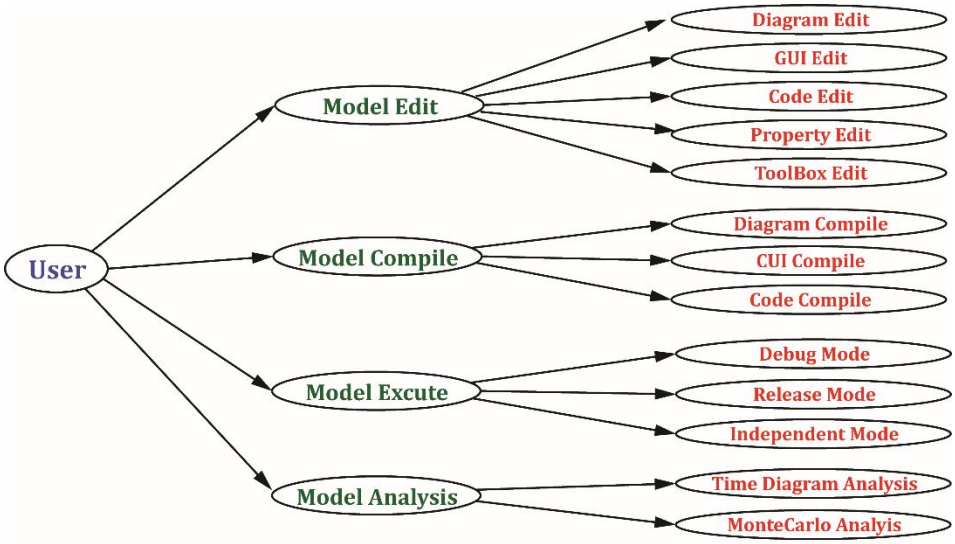

Fig.2. Use case Diagram

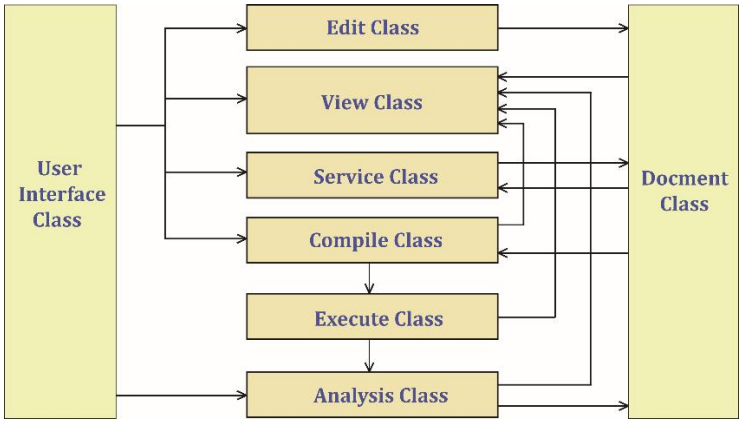

Fig.3. Class Connection Diagram

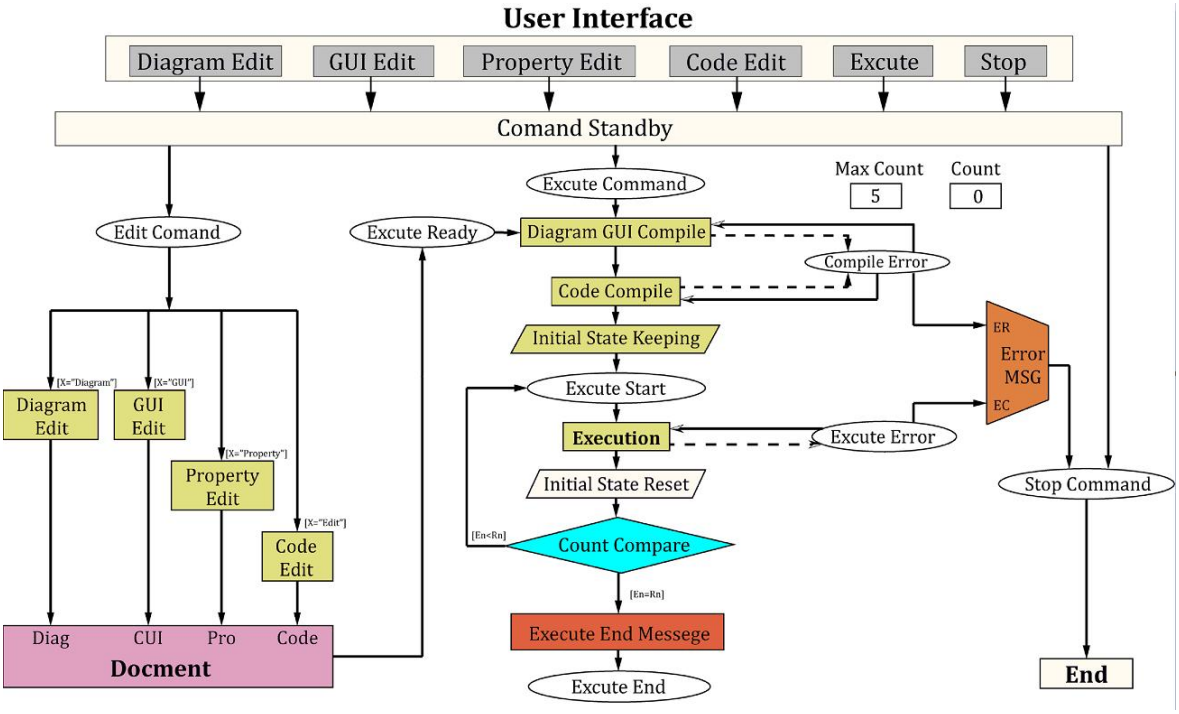

Fig.4. Algorithm Diagram

2.2. Modeling element

Modeling elements are contained place, transition, arc, subsystems, control and functional code.

2.2.1. Place

The place reflects the state of system and there are integer place, variable place (includes object), real number place, stack place, FIFO place and equal place according to the contained contents and uses. (Figure 5) The integer place, the stack place and the FIFO place have their own capacities.

NSDL can explain the state of continuous or hybrid systems as well as the state of discrete systems by using these places.

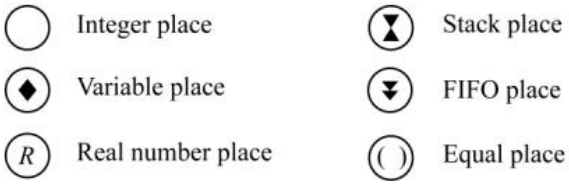

Fig5. Places.

2.2.2. Transition

Transitions are fired according to the states of input places, input/output arc properties and it's use condition functions. And then corresponding functions are executed as a certain event happens.

The transition is a processing unit that executes the corresponding function as a certain event happens according to the states of input places and inner terminals (variable or simplified input/output place terminal) added to it. (Figure 6)

