MODELING WORKFLOW MANAGEMENT IN A DISTRIBUTED COMPUTING SYSTEM USING PETRI NETS

Bakpo, F.S
Department of Computer Science
University of Nigeria, Nsukka
E-mail: fbakpo@yahoo.com

ABSTRACT
Distributed computing is becoming increasingly important in our daily life. This is because it enables the people who use it to share information more rapidly and increases their productivity. A major characteristic feature or distributed computing is the explicit representation of process logic within a communication system, which allows for computerized support. This paper presents a model of distributed computing system and applies the concepts of Petri nets in modelling workflow in this domain using e- purchase as a case study. Petri nets are an established tool for modelling and analyzing processes.
Keywords: Distributed computing system; Petri nets;Workflow management

1. INTRODUCTION

It is a fact of life that various organisations and individuals are becoming increasingly dependent on distributed computing systems. According to V. Glushkov, a well-known Soviet Scientist, "the development of computer networks and terminals results in a situation where the ever greater part of information, first and foremost, that in the scientific- and-technological, economic and socio-political fields, is transferred to computer memory[1]". Distributed computing systems (DCS) enable computers and the people who use them to share information more rapidly and to increase their productivity. Even if a Local Area Network (LAN) is limited to carrying e-mail between desks in the same office, it can save thousands of steps (and realms of paper) every year[2]. Connecting several Personal Computers (PCs) to a shared printer saves money and space and makes it possible to provide faster, higher quality printing to every user. Sharing files over a LAN enables everyone to literally read off the same page and drastically reduces errors and time-wasting duplication of efforts. Instant access to shared databases and applications can also provide powerful productivity tools. The main purpose of a DCS is the support of the definition, execution and control of' processes. Because processes are a dominant factor in any DCS, it is important to use an established framework for modelling and analysis of distributed computing processes. In this paper, Petri net -a well-founded process modelling technique is used. Carl Adam Petri invented Petri nets theory in the sixties[8]. Since then Petri nets and their extension in colour, time etc, have been applied successfully to model and analyze all kinds of processes with applications ranging from communication protocols to performance study of complex systems [1.3.7]

2. What a Distributed Computing System Is

The term distributed computing system (DCS) refers to a collection of autonomous computer systems capable of communication and cooperation via their hardware and software interconnections [3] There are also people who use the term computer network, in place of DCS. The general characteristics are:

(i) The absence of' shared memory;
(ii) Unpredictable inter-node communication delays; and
(iii) Practically no global system state observable by component machines.

Due to the lack of shared memory, inter-site communication and synchronization is usually carried out by means of inter-node messages. As a consequence of' the communication delays incurred in assembling status messages and of dynamic component state changes, including potential link and site failure, it is almost impossible for a given node to assess the global state of' a DCS at a given point in time. A distributed operating system usually governs the operation of' a DCS and provides a virtual-machine system abstraction to its

users. Virtual machine abstraction (VMA) means that the actual network component and resource distribution is hidden from the users and application programs, unless they explicitly demand otherwise. The key objective of a DCS is transparency. The major potential benefits of DCS include:

(i) Resource sharing and load balancing;
(ii) Communication and information sharing;
(iii) Incremental growth;
(iv) Reliability, availability, fault tolerance;
(v) Performance,

Depending on the physical distance spanned by the communication sub net works. DCS are usually classified as:

(a) Wide-area networks (WANs)
(b) Local-area networks (LANs)

While no specific delimiting distance is defined, it is reasonable to think of a LAN within the confines of a single building or a small campus. WANs on the other hand, may connect hosts that are many miles apart, including even intercontinental distances. Mostly for technological reasons, WANs tend to have comparatively low bandwidth and high communication delays. LANs, on the other hand, are often characterized by high bandwidth and low communication delays. These fundamental differences have a significant impact on the design choices and selection of the distributed algorithms whose characteristics may make them more suitable for one type of the network than the other.

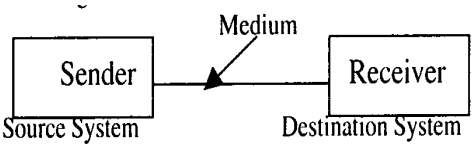

Fig 1: A generic structure of distributed computing system

3. Model of Distribution Computing Systems

Any communication set-up can be represented as shown in figure 1.

