TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.1, January 2014, pp. 157 ~ 166
DOI: http://dx.doi.org/10.11591/telkomnika.v12i1.3901

Automatic Building Process of Self-Closed Modified N-tree

Yibing Liu, Xiaodong Zhu, Ying Chen, Yu Li, Ning Deng
College of Software &College of Computer Science of Jilin University
Jilin Univeristy, China
*Corresponding author, e-mail: zhuxd@jlu.edu.cn

Abstract

Some features of prevailed workflow like Petri net and Grid workflow make them cannot adapt to the dynamic operation. So, we proposed a modified N-tree model to control a workflow. Modified N-tree model can remedy some problems exist in these prevailed workflow models. Firstly, we approve the proposed modified N-tree model is self-closed. This feature makes sure that this workflow can accomplish its tasks, when we change nodes of a well-running modified N-tree workflow before or while its execution. It is the prerequisite of dynamic characteristics of modified N-tree model. And, then we give a method to change this tree dynamically based on the self-closed merit. Finally, based on the dynamic characteristics of this model, we give a method to build on this N-tree workflow model automatically by using left root (LR) analysis method proposed by Mr. D.Knuth. This is the most important performance of this model.

Keywords*: Modified N-tree model; workflow; automatically process; state machine*

1. Introduction

The workflow management technology is one of emerging technologies in computer application domain in recent years, which had achieved the enhancement production organization level and the efficiency goal, and widely applied to various fields through the process integration. A perfect product data management (PDM) system can keep track of the masses of data and information during the entire lifecycle of product development [1]. It also ensures that the right information is available for the right person at the right time and in the right format. Such as the nodes of workflow, the product-related information controlled by system includes part definitions and other design data, engineering drawings, product specifications, bills of materials, failure report etc.

Nowadays, the enterprises want to use this new technology to rein the process of whole producing. This process includes the purchasing of source, the producing of products, transmitting of information and so on. In different enterprise, the process is always different, even in the same enterprise for different processes of producing. So it gives us a problem that the statically traditional workflow cannot adapt to their needs. [1-4]

We need to define a dynamic workflow model to tackle it. In some previous researches, people mostly focus on Petri net and Grid workflow primarily. They all have some prominent features to build on a workflow. Petri net is distributed, concurrent and asynchronous [5,6]. While Grid workflow has the following three features: distribution, service interaction and dynamic. Especially the grid workflow, it can be applied into web service. However, there are two problems exist in these models:

(1) Since the Grid workflow and Petri net have complex definition and attributes, their dynamic control process are always complicated.

(2) People put more efforts on massive material tackling, but they ignore the automatically building. To a very complex productive process, the workflow will be very big. If we build on the workflow model manually, some mistakes may be leaved in this process.

To solve the first problem, we need to build on a new workflow model. Our former research has shows that a well defined N-tree model can adapt to the workflow perfectly and gives a process to build on this tree. To simplify the dynamic control process, we refine the N-tree workflow model and give a modified N-tree model to build on workflow. Then approve that modified N-tree model have dynamic property. It is valid for any changes of a consequently

Received May 23, 2013; Revised Aug 8, 2013; Accepted Aug 26, 2013

producing-workflow. For any changes, adding and deleting sub-trees, of a well-defined workflow, especially when it was processing, this procedure can lead to a result. It means when the workflow have been commenced we can add and delete any sub-tree, and this will lead to a result. An unqualified result is better than null result in productive process, because we can renovate this unqualified product by modifying this workflow.

For a complex productive process, we need to build on a complicated workflow to control this procedure. Until now, most workflow procedures were built manually, and some unrecognized blunders may happen in this procedure. To restore these problems will lead to extra expenses and delay the work in process. So an automatically building process is necessary to avoid unrecognized errors.

2. Modified Definition of N-tree Model

A complex model will lead a complicated dynamic control procedure. In order to prove the dynamic procedure we need an improved and modified definition of N-tree model. Proposed modified N-tree has the following merit:

(1) It is very simple and it only has node type, node model, and node state.

(2) By canceling some non-related attributes of previous definition, we can make the self-closed proving process more clearly.

Workflow unit mode has defined the rules for sequences of executing tasks in the processes of management. In this paper, each task is depicted as a node in N-tree model and every unit mode consists of one or more nodes. The node of a tree should have different types, processing models and states.

2.1. Types of Node

In order to define a modified N-tree, we need two types of node show in table 1.

2.2. Processing models of Parent node

In the following table 2, we will give four needed processing models of parent node.We will give some examples in Fig. 1, so we can see four processing models more clearly.