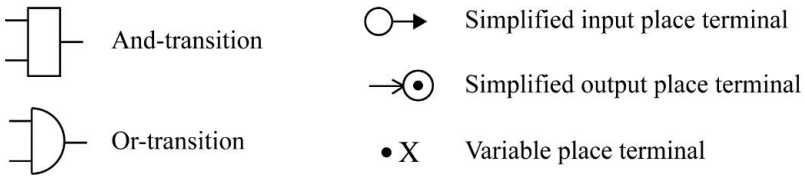

Fig.6. Transition and inner terminal added to it.

For convenience of modeling, And-transition and Or-transition are used and they can be linked with simplified input/output places or variable terminals.

Boolean, integer, real number, string and object type, etc. can be used in these transitions while variable terminals are only used as defined variables here.

Several transitions of custom controls which are widely used in interface modeling, such as Graph, Table, Text Box, Button, Check Box, Combo Box can be used in NSDL to improve modeling capability. (Figure 7)

Such transitions can execute the action according to the standard transition and itself private function. The attributes of firing time, firing rate, priority and the like can be set in all the transitions.

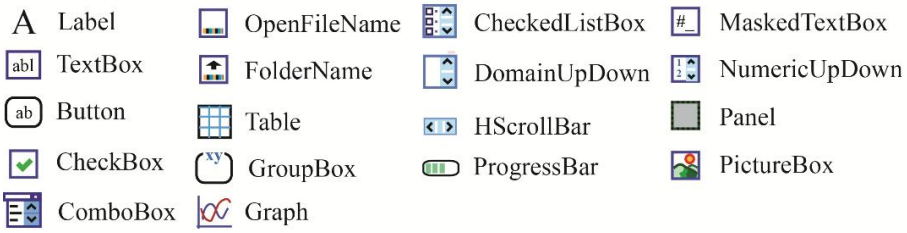

Fig.7. Typical transitions of custom controls.

2.2.3. Arcs

Arc is a linking element that explains the relationship between places or transitions and has two types: input arc and output arc.

According to the manner of information transmission, input arc can be classified into three types: Flow arc, Reference arc, Inhibitor arc.