It consists of three components: the source system, the medium and the destination system. Each source and destination system is capable of receiving and transmitting signals. The medium is a channel or path for sending a message between two communicating systems. Thus, a distributed computing system comprises a set of processes, which communicate with each other exclusively by sending and receiving messages. Each process can also do some local computation and access its local state (but not the state of any other process). Communication between the processes is used to decrease the uncertainty each process has about the states of the distributed computation; that is, the processes come to **know** facts about the system computation as that computation evolves.

A distributed computing system thus consists of two types of elements[6]: (i) Processes which execute events (let *P* represent the set of n processes in the system); and

(ii) A communication system, **N**, which contains a set of message packets of the form (**p, m, q**), representing a message m from process **p** to process **q**.

Each process in a DCS is characterized by a set of process executions, each of which is a finite sequence of abstract events. These events, the elementary building blocks of the processes which make up distributed computations, include: **local** (the executing process performs an unspecified internal action with no external communication); **send** (**m, p**), (the executing process sends message **m** to process **p**); and r**ecv** (**m, p**) (the executing process receives message **m** from process **p**). We define the domain **D** for process event sequences in our distributed model as the set of events a process may execute, Currently, for each process,

D = {**local**}∪(**send** (m, P) / m ∈ M and p∈ P) ∪{**recv**(m, p) / m∈ M and p∈P}; other events will be added to the domain as required.

Given a protocol or algorithm for a process set **P**, the DCS prescribed by that protocol is modelled by the set, ε, of all possible executions over **P**. Each member of ε captures a process execution for each process in **P** and the behaviour of the communication system.

4. Representing Workflow managements in DCS using Petrt nets

4. I Petri net concepts

The concept of Petri nets has its origin in Carl Adam Petri's dissertation **Kommunikation mit Automaton**, submitted in 1962 to the Faculty of Mathematics and Physics at the Technische University Darrnstadt, German [8], Since then the use and study of Petri nets have increased considerably.

A Petri net has been variously defined in [1.6.10] as a quadruple N= (P, T, F, W) where: P is the set of places (graphically represented as circles) with /p/ = n and p = n and p ≠0. T is the set of transitions (graphically represented as bars\) with /T/= m and T≠0, T∩ P =0. F ⊆ (P x T) ∪ (T x P) is the flow relation of N W: F→N \ {0} attaches a weight to each arc of the net. Figure 1 shows an example of a Petri net.

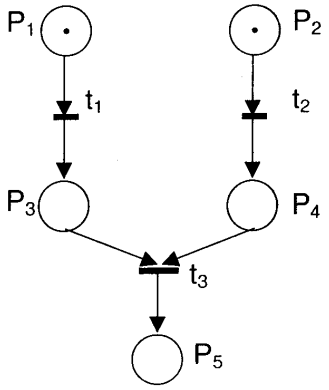

Fig.1: A Petri net diagram.

It consists of places (circle), transitions (bar) and directed arcs (flow relations) that connect them. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. A place may contain zero or more markings also called weights or tokens. The current state of a modelled system (the marking) is given by the number (and type) of tokens in each place. Transitions are active components and are used in modelling activities
(events), which can occur (the transition fires), thus changing the state of the system (the marking of the Petri net). Transitions may fire only if they are enabled, which means that all the preconditions for the activity (or events) must be fulfilled. Usually, this happens if there are enough tokens available in the input places. When the transition fires, it removes tokens from its input places and adds some at its entire output place. The number of tokens removed/ added depends on the cardinality of each arc. In figure I, places PI, P2 initially hold one marker each, as follows:

Before firing: P_1, P_2, P_3, P_4, P_5: = 1, 1,0,0,0.
During firing: P_1, P_3: = P_1, -1, P_3, + 1 if $P_1 > 0$
P_2, P_4: = P_2- 1, P_4+ 1 ifP_2> 0
P_3 P_4, P_5: = P_3,-1, P_{4-1}, P_5 + 1,if P_3> 0 & P_4> 0
After firing: P1, P_2, P_3 P_4, P_5: = 0, 0, 0, 0, 1.

A Petri net is preferred for the following reasons:

(1) It provides formal semantics despite its graphical nature
(2) It models the occurrence of events by the state, and
(3) PN provides an abundance of analysis techniques.

4.2 Workflow Management in Distributed Computing System Using Petri nets

In the process dimension, it is specified which tasks need to be executed and in what order. Modelling a workflow process definition in terms of Petri nets is rather straightforward: tasks are modelled by transitions, conditions are modelled by places, and cases are modelled by tokens. The overall Send/Receive flow of communication between source and destination systems can be represented graphically in Petri net notation using labelled place/transition net as shown in figure 2.