Table 1. Types of Node

Type	Meaning	Description
R_n	Parent	R_n can be parent node and child node, but it cannot be a leaf node of any tree. It is just used to control the skipping between its child nodes, and don't contain the practical producing process.
lf	Leaf	Just can be a leaf node of any tree. All vocational works conserved in this kind of nodes.

Table 2. Processing Models of Parent Node

Model	Meaning	Description
S	Sequence	If the model is sequence, its child nodes will be executed by their order.
L	Loop	If the model is loop, its child node was executed by order. After that, if the loop conditions are effective, its child nodes should be executed by order again.
B	Branch	If the model is branch, only one of its child nodes will be executed
P	Parallel	If the model is parallel, all of its child nodes will be executed at the same time

2.3. States of Parent and Leaf node

Workflow model must have recorded each state of nodes to achieve automatic executing. Table 3shows all kinds of states we need in this paper.

With the definition of the node and details showing in three tables, we can see that the child nodes of one parent node are unordered. So we can give definition of a modified N-tree.

An N-tree can be expressed as T=(root, R, L, Cr, Clf). Root means the root node, it just contains one element; R is a collection of parent nodes; L is a collection of leaf node; Cr collects the relationship between parent node; Clf collects the relationship between parent node and leaf node.

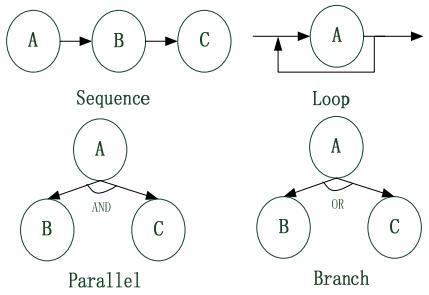

Figure 1. Examples of four processing models

Table 3. States of Node

State	Meaning	Description
E	Executing	This work have been commenced
N	Non-executing	This work cannot be executed, since the condition were not fulfilled
W	Wait	Wait the judgment of conditions
F	Finish	The work has been executed

3. Merits of Modified N-tree Workflow

With the development of modern manufacture, one factory can produce different kind of products and these goods produced by the same enterprise may be very similar. If we use a static workflow, we need to design many different workflow procedures to come out different products. It will be a huge cost, and also very complex for managers. Compared with these shortages in static workflow, the dynamic workflow can be easily reused. One workflow procedure can be easily reused by other similar products through a bit of changes. Before we give a dynamic method to change the N-tree model, we need to approve an essential merit—N-tree model is self-closed.

3.1. Modified N-tree is self-closed

A dynamic workflow model has many advantages, however, it has problem at the same. When one changes the workflow procedure, he can't make sure the workflow procedure lead to a determined result. To solve this problem, we need to approve that this N-tree workflow model is self-closed.

An N-tree is self-closed means to any tree, its root node, a parent type node, can change from F state to E state. A modified N-tree workflow can by showed as $T=(R_0, (R_0,R_1,\ldots,R_n), (lf_1,lf_2,\ldots, lf_k), Cr, Clf)$. We define three functions in this tree as:

$$State(R_i)\ (0<=i<=n) \tag{1}$$

this equation means: the state of R_i node;

$$State(lf_i)\ (0<=i<=k) \tag{2}$$

this equation means the state of lf_i node;

$$Type(R_i)\ (0<=i<=n) \tag{3}$$

this equation means the type of R_i node.

Then the following four steps approve that the N-tree workflow model is self-closed.

Step 1. We approve that an N-tree, which only has one parent node as its root and several leaf nodes as its child node, is self-closed. This tree can be shown as $T=(R, (R), (lf_1, lf_2, \ldots, lf_k), Cr, Clf), Cr=\varnothing, Clf=(R\rightarrow lf_1lf_2\ldots lf_k)$

(1) *Type(R)= S or P*, when State(R)=E, we set $state(lf_i)=W, (0<=i<=k)$, then set $State(lf_i)$ and execute these leaf nodes by their order. When all leaf nodes' state become F, we set *State(R)=F*.

(2) *Type(R)=L*, when *State(R)=E*, we set $State(lf_i)=W, (0<=i<=k)$, then set $State(lf_i)$ and execute these leaf nodes by their order. When all leaf nodes' state become F, we set *State(R)=W*. If the conditions are fulfilled, set *State(R)=F*, else execute (2) again.

(3)*Type(R)=B*, when *State(R)=W,* based on the conditions choice lf_i from (lf_1,…,lf_k) and set *State(lf_i)=W*. Then set *State(R)=E*, execute lf_i node. Set *State(R)=F*, after it have been finish.