Flow arc transfers information of input place to transition connected to it which can be fired, and removes it in the input place.

On the other hand, a reference arc keeps the information in the input place after the relevant transitions are fired.

Inhibitor arc comes into being the possible state of firing when there is no information in the input place.

Output arc outputs the information. Also multiplicity, flow quantity, conditional expression and output function can be defined in the output arc. (Figure 8)

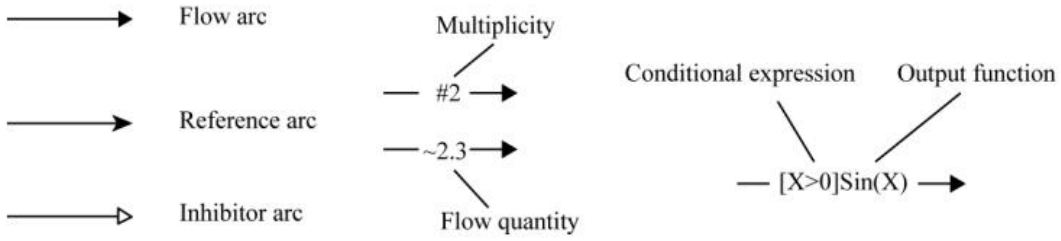

Fig.8. Type of arc.

2.2.4. Subsystem model element

The hierarchical structure of complex information system can be modeled with subsystem model elements, simplified input places and simplified output places. (Figure 9)

Input/output place terminals are only connected to the places of subsystem model.

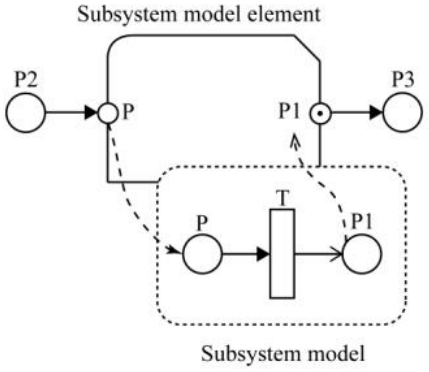

Fig.9. Expression of input/output terminal of subsystem element.

2.2.5 Functional code model

Functional code model describes the functions which model elements created in the upper level of systems execute.

They can use not only the arbitrary function and procedure that Framework4.0 library supports, but also following several custom functions and procedures in NSDL.

- Sub Initialize(): a procedure for initialization.
- Sub Terminate(): a procedure occurring as the transition ends acting.

- Sub InputValue(): a procedure occurring as the transition inputs information.
- Sub OutputValue(): a procedure occurring as the transition outputs information.
- Sub Action(): a procedure that describes the action of transition.
- Function UserCondition(): a function that processes an event occurring as the model checks firing condition.

And so on

The system variables, procedures and functions that NSDL support in default are:

- Systime: Current running time of model
- Modelpath: Current model path
- Mainform: Main formwindows
- Resetmodel: Reset the initial condition of the diagrammatic model
- Runcount: Run count at the repeat run of model
- Runstyle: Run style (Debug, Release, Independent)
- Updatemodel: Redraw of the current diagrammatic model

And so on

2.3. Firing rule

The state change of system in NSDL is realized by the firing of transition i.e. the occurrence of an event.

The firing of transition follows the standard firing condition of Petri net and the type of transition (And, Or, Custom control action), the kind and attributes of input arcs (conditional expression, multiplicity, flow quantity). (Figure 10)

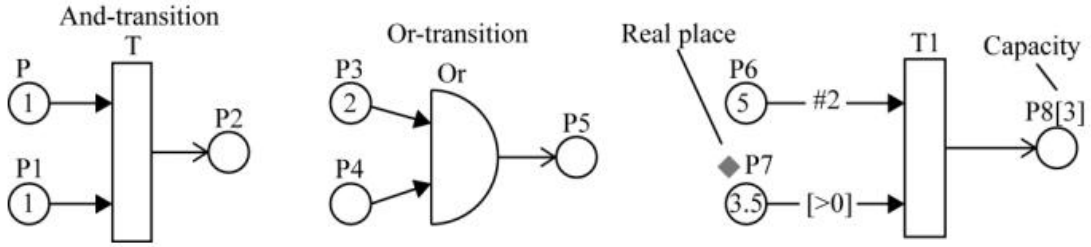

Fig.10. Firing conditions of the transition (possible states).

For the transitions which the function "User Condition ()" is defined, their possible state of firing can only be in case that the above-mentioned conditions and this function value conditions are satisfied.

