A Method for identifying loops in a Workflow using Petri Nets

V.R. Kavitha[1] , N. Suresh Kumar[2]

[1.] MCA Department, Velammal College of Engineering and Technology, Madurai, India, 625009.
[2] Principal, Velammal College of Engineering and Technology, Madurai, India, 625009
kavi_mani14@yahoo.com

Abstract: In the current scenario most of the organizations adopt Business Process Modeling Techniques to represent their Workflow. The process model for the Workflow involves multiple organizations and multiple departments. Decision making will be done by more than one person; this will lead to increase in Workflow complexity. Proper modeling is needed for efficient business decision making. Workflow complexity depicts the presence of loops of any length. Most of the existing algorithms check the loops which is having the length of 1 or 2. The new algorithm featured in this work which automatically checks the boundedness of Workflow by identifying the presence of loops having any length in the Workflow. The algorithm works in two stages. First, it checks for loop-less Workflow. If the first stage detects the presence of loops, then in the second stage, individual loops are identified and proper remedial measures are taken to modify the original Workflow which ensures boundedness.
[V.R. Kavitha, N. Suresh Kumar. **A Method for identifying loops in a Workflow using Petri Nets.** *Life Sci J* 2013;10(3):339-343] (ISSN:1097-8135). http://www.lifesciencesite.com. 52

Keywords: Boundedness; Petri Nets; Workflow

1. Introduction

The new technological developments help the organizations to automate their complex business processes by making use of Workflow Management techniques. The main objectives of the organizations are to analyze the process, categories them and sequence them based on the nature of work. This process may be fully automatic, semiautomatic or manual. Workflow Management System takes care of assigning jobs to the resources, keeping track of the status of individual jobs and execution of activities [1, 2, 3 and 4].

The term Workflow Management refers to the ideas, methods, techniques and softwares used to support structured business processes. Staffware, IBM MQseries, COSA are Workflow Managements systems that offer generic modeling capabilities. Workflows are case based. A job is divided into set of cases and each case is executed according to the specific order. The goals of Workflow Management are scheduling and executing the cases as effectively as possible. Basically workflow model is considered as a graph; also referred as Flow Diagram, Workflow Graph, Procedure and Routing Definition. When the business processes are represented in the form of Workflow, the designer designs the model accurately, and then tests for correctness. Designer should have in-depth knowledge in Workflow languages. He should conduct number of meetings with the managers for clarification of work sequences[5,6]. To avoid such meetings and to automate and analyze workflow several new tools are developed by the researchers.

Petri net is selected for representing workflow models because checking the boundedness property is easy. As the flow of execution is defined, it is easy to transform Petri Net into digraph. In literature process logs are used to identify short loops of length two; but higher lengths in a workflow are identified by making use of the proposed algorithm. The complexity of the workflow increases if the size of the workflow increases. If it is exponential, then complexity of identifying loops present in the net is also exponential. Section 2 presents preliminary definitions and theories needed for understanding the current paper. Section 3 presents the proposed algorithm. Section 4 given the implementation part of the algorithm with illustrations. Section 5 concludes the paper.

2. Preliminaries

2.1 Petri Net

An *Petri net*, ' N ' is a bipartite, weighted, directed multigraph, mathematically represented by a four-tuple $N = (P,T, I,O)$ where

$P = \{ p_1, p_2 ,..., p_i ,..., p_n \}$ is a finite set of places,
$T = \{ t_1,t_2 ,...,t_j ,...,t_m \}$ is a finite set of transitions,
$P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$
$I : (P \times T) \rightarrow N_0^+$
$O : (P \times T) \rightarrow N_0^+$ where $N_o = \{0,1,2,\ldots\}$

A Petri net structure N = (P,T,F,W) without any specific initial marking is denoted by N. A Petri net with the given initial marking is denoted by (N,M_0) [5,6 and 7].

2.2 Bounded

A Petri net (N,M_0) is said to be k-bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0, i.e., $M(p) \leq k$ for every place p and every marking [8,9 and 10] $M \in R(M_0)$

2.3 Firing rule

A transaction is said to be enabled when all its input places should be filled with at least one token. An enabled transition may or may not fire depending on whether the event actually takes place or not. But once enabled, a transition has the potential to fire; hence, the transition is called potentially friable. The firing of a Petri net will remove tokens from its input place and place the tokens in all its output places. If there is no transaction is enabled then the execution halts. A transaction $t \in T$ is enabled iff $M(p) \geq I(p,t); \forall\ p \in P$. If an enabled transaction t fires then it causes a change in marking from M(p)toM′(p)given by the equation:
$M'(p) = M(p) - I(p,t) + O(p,t)\ ; \forall\ p \in P.$

The firing rule is called Strict Firing Rule, when it is applied to finite capacity Petri nets. And it is called Weak Firing Rule when it is applied to infinite capacity Petri nets. At any point of time the output place transaction will not exceed the maximum token carrying capacity C (p).

