IJARIIT INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 8, Issue 1 - V8I1-1182)
Available online at: https://www.ijariit.com

Deadlock Detection in Concurrency System using Timed Petri Net

Ahmed Elajeli Rgibi
ahmed.rgibi@sabu.edu.ly
Sabratha University, Sabratah, Libya

Amany Khalifa Alarbish
Amany.alarbish@sabu.edu.ly
Sabratha University, Sabratah, Libya

Amal Apojila Oshah
Amal.adoude@sabu.edu.ly
Sabratha University, Sabratah, Libya

ABSTRACT

Deadlock is a state in which each user of a group of users on networks(distributed system) waits for another user, including itself to take action, such as sending a message or using network resources commonly releasing a lock, this lock is common problem facing users in network systems(NWSs), it is named deadlock problem. This problem is occurring in distributed systems were software and hardware sharing resources through network devices. The deadlock problem has received. To solve deadlock problem the important step is deadlock detection which helps us to void deadlock problem.

Keywords: *Deadlock, Timed Petri nets, NWSs, Distributed system, Concurrency system.*

1. INTRODUCTION

Recently, Deadlock issue has a great attention due to its effect in concurrent systems development. Consequently, more important business benefits have been achieved. The simple definition of deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for other resources acquired by some other process [1]. Consider an example when two cars are coming toward each other on same lane and there is only one lane, none of the cars can move once they are in front of each other. Similar situation occurs in concurrent systems when there are two or more processes hold some resources and wait for resources held by other(s) as fig-1.

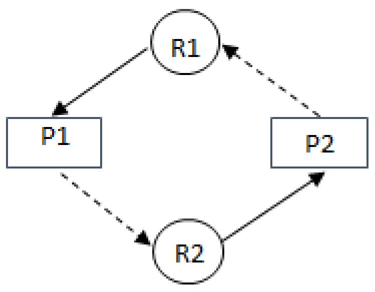

Fig-1: Process waiting resources

A network system is a system consisting of a set of interconnected computers or other devices as printers, scanners for sharing information and applications. The network systems have characteristics of high degree of distributed system and multi computers to serve users and sharing software and hardware resources between users and devices. Nowadays, information technology is extensively applied to contemporary network systems [2]. And a distributed system is uniform, it can be decomposed to server processes communicating by messages or to agent processes communicating by shared resources (servers' states). Finally, Timed Petri Nets have proven to be a very powerful tool for the modeling and analysis of distributed systems. To date several approaches have been defined that use Timed Petri Nets to model a system being analyzed.

2. UNTIMED OBJECT-ORIENTED PETRI NET FORMAL DEFINITION AND STRUCTURE

The formal definition and structure of untimed OOPN net details are described in [3] and is summarized as follows:

2.1. OOPN system

SYSTEM = (*O*, *R*), where
O: a set of objects
R: a set of interconnection relations

2.2. External structure of untimed OOPN object

An object Oi $\in$O is externally represented by the six-tuple;
Oi = (Hi, IGi, OGi,IMi, OMi, Fi) where
Hi : an object hierarchy
IGi: a set of input gates of object Oi
OGi: a set of output gates of object Oi
IMi: a set of input message queues of object Oi
OMi: a set of output message queues of object Oi
Fi: a set of flow relations of object Oi

2.3. Internal structure of a composite object

The internal structure of composite object COi, ICOi, is described by the following three tuple:
ICOi= (X, Y, Ri), where
X: $X= \{X|X\in\rho(CO), COj \notin X\}$
Y: $Y= \{Y|Y\in\rho(PO)\}$
Ri: a set of interconnection relations. X is an element of power set of CO i.e. ρ (CO), excluding sets containing COj itself, and Y is an element of the power set of PO. Interconnection relations Rj between objects is represented in terms of gates and their input and output flow relations.