3. Examples of modeling

3.1. Neural net model

Forward propagation process of BP neural net is as follows. [15]
First, middle layer (hidden layer) output can be determined as

$$U_j = \sum_{i=1}^{q} W_{ji} \cdot I_i + \theta_j, \quad H_j = \varphi(U_j) \tag{2}$$

where

- q is the number of input layer neural cells,

- W_{ji} is the weight coefficient from input layer neural cell i to j,
- I_i is the output if input layer neural cell i,
- θ_j is the threshold of middle layer neural cell j,
- H_j is the output if middle layer neural cell j and
- $\varphi = [1 + \tanh(x/u_0)]/2$ is the activation (sigmoidal) function.

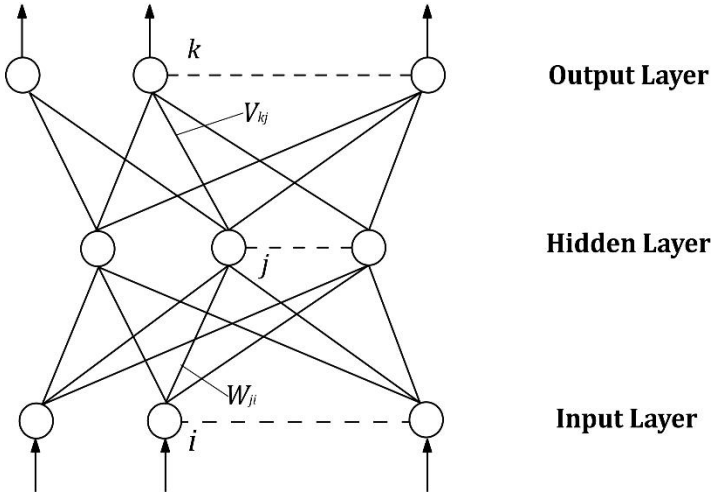

Fig.11. Structure of BP neural net

Next, output layer can be determined from middle layer as

$$S_k = \sum_{j=1}^{l} V_{kj} \cdot H_j + \gamma_k, \quad O_k = \varphi(S_k) \tag{3}$$

where

·l is the number of middle layer neural cells,

·V_{kj} is the weight coefficient from middle layer neural cell j to k,

·γ_k is the threshold of output layer neural cell k and

·O_k is the output if output layer neural cell k.

Back propagation (learning) process of BP neural net by successive correcting method is as follows.

From the difference between learning data T_k and output O_k of output layer, thresholds and weight coefficients of middle and output layer can be obtained as follows.

$$\delta_k = (O_k - T_k) \cdot O_k \cdot (1 - O_k), \ \sigma_j = \sum_{k=1}^{m} \delta_k \cdot V_{kj} \cdot H_j \cdot (1 - H_j) \tag{4}$$

$$V_{kj} = V_{kj} + \alpha \cdot \delta_k \cdot H_j, \ \gamma_k = \gamma_k + \beta \cdot \delta_k \tag{5}$$

$$W_{ji} = W_{ji} + \alpha \cdot \sigma_j \cdot I_i, \ \theta_j = \theta_j + \beta \cdot \sigma_j \tag{6}$$

where m is the number of output layer neural cells.

At each learning process approximation error of neural net is

$$E = \frac{1}{|S|} \sum_S \sum_{k=1}^{m} |O_k - T_k| \tag{7}$$

where s is the set of learning data.

3.2. The examples of BP neural net

The BP neural net approximation of exclusive-or (XOR) was modeled with the three different methods.