Step 2. We construct a tree with one root node R, and the R just has parent node, R_1,R_2…,R_n, as its child node. R_i (1<=i<=n) just have leaf node to be their child nodes. We approve this tree is self-closed. We can show the tree defined above as *T=(R,(R, R_1,…, R_n), (lf_1, lf_2, …, lf_k), Cr, Cl)*

(1)Type(R)=S or P, until its all child nodes' state become F, we can set *State(R)=F*. Since each of its child nodes belongs to one situation in 1, so its child nodes' state can change to F from E. Hence the state of R can change from E to F.

(2) *Type(R)=L*, until its all child nodes' state become F, we can set *State(R)=W*. Based on the conditions, if all conditions are fulfilled, this step is finished; else, set *State(R)=E*, change its child nodes' states become W and execute them by their order. Since each of its child nodes belongs to one situation in 1, so its child nodes' state can change to F from E. When the state of all child nodes become F, set *State(R)=W*, repeat step (2) again.

(3) *Type(R)=B, State(R)=E*, based on the conditions choice one child R_i *(0<=i<=n)* and set *State(R_i)=W*. Then set *State(R)=E* and execute R_i node, since each of its child nodes belongs to one situation in 1, so its child nodes' state can change to F from E. Then set *State(R)=F.*

Step 3. We construct a tree with one root node R, and R has parent nodes and leaf nodes, R_1,R_2,…,R_n, as its children. R_i(1<=i<=n) only has leaf nodes as its children. We approve this tree is self-closed. This tree can be shown as *T=(R, (R, R_1, … , R_n) , (lf_1, lf_2, … , lf_k), Cr, Cl)*, *Type(R)= S or P or L or B.* If lfi_h, … , lfi_k (|*h-k*|>=*1)* are contiguous, add one child node R_i which is parent to R and set *State(R_i)=State(R)*. We let lfi_h, … ,lfi_k point to R_i, then we get a new tree. From Step 2 we can know that if we set State(R)=E, its state can change to F finally.

Step 4. Based on above mentioned three steps, we can construct a tree and it is self-closed.

From the above four step, we can get three rules of self-closed merit.

Rule 1: R is root node of tree T, and it has child node R_m, … , R_n. We treat its child nodes as root of sub-tree T_i *(n<= I <=m)*. When all of these sub-trees can be self-close, then tree T is self-closed.

Rule 2: Any tree can be self-closed.

Rule 3: A tree T has parent node R. This node has leaf node lf_1,…,lf_k. *Type(R)= S or P or L or B*. If lf_{ih}, … , lf_{ik} (|*h-k*|>=*1)* are contiguous, add one child node R_i which is parent to R and set *State(R_i)=State(R)*. We let lf_{ih}, … ,lf_{ik} point to *Ri*, then we get a new tree. From Step 2 we can know that if we set *State(R)=E*, its state can change to F finally.

3.2. Dynamical change method

To change a workflow procedure of one product to another, we may need to delete some redundant procedures and add other insufficient processes. So the dynamic procedure should include dynamically adding and deleting.

(1) Dynamically adding:

The following steps are dynamical adding:

Step 1. R_i is a parent node of a tree *T*. Let R_i become root of tree T_i, then T_i=(R_i, (R_i, R_{i1}, … , R_{ij}), (lf_{i1}, lf_{i2}, …, lf_{ik}), Cr_i, Cl_i). R_i has child nodes R_{in}, …. , R_{im} and lf_{ii}, ..., lf_{ij}. Based on Rule 1, we treat R_{im}, … , R_{in} as a root of sub-tree T_k *(m<= k <=n),* and based on Rule 3, we can get that tree is self-closed.

Step 2. T_j=(R_j, (R_j, R_{j1}, … , R_{jh}), (lf_{j1}, lf_{j2}, … , lf_{jm}), Cr_j, Cl_j), let R_j becom a child node of R_i node of tree T_i. Then we get a new tree T_{ij}= (R_i, (R_i, R_i, R_{i1}, … , R_{ij}, R_j, R_{j1}, R_{j2}, … , R_{jh}), (lf_{i1}, … , lf_{ik}, lf_{j1}, … , lf_{jm}), $Cr_i \cup Cr_j$, $Cl_i \cup Cl_j$)

Step 3. Based on Rule 2 we can get that R_j can change to F from E. So tree T_j is self-closed.