2.4. Internal structure of a primitive object

An internal structure of a primitive object POi can be defined as follows:
IPOi = (Di, SVi, Si, ATi, LFi, INi, M0)
where
Di: a set of attributes of POi
SVi: a set of state variants of POi
Si: a set of states of POi
ATi: a set of action transitions of POi
LFi: a set of local flow relations of POi
INi: a set of instances of POi
M0: initial marking of POi

3. DEADLOCK DETECTION IN TIMED OOPN NETS

Deadlock in a timed OOPN net is a state of an object in which no further action is possible. It is a state an object Petri net attains when there is no intercommunication links between supposedly interacting objects. Deadlock detection, therefore, in a timed compound OOPN net entails detecting for lack of synchronized intercommunication between communicating objects. Intercommunication in OOPN nets involves sending and receiving of messages modeled tokens among communicating objects through message queues. Thus, deadlock detection analysis in OOPN net system can be reduced to analyzing for deadlock on the objects observable token flow path reachability tree. However, constructing a reachability tree directly for a timed compound OOPN net directly is not possible owing to design constraints imposed on the system. To overcome these limitations, a simple observable token path equivalent Petri net is abstracted from the interacting object system.

4. PROPOSITIONS AND DEFINITIONS TO DETECT DEADLOCK

The following propositions and definitions suggest a straight forward way to detect deadlock in OOPN nets. The net is transformed into a token firing path equivalent net and its reachability graph is computed and analyzed for deadlock as in ordinary Petri nets.
Proposition 1: A dead marking in a reachability graph indicates the existence of a deadlock.
Proposition 2: A timed token firing path net is equivalent to its corresponding timed OOPN net.
Proposition 3: A timed compound OOPN net *O* is deadlock free if its token firing path equivalent net is deadlock free.
Proposition 4: The reachability graph of token path equivalent Petri net is finite.

5. TOKEN FLOW PATHS IN OOPN OBJECTS

A token flow path is a path, observable at compound net level, a token takes through cooperating object nets. It describes an overall connectivity channel of cooperating object nets. Token flow paths in OOPN nets are classified into two types;

5.1. External flow paths

It representing a token flow path between two composite objects input and output message queues described by a simple timed Petri net set of message queues and gates.

5.2. Internal flow paths

It representing a token flow path from an objects internal input and output message queue calling pairs described by a simple timed Petri net set of action transitions and simple states.

6. TOKEN FLOW PATH EQUIVALENT NET DEFINITION

Let object Oi and Oj be a set of communicating objects where Oi, Oj $\in$ O. The token flow path equivalent nets for these objects are obtained as follows [4]:

6.1 A finite Petri net $M_1 = (P_1, T_1, IM, OM, F, FD)$ is an internal timed token flow path of a primitive object O_i iff

a. FD is the fixed delay time associated with transitions and gates.
b. The set of places is divided into three disjoint sets: internal places $p_i \in P_1$, input message queues $imq_i \in IM$ and output message queues $omq_i \in OM$.
c. A set of transitions $t_i \in T_1$
d. The flow relation is divided into internal flow $F_1 \subseteq (p_i \text{ X } t_i) \cup (t_i \text{ X } p_i)$ and communication flow $F^{C1} \subseteq (imq_i \text{ X } t_i) \cup (t_i \text{ X } omq_i)$
e. The net $N(M_1) = (p_i, t_i, F_1, FD)$ is an internal timed token flow path equivalent net for object O_i. Similarly, the internal token flow path equivalent net for O_j is obtained to be $N(M_2) = (p_j, t_j, F_2, FD)$.

6.2 A finite Petri net $M_3 = (IM, OM, IG, OG, F, FD)$ is an external token flow path of two token exchanging objects $O_i, O_j \in O$ iff:

a. The set of gates are divided into two parts: input gates $IG_j \in IG$ and output gates $OG_i \in OG$
b. The set of message queues are divided into two: input message queues $O_i.imq_i \in IM$ and $O_j.imq_j \in IM$ and output message queues $omq_i \in OM$ and $O_j.omq_j \in OM$.
c. $O_i.OG_i \cap O_j.IG_j \neq \phi$ where $O_i.OG_i = g_k \in OG_i$, $O_j.IG_j = g_k \in IG_j$ and $g_k \in O_i.OG_i \cap O_j.IG_j$
d. $O_i.OM_i = \{om_{i\}} \neq \phi\}$ and $O_j.IM_i = \{ im_i \neq \phi \}$
e. $om_i \in \bullet g_k$ and $im_i \in g_k\bullet$
f. The communication flow relation $F_3 \subseteq (O_i.imq_i \text{ X } g_k) \cup (g_k \text{ X } O_j.imq_j)$ and vice versa if the token sending object is from O_j.
g. The net $N(M_3) = (O_i.imq_i, O_j.imq_j, g_k, F_3, FD)$ is an external timed token flow path equivalent net for the message exchanging objects object $O_i, O_j \in O$.