3.2.1. Method using diagrammatic model

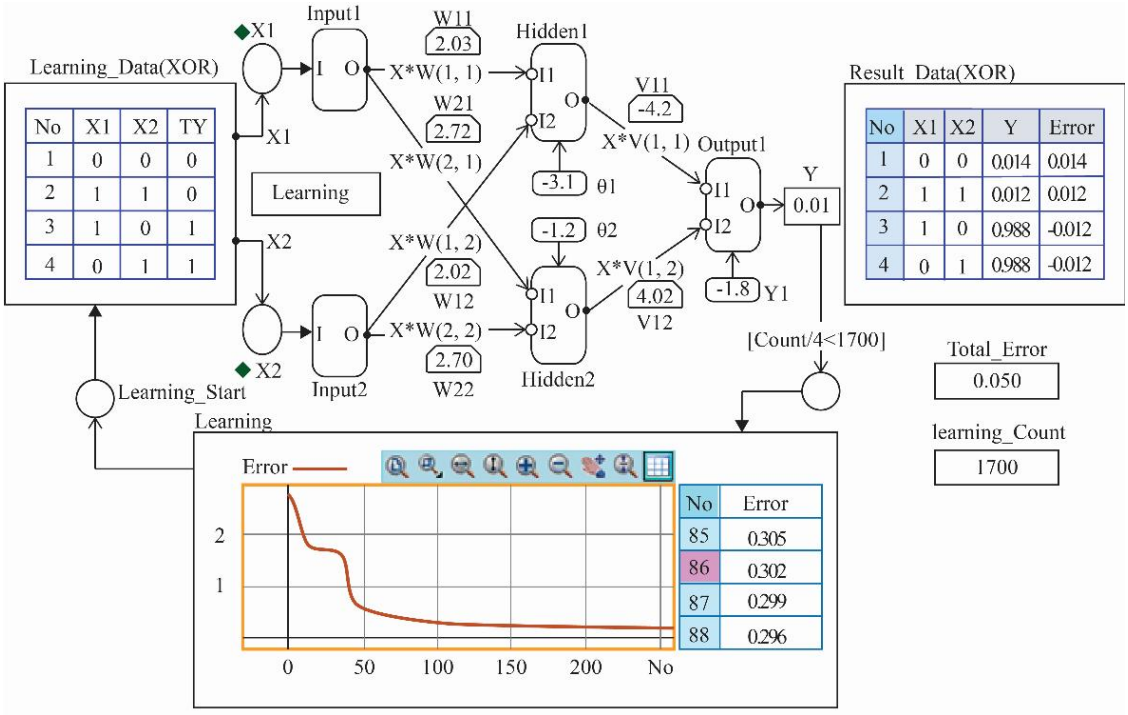

Fig.12. Method using diagrammatic model.

Figure 12 shows convergence state of BP neural net approximation.

Here, number of input layer neural cells, output layer neural cells, output layer neural cells are 2, 2, 1 respectively.

Each neural cell was modeled with transition which have simplified input places and output variable terminals.

Calculations according to each cell transition was modeled with functional codes.

Weight coefficients between layers of neural net was modeled by using output function of output arcs and corresponding values was displayed in 'Textbox'.

Two transitions of custom controls ("Teaching_data" and "Teaching") was used to provide the visualization of teaching data, output data, and convergence process of the error.

The transition "Teaching_data" is a transition of custom control (table) which inputs the given learning data, whereas "Teaching" is one (graph) which shows the back propagation process of BP neural net-learning process, and the convergence process of the error on graphs and tables.

Functional code according to the transition of graph control is as follows.

```
Public Sub Action(ByRef Cancel As Boolean)
  …
  ' ------- Calculate Error of Output Layer ------
  For n = 1 To ON_NUM
    wk = OT_OT(n)
    wkb = Teach(n) - wk        '---- Difference Between Teaching Signal and Output-----
    Error_Val = Error_Val + Math.Abs(wkb)
    Del_OT(n) = wkb * ruo * wk * (1 - wk)    ' ------ Error of Output Layer ------
  Next
  ' ------ Calculate Error of Hidden Layer ------
  For k = 1 To HN_NUM
    sum = 0
    For m = 1 To ON_NUM
      sum = sum + Del_OT(m) * V(m, k)
    Next
    wk = OT_HD(k)
    Del_HD(k) = sum * ruo * wk * (1 - wk)   ' ----- Error of Hidden Layer -------
  Next
```

```
    ' ------------- Teaching -----------------
      ' ------- Hidden Layer - Output Layer ------
    For k = 1 To ON_NUM
       For m = 1 To HN_NUM
          V(k, m) += α * Del_OT(k) * OT_HD(m)
       Next
       γ(k) += β * Del_OT(k)
    Next
      '-------  Input Layer - Hidden Layer ------
    For k = 1 To HN_NUM
       For m = 1 To IN_NUM
          W(k, m) += α * Del_HD(k) * OT_IN(m)
       Next
       θ(k) += β * Del_HD(k)
    Next
  ' --------------- Teaching Graph Transition display the error history----------
    If Count Mod 4 = 0 Then
       Approximate_Error.Text = Error_Val / 4
       Teaching.AddRowData(Error_Val)
    End If
  End Sub
```

3.2.2. Method using diagram and user interface integration

Above mentioned method does model every cell with the transition and thereafter it makes the modeling rather complicated in case of a great number of cells.