2.4 Graph

A graph is a pair of sets (V,E) where V is a finite set called the set of vertices and E is a set of 2-element subsets of V , called the set of edges.

2.5 Path and Cycle

A path in a graph G = (V,E) is a walk of length k in which the vertices $v_1, \ldots, v_{k+1}$ are all distinct. Here $v_i, v_j \in V$ and $(v_i, v_j) \in E$. A cycle in a graph G = (V,E) is a walk of length k _ 1 in which $v_1, \ldots, v_k$ are distinct and $v_1 = v_{k+1}$.

2.6 Digraph

A digraph D = (V, E) can be interpreted as a relation R on the set V in the following way. For every x, y ∈ V , we write xRy if and only if (x, y) ∈ E.

3. Proposed Algorithm To Check For Boundedness

Workflow complexity depicts the presence of loops of any length. Most of the existing algorithms check the loops which are having the length of 1 or 2. The proposed algorithm which automatically checks the boundedness of Workflow by identifying the presence of loops having any length. Since Petri net gives flow direction of the processes, it is easy for the designer to convert it into a directed graph. This property helps in finding the loops present in the net.

The algorithm works in two stages. Here P_i represents different stages of the algorithm where i = 1 or 2. First, it checks for loop-less Workflow. If the first stage detects the presence of loops, then in the second stage, individual loops are identified and proper remedial measures are taken to modify the original Workflow which ensures boundedness. During the First stage the Workflow is converted into Petri net. Let $I(P_i)$: Input nodes for the i^{th} phase of the algorithm $O(P_i)$: Output nodes for the i^{th} phase of the algorithm. In phase one $I(P_1) = n$ and check the $O(P_1)$ which is n_1, if $n_1 = n$ then the number of input nodes and the output nodes are equal. The algorithm concludes that no loop present in the Workflow and the Workflow is bounded. If $n_1 < n$ then phase one of the algorithm identifies the presence of loop in the Workflow and the output is routed to phase two.

$I(P_2) = n\text{-}n_1$ where n_1 is the number of nodes identified which will not form loop in the Workflow. The second phase of the algorithm uses depth first algorithm for traversal and uses stack for storing multiple paths.

3.1 Pseudo code to check for boundedness

Findloop(G(V,E))

1. Phase 1 ← V, E
2. Zero-in-degree(V)
3. for all V ∈ G do begin
   if (in-degree(V_i) ==0) then
   delete V_i from G and update in-degree for all the nodes adjacent to V
   end
4. Phase 2 → for all V ∈ G do begin
   traverse(G)
   end
5. traverse(G) begin
   for all V ∈ G if outdegree >1 then create stack and store traversal
6. if any repetition of node then
   L ← vertices from stack
7. Make changes made in step 6 leads to nodes having in-degree zero then go to Phase 1 and repeat the steps
8. Algorithm terminates when V = Φ

4. Algorithm Implementation with Illustrations

To illustrate the algorithm, Five year plan Workflow of an organization is taken as input, which is given in Figure 1. Using Figure 1. A Petri net model is derived and is given in Figure 2. The Petri net given in Figure 2 is converted into a digraph and the output is shown in Figure 3.

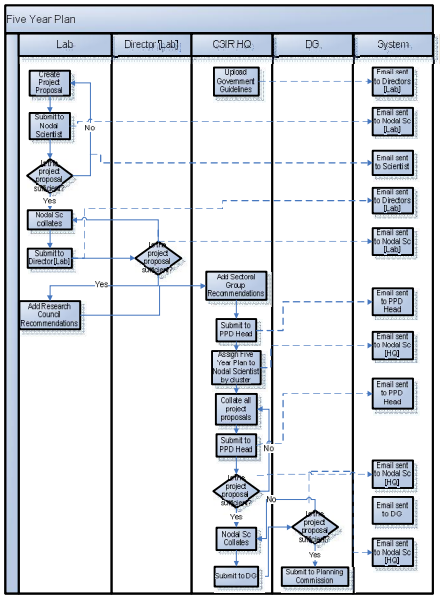


Figure 1: Workflow of Five year plan

a). In the first iteration zero in-degree nodes removed are P_{30}, P_{31}, P_{22}, P_{17} and in-degree information is updated this leads to further removal of nodes in the second iteration.

b). Remove zero in-degree nodes P_3, P_{29}, P_{18}, P_{23} the remaining graph is given in Figure 4. From the Figure. 4 it is identified that the algorithm cannot proceed further because there is no nodes having in-degree zero and number of vertices is not equal to the number of nodes visited. So, first stage is giving the output as loop present in the network, and it may lead to unboundedness.