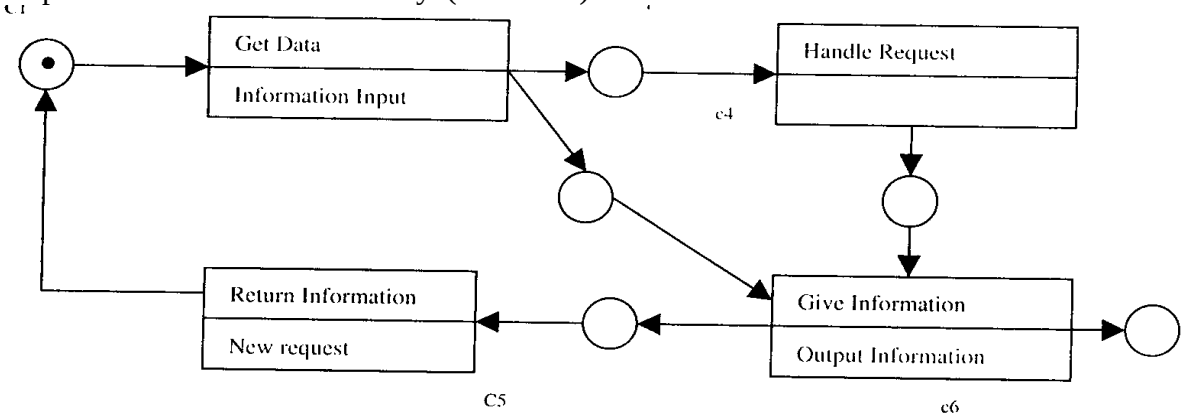

Fig. 2: Labelled place/Transition Net

First, a request is initiated (i.e., availability of token at input place cl. Second, the transition ***Get Data*** fires invoking the service **information. Input**, which supplies the necessary input parameters. Third, the

transition ***Handle Request*** takes over. Finally, by transition *Give Information* the requested information is given and a cost item is created using the service **information. Out.**

Now, to illustrate workflow processes in a specific DCS using the concept of Petri nets we consider the processing of order for the purchase of a "book" via the Internet as in e-purchase (electronic purchase). First the order is registered *(task register),* then a questionnaire is sent to the buyer ***(task send***

The tasks register, send-questionnaire, evaluate, process-questionnaire, time-out, process- order, check processing have been modelled by transitions.

The transitions ***processing-Ok*** and *processing-**NOk*** are added to model the two possible outcomes of executing task, conditions have been added. Each condition is modelled by a place. A Petri net which models a workflow process definition, that is, the life cycle of one case in isolation, is called a *questionnaire)* and the buyer is evaluated ***(task evaluate).*** If the buyer returns a valid data, the *task process-questionnaire* is executed. Where the questionnaire is returned with invalid data, the result of the questionnaire is discarded ***(task time-out).*** Based on the result of the evaluation, the request is processed or not. The actual processing of the order ***(task process-order)*** is delayed until the questionnaire is processed or a time-out has occurred. The processing of the order is checked via task ***check- processing***. Finally, task archive is executed. Figure 3 depicts a workflow process definition for the processing of e-purchase specified in terms of Petri net, work-flow net (WF-net). A WF-net satisfies two requirements. First of all, a WF-net has one input place (I) and one output place (O). A token in a place may belong to either of **send, local** or **receive** state of the system and corresponds to a case which needs to be handled; a token in O corresponds to a case which has been handled. Secondly, in a WF-net there are no dangling tasks and/or conditions. Every task (transition) and condition (place) should contribute to the processing of cases. Therefore, every transition t (Place p) should be located on a path from place I to place O. There is also a causal dependency constraint regarding I, and O as follows (**output ⇒ once input**). This means that the invocation of service output implies that service input has already been invoked at least once.