Step 4. Since the sub-trees of T_i which the root node is child nodes of R_i can be self-closed, and the tree T_j is self-closed, so based on Rule 1 we can get that T_{ij} is self-closed.

(2) Dynamically deleting:

The following steps are dynamical deleting:

Step 1. R_i is a parent node of a tree T. Let R_i become root of tree T_i, then T_i=(R_i, (R_i, R_{i1}, … , R_{ij}), (*Ifi1*, lf_{i2}, …, lf_{ik}), Cr_i, Cl_i). R_i has child nodes R_{in}, …. , R_{im} and lf_{ii}, ..., lf_{ij}. Based on Rule 1,

we treat $R_{im}, \dots, R_{in}$ as a root of sub-tree T_k $(m<= k <=n)$, and based on Rule 3, we can get that tree is self-closed.

Step 2. Delete a child node of R_i, named $R_{ik}(i<= k <=j)$, then we get a new tree T_{ik}. Based on Rule 2 the sub-trees which the root node is the other child node of R_i, are self-closed. Based on Rule 1, tree T_{ik} is self-closed.

Step 3. Based on Rule 3, delete any leaf node of T_i, the tree T_i is still self-closed.

4. Error Tackling

In any workflow, an error may happen because of different factors, such the lack of finance or materials. Based on the previous experience, the factories should know which reason lead to the problem. So we give the Error tackling function to solve the problems.

We define $WF=(T, E)$ to manifest a workflow model. In this model, T stands for the tree model defined before. $E = (e_1, e_2, e_3, \dots, e_k)$ stands for a collection of errors tackling functions of leaf nodes in T. The errors may happen in the Leaf node of T. Since we need to apply this tree model to supply chain, we must tackle errors in workflow. For a supply chain, the error is because of the shortage of materials.

(1) $WF=(T, E)$, $T=(R_0, (R_0, R_1, \dots, R_n), (If_1, If_2, \dots, If_k), Cr, Clf)$, $E=(e_1, e_2, \dots, e_k)$

(2) R_i is a parent node of tree and it has leaf node If_i, when we tackle the If_i and an error happened, we use e_i to tackle this problem.

5. Automatically Build Up A Modified N-tree Workflow

We want to give a method to come true that just give some logical relationship and necessary parent and leaf nodes to build on a N-tree workflow automatically. We refer the LR(k) analysis method which has been defined by D.Knuth in 1965. We need to build on the state machine, and then construct action form and goto form. After that we will give a method to construct a N-tree workflow automatically.

Definition 1 *Let $A \rightarrow \alpha\beta$ and $A \in R$, $\alpha,\beta \in R \cup L$, then we call $R \rightarrow \alpha \cdot \beta$ a program of LR*

Definition 2 *Let IS become a program of LR, we call IS(x) is project of IS respect to x.*

$$\begin{aligned} IS(x)=\{&A\rightarrow \alpha X\bullet \beta \mid A\rightarrow \alpha \bullet X\beta \\ &\in IS, X\in (L\cup R)\} \end{aligned} \tag{4}$$

Definition 3 *Let IS become a program of LR, we call CLOSURE(IS) the collection of the closure of IS.*

$$\begin{aligned} CLOSURE(IS)=IS\cup\{&B\rightarrow \bullet \pi \mid A\rightarrow \\ &\beta \bullet B\eta \in CLOSURE(IS), B\rightarrow \pi\} \end{aligned} \tag{5}$$

Definition 4 *Let IS become a program of LR, X belong to $L \cup R$, then we define GO(IS, X) as*

$$GO(IS, X) = CLOSURE(IS(x)) \tag{6}$$

The LR analysis method to build on a modified N-tree workflow model has three steps: (1) building on a state machine; (2) building on goto and action form; (3) building on the N-tree workflow.

5.1. Build up State Machine

The state machine is the first step of LR analysis. It gives people a guidance to know the transference of node at different state, clearly. Only based on the state machine, we can build up the goto and action form.

(1) Build up initializing program collection $ISS=\{CLOSURE(\{Z\rightarrow \cdot S\#\})\}$. We create a new relationship $Z\rightarrow S$. It has the same meaning with the formula in *Cr*, but don't belong to neither *Cr* nor *Clf*. S is root node.

(2) To the program *IS* and $X \in L \cup R \cup \{\#\}$ in *ISS*, let $IS_j=GO(IS_j, X)$. If *j* is not empty and don't belong to *ISS*, then $ISS=ISS \cup IS_j$.

(3) Repeat step 2, until *ISS* is convergent.