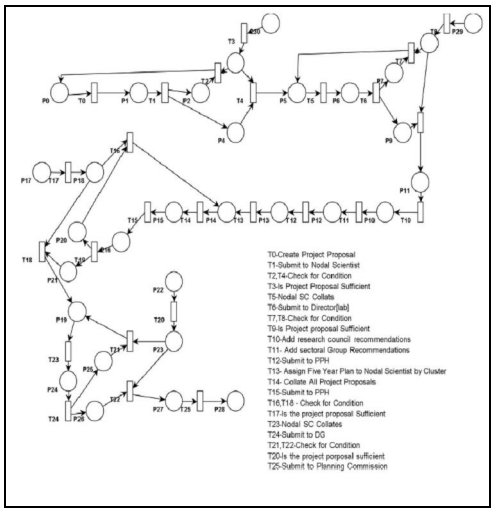


Figure 2: Five year plan Workflow represented as Petri Net

c) Depth first traversal starts from the node P_0 and when it is encountering multiple paths further traversal information is stored in a stack and algorithm proceeds and generates sub graphs having loops. These sub graphs are analyzed further and the nodes involved in the loop are identified. In our example the identified loops are:

$L_1 = P_0, P_1, P_2, P_0$;
$L_2 = P_5, P_6, P_7, P_0$;
$L_3 = P_{19}, P_{24}, P_{25}, P_{19}$;
$L_4 = P_{16}, P_{15}, P_{14}, P_{13}, P_{20}$;

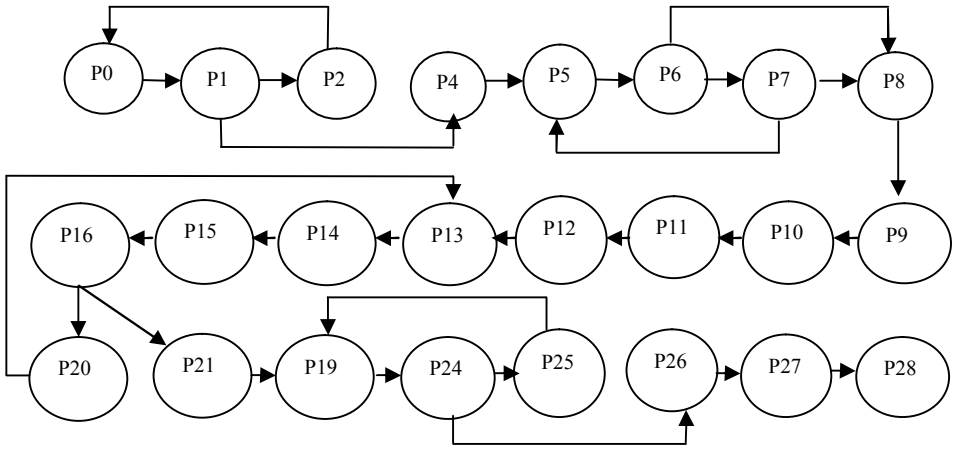


Figure 3: Graph representation of the Petri net given in Figure 2

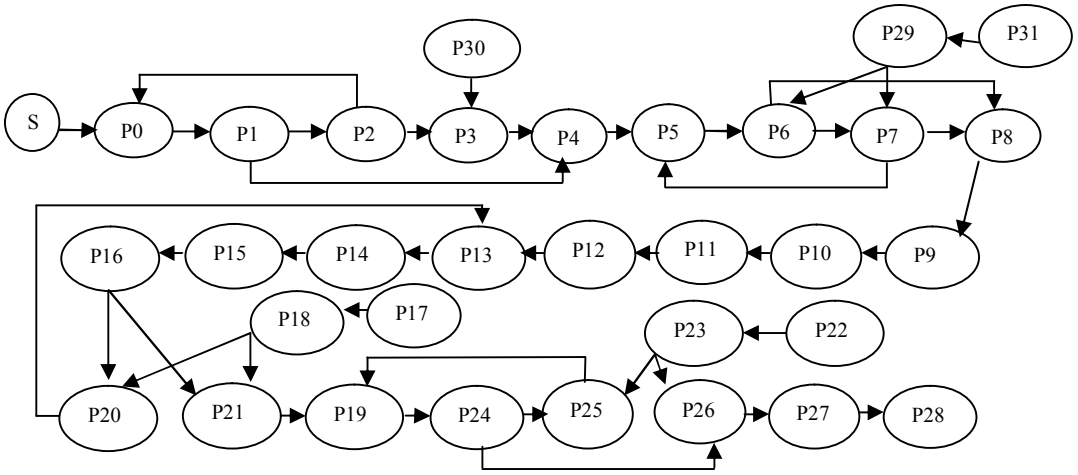
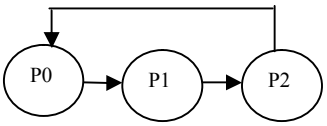


Figure. 4: Resultant graph after first phase

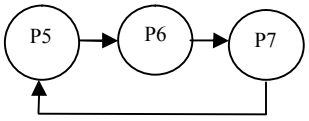
d) Identified loops are given in Figure 5.