Introducing the object of BP neural net, however, enables the modeling to be easier as it can be modeled with one transition. (Figure 13)

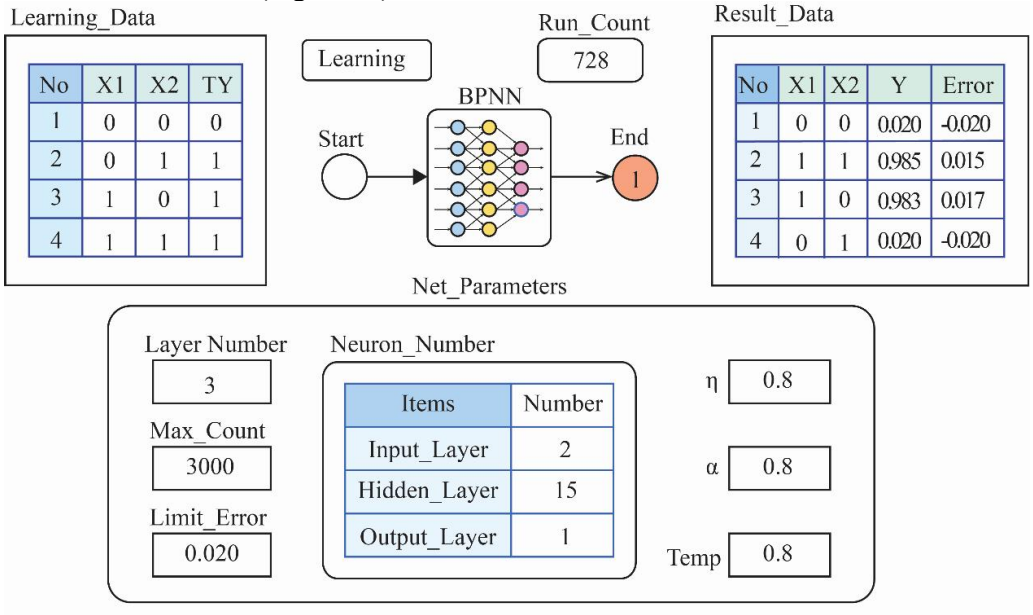

Fig.13. Method using BP neural net object (diagram and GUI integration).

The transition "BPNN" is the standard transition of BP neural net object and the interface "Net_Parameters" ensures the dialog between user and neural net.

Functional codes of "BPNN" transition is as following.

```
Public Const MAXLAYERS =3    ' Maximum Number Layer of NN (include The Input Layer)
Public Const MAXNEURONS = 120   ' Maximum Number of Hidden Layer Cell
…
 ' ------------- Teaching Code of Transition BPNN -----------------
```

```
Public Sub Action(ByRef Cancel As Boolean)
    Dim BPnet As New BackProp   '    BackProp is a Class of the BP_Neural_Network
    Dim Converged As Boolean = False
    Converged = BPnet.Train()     ‘ BP_Neural_Network Train
End Sub

' ------------- Define BackProp Class-----------------
Public Class BackProp
    Dim Neuron(MAXLAYERS, MAXNEURONS) As Double
     …
  ' -------- Sigmoid Function --------
    Public Function Sigmoid(ByVal Net As Double, ByVal Tempr As Double) As Double
        Dim su As Double
        su = Net / Tempr
        Return 0.5 * (1 + Math.Tanh(su))
    End Function
   ' -------- Backforward Error Calculation about Hidden Layer Cell -------
    Public Function HCalcδ(ByVal lyr As Byte, ByVal j As Integer, ByVal dNETj As Double) As Double
          …
    End Function
  ' --------  Backward Process of BP Net -------
    Public Sub AdaptWeights()
        …
    End Sub
  ' -------- Convergency Determinatin -------
    Public Function CalcErrors() As Boolean
         …
        Return ConvergeFlg
    End Function
   ' -------- Forward Process of BP Net -------
    Public Sub RunNet()
        …
    End Sub
       …
  ' -------- Teaching Process Function -------
    Public Function Train() As Boolean
        …
        Dim Converged As Boolean = False
        While (Converged = False) And (CurrIter < MAXITER)
            GetInputs()
            RunNet()
            Converged = CalcErrors()
            AdaptWeights()
        End While
        Return Converged
    End Function

End Class
```

3.2.3. Method using MATLAB integration

In this example BP neural net approximation was performed by using MATLAB thanks to the advantage of VB programming language which makes it easy to integrate with other application programs.