7. REACHABILITY TREES

A reachability tree specifies reachable markings of a Petri net. A reachability tree is constructed to generate a Petri net reachability sets from the initial marking. It presents reachability of states in a given object Petri net by specifying the paths for its reachability through a path of transition firing sequences. Every path in the tree, starting at the root, corresponds to a legal firing transition sequence. A reachability tree is essentially a state graph and contains a place corresponding to every reachable marking and optimization analysis tool for Petri nets that maintains the behavior of the Petri net [5].

8. OOPN DEADLOCK DETECTION ALGORITHMS

Analyzing a large complex net in a single step often produces erroneous results and is computationally inefficient. This paper introduces a two-step deadlock detection algorithm that validates each object in a first step and then constructs a reduced token path equivalent net in the second step. In Petri net reduction stage [6], a local analysis is first performed to validate the internal behavior of each primitive object and a simple timed token flow path equivalent net is constructed to replace the primitive net being reduced by applying appropriate reduction rules. After the primitive objects have been reduced, a composite objects token flow path equivalent net is constructed from the underlying primitive object nets token flow path equivalent nets. The reduced composite net is then analyzed for deadlock by performing a reachability analysis as in ordinary Petri nets.

10. OOPN NET REDUCTION ALGORITHM

The algorithm for reducing a timed OOPN net is summarized as follows:

1. If the object net being analyzed is a primitive object net
a. Perform a reachability analysis for the internal behavior of each timed primitive object to obtain its transition firing sequence starting from the initial state.
b. Identify all token flow paths in the object by finding the sequence of token input message queues and output message queues connected to the action transitions firing sequence obtained in 1(a). Select an appropriate rule to get token flow paths for the object.
c. For each token flow paths obtained in (b), get transitions with their corresponding timings in their path. Fuse the transitions and set its time to a new transition connecting the pre-path and post-path parts of the token flow paths.
d. Construct a reduced token flow path equivalent net from the token flow paths obtained in (c).
2. Else if it is a compound object net
a. For all the primitive object nets, do a model deadlock check on the structure for safeness and non-concurrency; if there is no deadlock in the primitive object net then reduce the object as in (1)
b. Construct a compound reduced net by merging the primitive objects reduced nets obtained in 2(a).

11. OOPN DEADLOCK DETECTION ALGORITHM

The algorithm for detecting deadlock in OOPN nets is summarized below:

c. If the object being analyzed is a primitive object net, perform a model check for presence of tokens and presence of concurrency.
d. Else if it is a compound object net, for all its primitive object nets get their reduced token flow path equivalent net.
e. Construct a merged net from the primitive nets reduced token flow paths obtained in (b).
f. Simplify the merged nets in (c).
g. Construct a reachability tree for the merged net obtained in (d) and detect for deadlock.

12. EXAMPLE

The vending machine dispenses two kinds of snack bars, 20c and 15c. Only two types of coins can be used, 10c coins and 5c coins. Note the machine does not return any change Figure-2a describe vending machine and Figure-2b describe equivalent vending machine (petri net) and Figure 2c describe reachability tree for vending machine [7]. There are three scenarios are expected:
Scenario 1: – Deposit 5c, deposit 5c, deposit 5c, deposit 5c, take 20c snack bar.
Scenario 2: – Deposit 10c, deposit 5c, take 15c snack bar.
Scenario 3: – Deposit 5c, deposit 10c, deposit 5c, take 20c snack bar.