(4) To the program IS_i and $X \in L \cup R \cup \{\#\}$ in *ISS*, let $IS_j=Go(IS_j, X)$. If IS_j is not empty then:

$$IS_i \xrightarrow{X} IS_i \quad (7)$$

5.2. Process of building up analysis form of state macnine

The analysis form contains action form and goto form. The action form ensure when we input a($\in$L), then what we should do the next. The goto form make sure when input A($\in$R), then what we should do the next.

(1) If $A \rightarrow a \cdot a\beta$, and $GO(IS_k, a)=IS_i$, $a \in L$, then action$(IS_k, a)=S_i$;

(2) If $A \rightarrow \alpha \cdot \in IS_k$, then to any $a \in L \cup \{\#\}$, let action $(IS_k, a)=R_j$ (We give each formula a number in *R*, *j* is the number of one formula in *R*). The number of $A \rightarrow \alpha$ is *j* in *L*;

(3) If $Z \rightarrow \alpha \cdot \in IS_k$, and *Z* is the root node, then action$(IS_k, \#)=Accept$;

(4) If $GO(IS_k, A)=IS_i$, $A \in R$, then $goto(IS_k, A)=i$.

Use the analysis form to construct a tree automatically

We set *(State Stack, Symbol Stack, Task Stack)* to control the procedure. At the beginning *(State Stack , Symbol Stack, Task Stack)* should be: *(S0, ∅,task)*

(1) If the current *(State Stack , Symbol Stack, Task Stack)* becomes $(S_0S_1 \ldots S_n, X_1X_2 \ldots X_n, a_ia_j \ldots a_k)$, and $action(S_n, a_i)=S_j$, $a_i \in L$, then a_i add into symbol stack and add S_j into state stack. Then the *(State Stack , Symbol Stack, Task Stack)* becomes: $(S_0S_1 \ldots S_nS_j, X_1X_2 \ldots X_n, a_i, a_j \ldots a_k)$

(2) If the current *(State Stack, Symbol Stack, Task Stack)* becomes: $(S_0S_1 \ldots S_n, X_1X_2 \ldots X_n, a_ia_j \ldots a_k)$, and $action(IS_n, a)=R_j$, $a \in L \cup \{\#\}$, then according to the jth formula in R, we assume this formula is $A \rightarrow \alpha$, $k=|\alpha|$ $(\alpha=X_{n-k+1} \ldots X_n)$, then the *(State Stack , Symbol Stack, Task Stack)* becomes: $(S_0S_1 \ldots S_{n-k}, X_1X_2 \ldots X_{n-k}A, a_ia_j \ldots a_k)$, and $S=goto(S_{n-k}, A)$.

After this step, we let the symbols in α point to R_j which is parent node in N-tree model.

(3) If the first state in the state stack is S_i, while the current task is # in the task stack, and $action(S_i, \#)=Accept$, then the analysis finished. The process of building N-tree workflow model has finished.

(4) If the first state in the state stack is S_i, while the current task is a in the task stack, and $action(S_i, a)=Error\ or\ empty$, then the analysis procedure is wrong.

5.3. Example of automatically building procedure

In chapter Error Tackling, we have talked about the process of building up an N-tree workflow model automatically. Now we will give an example to show you the process more clearly.

The N-tree is : (S, (S, R_1, R_2, R_3), (e_1, e_2, e_3, e_4, e_5, e_6), ∅,(S→R_1e_1, $R_1 \rightarrow R_2e_2R_3$, $R_3 \rightarrow e_3e_4$, $R_2 \rightarrow e_5e_6$)) And we mark:

S→R_1e_1 [1],

$R_1 \rightarrow R_2e_2R_3$[2],

$R_3 \rightarrow e_3e_4$[3],

$R_2 \rightarrow e_5e_6$[4],

Where [i] (i=1,...,4) means the number of formula.

The state machine and analysis form of this tree show in following Figure 2 and Table 4.

With the help of state machine and analysis form we can construct the tree. The procedure to construct a tree shows in Table 5.