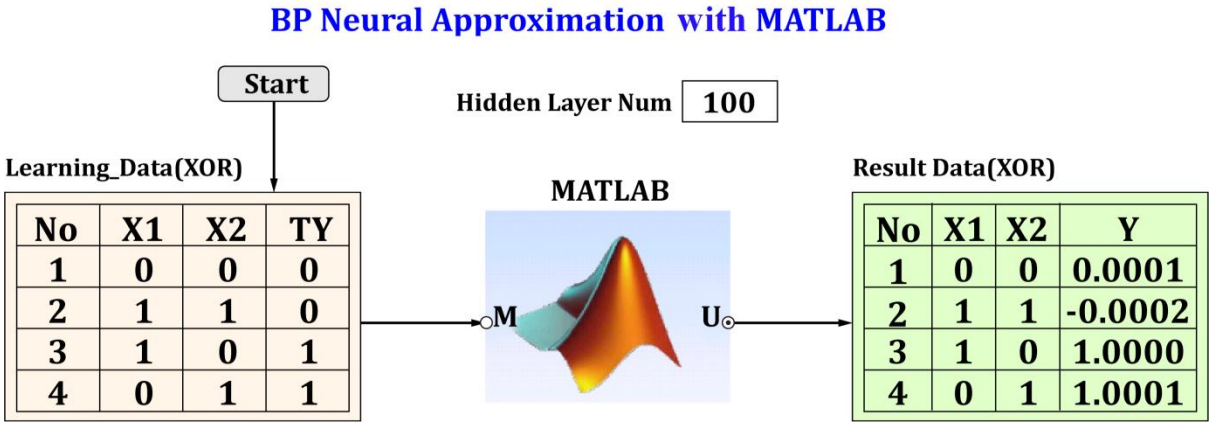

Fig.14. Method using MATLAB integration.

Here, the standard transition "MATLAB" has its functional code like below.

```
Dim MatLab As Object   ' ------- Define MATLAB Object Variable
Dim Row, Col, Hneu_Num  As Integer
 …
' ------- MATLAB Transition Code -------
Public Sub Action(ByRef Cancel As Boolean)
        …
     MatLab.Execute("cd " & Modelpath)
     MatLab.Execute(InString)          ' ------ Teaching Data Input
     Postprocess(MatLab.Execute("[Z]=BPNN(p, t, Hneu_Num)"))  ' ---- Executing BPNN .m File
End Sub

 ' ---- Creating MATLAB Object Variable -----
Private Function CreateMatlab() As Object
        …
  Return CreateObject("", "Matlab.Application")
        …
End Function

 ' ------ Teaching Data Input Function -----
Private Function InString() As String
  Dim St As String
    …
   St &= HNeuron_Num.Text
   Return St
End Function

 ' ------ Result Postprocess ------
Private Sub Postprocess(ByVal Result As String)
        …
   Dim Res() As String = Split(Result, " ")
   For k = 0 To Res.Length - 1
      If IsNumeric(Res(k)) Then
         Dim A1 As Double = Val(Res(k))
         U(i, 2) = A1
      End If
   Next
End Sub
```

MATLAB BPNN m file codes are as following.

```
      function [Z]=BPNN(p, t, neu)
          net = newff(p,t,neu);
```

```
    net = init(net);
    net.divideFcn = '';
    net.trainParam.show = 50;
    net.trainParam.lr = 0.05;
    net.trainParam.epochs = 300;
    net.trainParam.goal = 1e-5;
  [net,tr]=train(net,p,t);
  Z= sim(net,p);
End
```

All the results of these three methods were matched well.

Table 2 illustrates the other validated examples of NSDL.

Table 2. Additional examples

No	Name of model	Content	Model	Simulation method
1	Help	Help model of NSDL	Deterministic	Virtual time
2	Sum	Sum model from one to a thousand	Deterministic	Virtual time
3	Excess And Low Voltage Blocker	Model of Excess And Low Voltage Blocker with the control transition of textbox	Deterministic	Real time
4	FMS	Model of flexible manufacturing system consisted of a NC lathe, a NC milling machine, a NC grinding machine, a robot, a work-piece storage, three storage and a finished storage	Stochastic	Virtual time
5	UUEV Power Circuit Reliability	Model for calculating reliability of UUEV power circuit with Monte Carlo method	Stochastic	Virtual time
6	Inverse Matrix Calculation	Calculation of inverse matrix hybrid Petri net with MATLAB	Deterministic	Virtual time
7	Ordinary Differential Equation Solve	Motion analysis of mathematical pendulum with Runge-Kutta method	Deterministic	Virtual time
8	Flight Simulation of Shell	Flight simulation of “Dae Jang Gun” shell	Deterministic	Virtual time
9	**AHP**	Analytic Hierarchy Process Tool	Indeterminate	Virtual time

4. Conclusion

The NSDL which has been newly projected in combination of Petri net with object oriented programming language can be conveniently used for the development of software or modeling of complex information processing systems.