From Figure. 5 it is clear that using the loops generated by the proposed algorithm, The unboundedness identified are four places. To make the unbounded net as bounded modification is required. This can be decided based on the current situation and properties of the Workflow. The restriction here identified are n=3 and dummy node is attached which will not affect the functioning of the Workflow at any point of time. Only when unboundedness occurs it will control the Workflow.

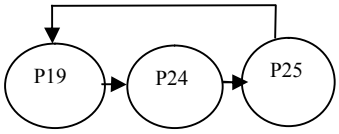
The modification of the Workflow is given in Figure 6. By doing this modification the unbounded Workflow is modified as a bounded Workflow.



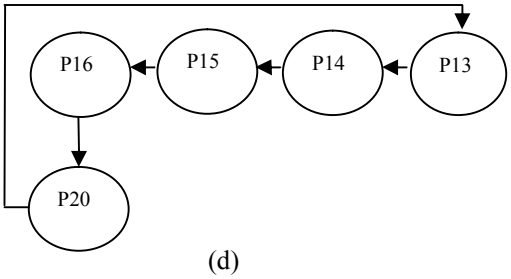
(a)



(b)

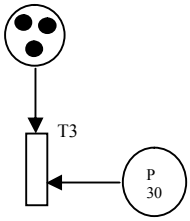


(c)

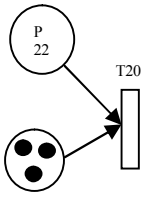


(d)

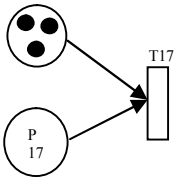
Figure 5. (a,b,c,d) Loops identified after Algorithm is applied on Figure 4.



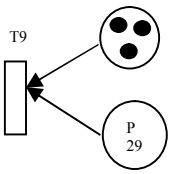
(a)



(b)



(c)



(d)

Figure 6: (a,b,c,d) Decision taken after the loops are identified

5. Conclusion

Current research work proposes an algorithm to find the loops of higher lengths that are present in a Workflow. The main advantage of applying the proposed algorithm is practically possible to apply even if the workflow grows in size and previous research works did not identify the loops which are having higher lengths. The algorithm expects the vertices are unique and no repetition is allowed. There is a need to further address these challenges and duplicate tasks with noise needs more attention.

Corresponding Author:
Mrs.V.R. Kavitha
MCA Department
Velammal College of Engineering and Technology
Madurai, 625009, India
E-mail: kavi_mani14@yahoo.com

References

1. Ana Karla Alves de Medeiros, Antonella Guzzo, Gianluigi Greco, Wil M. P. Van der Aalst, A.J.M.M. Weijiters, Boudewijn F. van Dongen, and Domenico Sacca. Process Mining Based on Clustering : A Quest for Precision. BPM Workshops, LNCS 4928, Springer-Verlag Berlin Heidelberg 2008.
2. W.M.P. van der Aalst and K.M. van Hee. Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA, 2002.
3. Yu Ru and Christoforos N. Hadjicostis. Reachability Analysis for a Class of Petri Nets. Joint IEEE Conference on Decision and Control and 28th Chinese Control Conference P.R., China, 2009.
4. van der Aalst, W., Rubin, V., van Dongen, B., Kindler, E., G¨unther, C. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPM Center, BPMcenter.org, 2006.
5. Scott Callaghan, Ewa Deelman, Dan Guntere, Gideon Juve, Philip Maechling, Christopher Brooks, Karan Vahi, Kevin Milner, Robert Graves, Edward Field, David Okaya, Thomas Jordan. Scaling up workflow-based applications. Journal of Computer and System Sciences, 76 (2010) :428–446.
6. Gergely Sipos. Protecting the consistency of workflow applications in collaborative development environments. Future Generation Computer Systems 28 (2012):500–512.
7. Wanchun Dou, J. Leon Zhao, Shaokun Fan. A collaborative scheduling approach for service-driven scientific workflow execution. Journal of Computer and System Sciences 76 (2010):416–427.
8. Lavanya Ramakrishnan, Jeffrey S. Chase, Dennis Gannon, Daniel Nurmi, Rich Wolski. Deadline-sensitive workflow orchestration without explicit resource control. J. Parallel Distrib. Comput. 71 (2011):343–353.
9. Haiping Zha, Jianmin Wang, Lijie Wen, Chaokun Wang, Jiaguang Sun. A workflow net similarity measure based on transition adjacency relations. Computers in Industry 61 (2010):463–471.
10. Erik Elmroth, Francisco Hernández, Johan Tordsson. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Generation Computer Systems, 26 (2010) :245_256.

6/26/2013