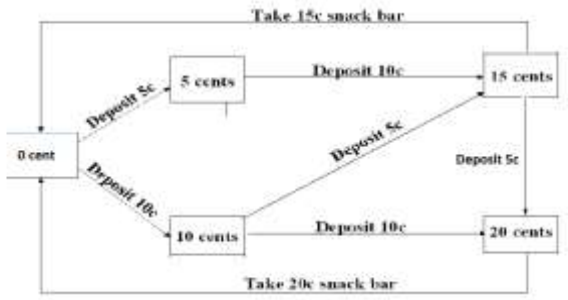

Fig-2a: vending machine diagram

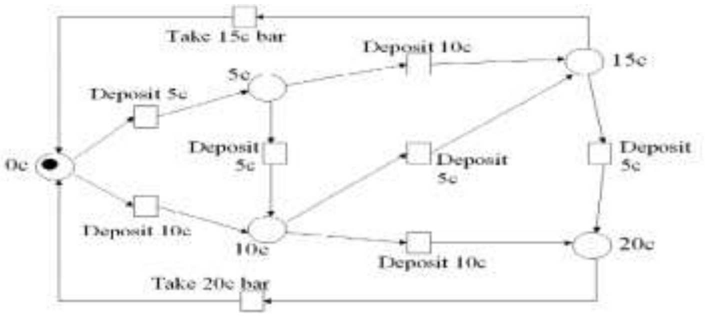

fig-2b: vending machine petri net

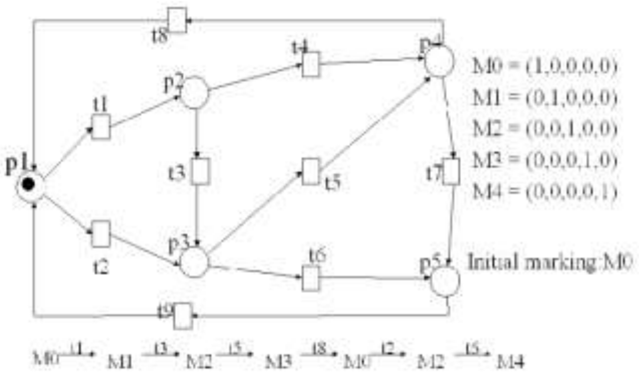

fig-2c: vending machine reachability tree

A Petri net with initial marking M_0 is live if, no matter what marking has been reached from M_0, it is possible to ultimately fire any transition by progressing through some further firing sequence. A live Petri net guarantees deadlock-free operation, no matter what firing sequence is chosen.

13. CONCLUSIONS

This paper has introduced a deadlock detection analysis approach in OOPN net models that uses structural model checking, a token flow path equivalent net, model reduction and reachability trees to analyze for deadlock and extended the results to analyze for deadlock in OOPN net models. The reduction rules used to reduce Petri net with varying timed OOPN structural designs and a reachability tree used for maintains the behavior of the system, finally introduced OOPN net reduction algorithm and OOPN Deadlock Detection Algorithms that used to deadlock detection.

14. REFERENCES

[1] Sumika Jain. An Overview on Deadlock Resolution Techniques. International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 Published by, www.ijert.org NCRIETS – 2019 Conference Proceedings.
[2] Kounev, S. (2006). Performance modeling and evaluation of distributed component-based systems using queueing petri nets. IEEE Transactions on Software Engineering, 32(7), 486-502.
[3] Ahmed ajeli rgibi, Dataflow Errors Detection in BPM, ICEIE 2011 conference, Tianjin – China, September 9 – 11, 2011.
[4] Guanjun Liu. Complexity of the deadlock problem for Petri nets modeling resource allocation systems. Elsevier. m3Gsc; December 1, 2015;7:41.
[5] ShouGuang Wang, MengChu Zhou, MengDi Gan, Dan You, and Yue Li. New Reachability Trees for Unbounded Petri Nets. 2015 IEEE International Conference on Robotics and Automation (ICRA) Washington State Convention Center Seattle, Washington, May 26-30, 2015.
[6] B. V. Dongen, W. Aalst, H. Verbeek, Verification of EPCs: Using Reduction Rules and Petri Nets. Published in CAISE 17 June 2005. DOI:10.1007/11431855_26 - Corpus ID: 1586896.
[7] Distributed Model: Petri Nets. Introduced by Carl Adam Petri in 1962. A diagrammatic tool to model concurrency and synchronization in distributed systems.