The introduction of transition of custom controls and the extension or restriction of place, transition and arc provides the possibility of the convenient and efficient modeling by integrating diagrammatic model, the functional model and the interface model so that it improves capability of modeling and communication for user.

Moreover, the hierarchical structure of system and the information exchange between subsystems can be described easily, and the requirements such as concurrency, parallelism, asynchronous and distributed character of systems can be explained sufficiently.

Thus NSDL could be somewhat easily used in modeling of complex information systems.

Acknowledgment

The authors would like to thank Prof. Dr. Kim Gwan Sik who is a boss of NSDL development team, all developers (Kim Jong Nam, Han Chol Min, Choe Ok Chol, Kim Un Il, Cha Chong Song and Jo Song Gun) that participated in the development of NSDL.

Also, we would like to thank all developers of the convenient and powerful Microsoft.NET Framework 4.0.

References

[1] Wolfgang Reisig, Understanding Petri Nets, Springer-Verlag Berlin Heidelberg, 2013.

[2] Rene David, Hassane Alla, Discrete, Continuous and Hybrid Petri Nets, Springer-Verlag Berlin Heidelberg, 2010.

[3] Frank Köster, Stefan Schöf, Michael Sonnenschein, and Ralf Wieting. Modeling of a Library with THORNs, Springer-Verlag Berlin Heidelberg, 2001.

[4] Olaf Kummer , Frank Wienberg, Michael Duvigneau, Lawrence Cabac, Michael Haustermann, David Mosteller, Renew- User Guide, University of Hamburg, April 28, 2022.

[5] Reggie Davidrajuh, Modeling Discrete-Event Systems with GPenSIM, Springer Briefs in Applied Sciences and Technology, 2018.

[6] Lawrence Cabac, Michael Haustermann, David Mosteller, Software Development with Petri Nets and Agents: Approach, Frameworks and Tool Set, Preprint submitted to Science of Computer Programming, December 12, 2017, 27 pages, https://doi.org/10.1016/j.scico. 2017.12.003.

[7] Kindler, Ekkart; Bergenthum, Robin, Algorithms and Tools for Petri Nets - Proceedings of the Workshop, AWPN 2017, DTU Compute-Technical Report-2017 Vol. 06.

[8] Sabrina Proß Bernhard Bachmann, Sebastian Jan Janowski Ralf Hofestädt, A NEW OBJECT-ORIENTED PETRI NET SIMULATION ENVIRONMENT BASED ON MODELICA, Proceedings of the 2012 Winter Simulation Conference, GERMANY, 2012.

[9] Erik Kuˇcera, Oto Haffner, Peter Drahoš, Roman Leskovský and Ján Cigánek, PetriNet Editor + PetriNet Engine: New Software Tool For Modelling and Control of Discrete Event Systems Using Petri Nets and Code Generation, Applied Sciences, 2020, 10, 7662; 19 pages, doi:10.3390/app10217662.

[10] Fabrice Kordon, Daniel Moldt. Introduction to the special issue from Petri Nets 2016, Science of Computer Programming 157, 2018.

[11] András Vörös, Dániel Darvas, Ákos Hajdu, Industrial applications of the PetriDotNet modeling and analysis tool, Preprint submitted to Science of Computer Programming, September 11, 2017, 24 pages, http://dx.doi.org/10.1016/j.scico.2017.09.003.

[12] Zhenhua Yu, Xiao Fu, Yu Liu, Jing Wang and Yuanli Ca, Modeling and Analyzing Software Architecture Using Object-Oriented Petri Nets and π-calculus, Petri Nets: Applications, India, 2010, 57-71.

[13] Armen Bagdasaryan, Systems theoretic techniques for modeling, control and decision support in complex dynamic systems, Artificial Intelligence Resources in Control and Automation Engineering, 2012, 15-72.

[14] Shmyrin Anatoly Mikhailovich , Lukyanova Elena Aleksandrovna, An algorithm for determining the structural parts of Petri models-based complex systems, International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies, Volume 10, No. 18, 2019, Paper ID: 10A18H, 10 pages http://TUENGR.C0M/V10A/ 10A18HM.pdf DOI:10.14456/ITJE- -MAST .2019.252.

[15] Zuohua Ding, Yuan Zhou, and Mengchu Zhou, Modeling Self-Adaptive Software Systems with Learning Petri Nets, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBER-NETICS: SYSTEMS, Vol.6, No.5, 2021.