Table 4. Action and Goto Form

	Action							Goto		
	e_1	e_2	e_3	e_4	e_5	e_6	#	R_1	R_2	R_3
S_0					S_4			1	3	
S_1	S_2									
S_2				Accept						
S_3		S_5								
S_4						S_9				
S_5			S_7							6
S_6				R_2						
S_7				S_8						
S_8				R_3						
S_9				R_4						

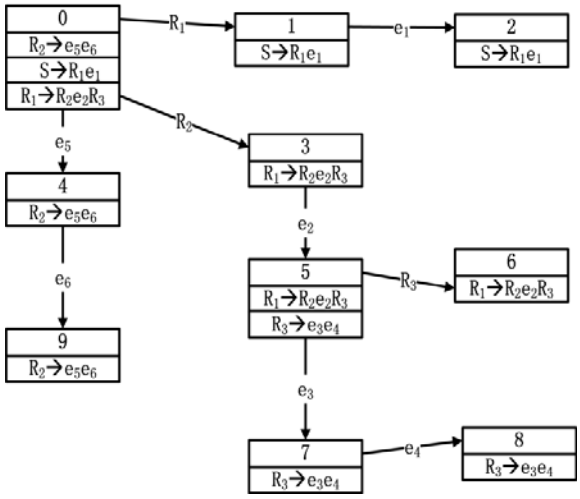

Figure 2. State Machine of Four Processing models

Finally, we can build on a N-tree with the construction form. The result shows in Figure 3.

This example shows, with the steps in section IV, a modified N-tree workflow model can be automatically building. This method can be useful especially when we build on a massive and complex producing workflow.

Table 5. Constructing Procedure

State Stack	Symbol Stack	Task Stack	Goto
0		$E_5e_6e_2e_3e_4e_1$#	4
04	e_5	$e_6e_2e_3e_4e_1$#	9
049	e_5e_6	$e_2e_3e_4e_1$#	3
03	R_2	$e_2e_3e_4e_1$#	5
035	R_2e_2	$e_3e_4e_1$#	7
0357	$R_2e_2e_3$	e_4e_1#	8
03578	$R_2e_2e_3e_4$	e_1#	6
0356	$R_2e_2R_3$	e_1#	1
01	R_1	e_1#	2
012	R_1e_1	e_1#	Accept

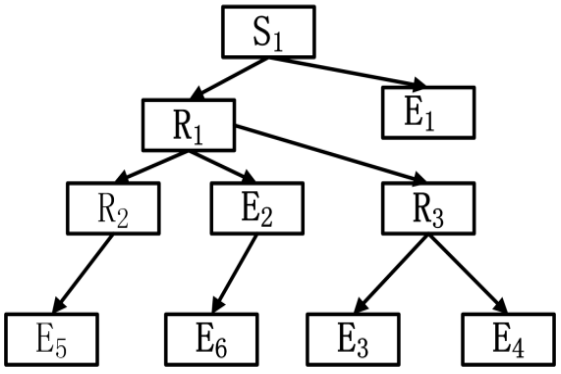

Figure 3. Automatically Built N-tree Workflow

6. Discrepancy between Petri-net, N-tree Model and Modified N-tree Model

Compared with the Petri net and N-tree model, the Modified N-tree workflow has the following prominent features:

(1) The modified N-tree workflow is very simple.

In the N-tree workflow model, the definition of node is very complicated. We show it in Table 6. From this table, we can get that nodes in N-tree model workflow is very complicated. It is not easy to control this whole tree, since any mistake in the execution process may lead to blunders. So in the Modified N-tree model, we simplify the definition of node, for its the main reason leading to errors. Our Modified N-tree workflow model just needs to define the states, processing models and types. These three elements are the most essential and indispensable attributes for any N-tree workflow. With the simplified definition of nodes, people just need to control three variables, and this simplification reduces the probability of mistakes.

(2) A well executed Modified N-tree workflow will have not deadlock, after deleting or adding of tree nodes.

In chapter Merits of modified N-tree workflow, we prove that the Modified N-tree is self-closed, this merit shows that no matter when we change a workflow, before or when its running, we can get a result. And also, in a tree, there is not cycle. It means that a tree has not cycle composed by Parent nodes. Furthermore, any errors can be detected by Error Tackling functions. So I think it's safe to say that in the Modified N-tree workflow, there is not deadlock.

While in the Petri-net, the probability of deadlock is very big, because most Petri-net workflows have one or more cycles. With the increase of the number of cycles, the probability of deadlock is also increasing. Although we can use some special means to find and transact these mistakes, but it will increase the cost. I will give an example of Petri-net workflow with one deadlock in the following.

We show a workflow based on Petri-net to produce commodities in Figure 4. In this workflow, we can see a cycle: G→I→H→J→G. This process can be used to provide necessary materials. When it cannot provide enough materials, this cycle will be deadlock.