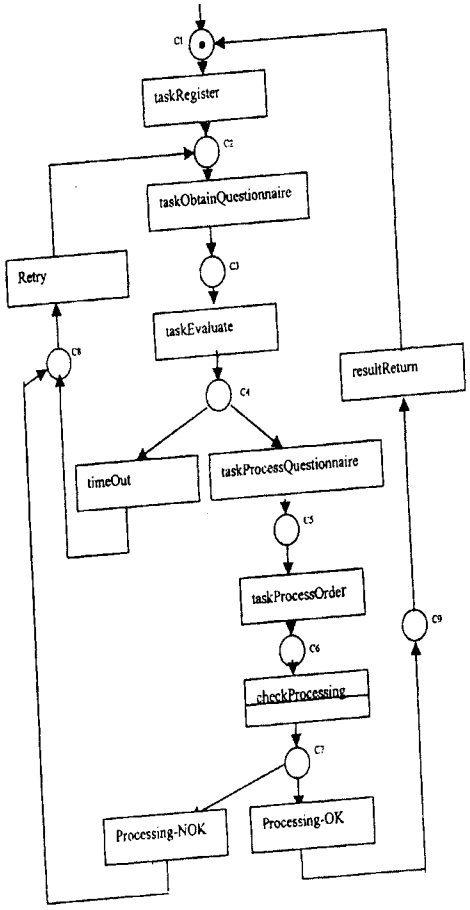

Fig. 3: A workflow process definition for e-purchase

4.3 Analysis of correctness of Workflow managements

Basically, there are three types of analysis that one may examine in the assessment of a workflow management system, namely [10]:

(i) Validation, i.e. testing whether the system behaves as expected,

(ii) Verification, i.e., establishing the correctness of a system, and

(iii) Performance analyses, i.e, evaluating, the ability to meet requirements with respect to throughput times, service levels, and resource utilization.

Because of space, we will only concentrate on verification, that is, establishing the correctness of our system. For this purpose, let us examine a notion of correctness known as soundness. For a workflow system to be correct, i.e., be sound, it must satisfy the following requirements:

a) A WF-net must have a source place **i** (start condition) and a sink place O (end

condition).

b) Each task/condition is on a path from **i** to O (this apply for Petri nets only).

c) The moment the procedure terminates, place 0 holds a token and all other places are empty.

d) There should be no dead tasks; i.e., it should be possible to execute an arbitrary task by following the appropriate route through the WF-net. The soundness property actually explains the dynamics of a WF-net and characterizes workflow design for our case study. A procedure modelled by a WF-net PN = (P, T. F) is **sound** if and only if:

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state 0: $\forall_M (i \rightarrow^* M) \Rightarrow (M^* \rightarrow O)$.

(ii) State 0 is the only state reachable from state I with at least one token in place O: $\forall_M (i \rightarrow M \wedge M \geq O) \Rightarrow (M=O)$

(iii) There are no dead transitions in (PN, i): $\forall_{t \in T} \exists_{M, M'} i \rightarrow^* M \rightarrow^t M^t$.

Since the case study used (see figure 3) satisfies the above requirements (i.e., (a)- (d)), then our work now system modelled after Petri nets must equally operate correctly according to specifications.

5. CONCLUSION

The framework of a distributed computing system is discussed, and its model introduced. This application domain is presented in such a way that Petri net could be easily proffered. Petri net concept is then used to model workflow management in e- purchase, which is an example of distributed computing. This work advances the current understanding of both distributed computation and the use of Petri nets theory for modelling and analyzing processes.

REFERENCES

1. Bakpo, F.S. (2003) "Software Design Modelling with Functional Petri Nets" *Nigeria Journal of Technology [NIJOTECH] University of Nigeria, Nsukka (Accepted for Publication)*

2. Bakpo, F. S. (2002) "Computer Use and Applications" Godjiksons Publishers, Nsukka, 128pp.

3. Eeonomopouls, P (1987) "Petri Nets: A model for the analysis of the behaviour and performance of concurrent systems" Office *and Database systems Research* 87. *Pp.99*-32, *Toronto.*

4. Ikekeonwu, G.A.M (2002 "Commuter Science: a first course" Ephrata press: Lagos: Nigeria. 387pp.

5. Milenkovic, M (1982) "Operating Systems, Concepts and Design, McGraw-Hill Computer Science Series, Singapore, 755pp.

6. Murray, S. M (1987) "On the Use of Negotiation in Distributed Computer systems" *Office and Database systems Research87, pp79-98,Tororuo.*

7. Padberg. J; Weber, H and Sunbul, A (2002) "Petri Net based components for Evolvable Architectures" the transactions of the SDPS journal of design &process science, Vol. 6, No1, pp1-10. USA

8. Petri, C.A. (1962) "Kommunication mit Automaten". Ph.D thesis, Institute for instrumentally Mathematik, Bonn [in German]

9. Savelyev, A and Venda, V (1989')" Higher Education and Computerization". Progress Publishers Moscow. 255 pp,

10. Van der Aalst, W.M. P. (2002) "The Application of Petri nets to Workflow Management" On Internet: http://psim.tm.tue.nl/staff/wvdaalst/publications/p53.pdf