Table 6. Attribute of Workflow node

Attribute of node	Meaning	Remark
Sequence	The sequence of execution	The sequence of execution of nodes in the same level
Name	Node name	The name of process of PDM system
Type	Task type	Shown in table 1
is Executing	State of execution	Shown in table 3
Application	Interface of application	Animating the PDM system in the corresponding process management application module
Parent	Parent node	Especially, when a node is root node, its attribute is null
Cycle	Period of executing a node	Supervising the execution time in workflow, and describing work schedule

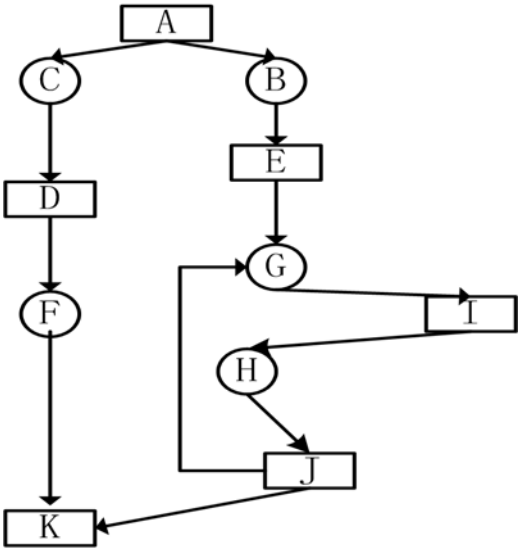

Figure 4. A Example of Petri-net Workflow

Compared with Petri net workflow model, we give a workflow model, to show how we can tackle the same problem by a low-cost function, in Figure 5. In this workflow, State of A, B, C, D, F is, by order: sequence, branch, loop, parallel and sequence. The E node stands for end. The dead lock may happen in the C node when the loop cannot be terminated. In the chapter Error Tackling, we define the Error Tackling function, we set this function in node C, and make sure this process must can be terminated after cycle of 50 times which bigger than the estimated values.

(3) The Modified N-tree workflow is self-closed.

For the well executed Modified N-tree workflow, after we change the node, we can make sure that this workflow can produce a result, as we proved in chapter Merits of modified N-tree workflow. But Grid workflow and Petri net workflow cannot keep the practicability after change.

In the Figure 6, we change the Modified N-tree workflow at Figure 5, deleting node F, and adding a new task node j. Based on contents in chapter Merits of modified N-tree workflow, this tree can be accomplished and lead to a result. Even some errors, such as the condition cannot be met by two of these tasks, happened, we can set an Error Tackling function to solve them by setting a Clock to trigger forced termination and feedback this error.

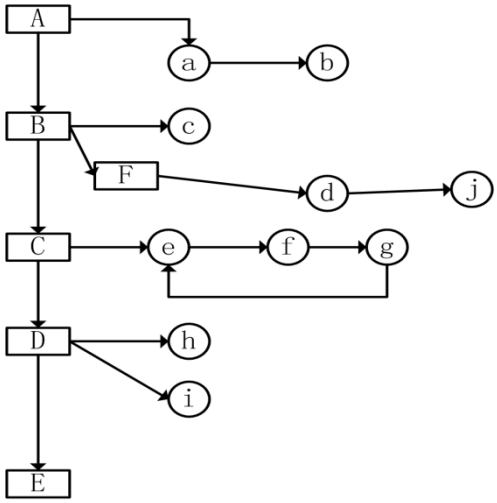

Figure 5. A Example of Modified N-tree Workflow

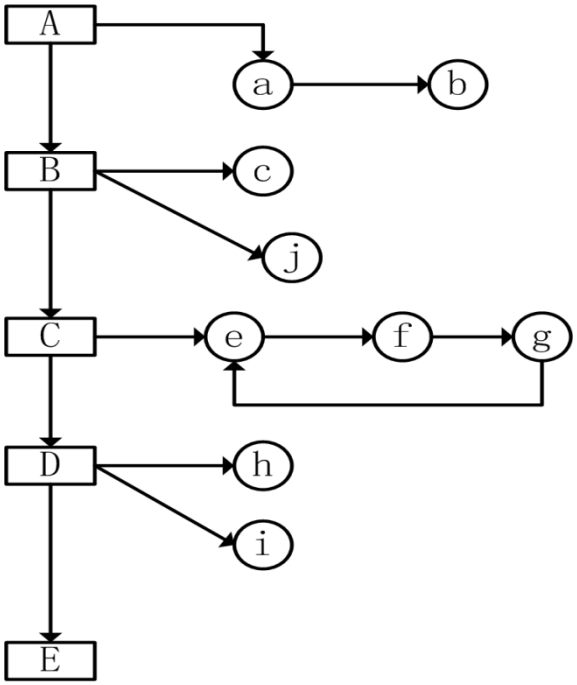

Figure 6. A Example of Modified N-tree Workflow

In Figure 7, we change the Petri net workflow at Figure 4, deleting G, I, H, J. After this change, the Petri net have not efficient mechanism to response to this change.

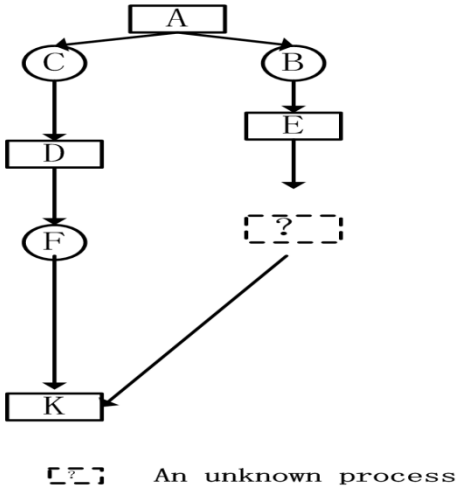

Figure 7. A Example of Grid Workflow

7. Conclusion

In the study, we define a modified of N-tree workflow model and give a detailed process to build on this workflow model. We prove some rules about modified N-tree workflow model. We make sure that workflow tree can work out a result no matter what changes happened. Moreover, we refer the LR(k) analysis method to build on a tree automatically, but there is some limitations to the relationship in *Cr* and *Clf*. If these limitations were violated, the tree will not be built. So, in the future research, we tend to prove that these limitations can be fulfilled, or if not,

we prefer to give more method similar to the above one to construct the N-tree workflow model automatically.

References

[1] Fei He, Yuan-ning Liu, Jing Liu, Ying Chen. Workflow Model Design Based on N-Tree for Process Management in PDM. *Advanced Material Research*. 2011; 268-270: 76-81.

[2] Zeng Qing-hua. *The Dynamic Modification Method of Adapt Workflow Model*. In Proceedings of 2011 3rd IEEE International Conference on Information Management and Engineering. 2011; 06: 640-644.

[3] Myung-Ju Shin, Sin-Wung Kim, Hye-Jin Jeong, Yong-Sun Kim. *Data Modeling of Workflow-XML Resource Model*. In Proceedings of 2012 3rd International Conference on e-Education,e-Business. 2012: 48-52.

[4] Sadiq Shazia W, Orlowska Maria, Saqin Wasim. Pockets of flexibility in workflow specifications. *Information Systems*. 2005; 30: 349-378.

[5] Koci Radek, Mazal Zdenek, Zboril Frantisek, Janousek Vladimir. *Object-oriented design of petri nets modeling tools.* In Proceedings of the 7th International Conference on Intelligent Systems Design and Applications. 2007: 15-20.

[6] Yang Guo-jun, Zheng Ying, Tang Jian, KeShan. The Design and Realization of Workflow Engine based on the Relational Data Model. *International Conference on Computer Application and System Modeling.* 2010; 8: V8408-V8411.

[7] Hong Jiang, Xiang-qian Ding. Modeling of Hierarchical Petri Net-Based Workflow. *International Conference on Control and Industrial Engineering*. 2011; 1: 113-116.

[8] Qiu ZM, Wong YS. Dynamic workflow change in PDM systems. *Computer in Industry*. 2007; 58: 453-463.

[9] Sonmez Ozan, Yigitbasi Nezih, Abrishami Saeid, Losup Alexandru, Epema Dick. *Performance analysis of dynamic workflow scheduling in multicluster grids*. In Proceedings of the 19th ACM International Symposium on High Performance Distributing Computing. 2010: 49-60.

[10] Sun Ping, Jiang Chang-jun, Li Xiang-Mei. Workflow process analysis responding to structural changes. *Journal of System Simulation*. 2008; 20(7): 1856-1863.

[11] Zhen Ling, Cui Shuo, Yue Dong, Zhang Xiaoliang. *A workflow structure verification method based on Warshall algorithm*. In Proceedings of 2011 7th International Conference on Natural Computation. 2011; 4: 1946-1949.

[12] Zhang Liang, Yao Shu-Zhen. Research on workflow patterens based on Petri nets. *Computer Integrated Manufacturing Systems*. 2006; 12(1): 54-58, [jisuanji jichengzhizao xitong]. 2006; 12(1): 54-58.

[13] Guang Deng. A kind of lidar application grid based on eScience's view. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(5): 1147-1150.

[14] Xue Sheng Jun. Scheduling workflow in cloud computing based on hybrid particle swarm algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(7): 1560-1566.