The Nucleus **51**, No. 2 (2014) 281-289

The Nucleus

A Quarterly Scientific Journal of Pakistan Atomic Energy Commission

NCLEAM, ISSN 0029-5698

TOWARDS FORMALIZATION OF INSPECTION USING PETRINETS

M. JAVED, *M. NAEEM, F. BAHADUR and A. WAHAB[1]

Department of Information Technology, Hazara University, Mansehra, Pakistan

[1]Department of Mathematics, Hazara University, Mansehra, Pakistan

(Received February 14, 2014 and accepted in revised form April 14, 2014)

Achieving better quality software has always been a challenge for software developers. Inspection is one of the most efficient techniques, which ensure the quality of software during its development. To the best of our knowledge, current inspection techniques are not realized by any formal approach. In this paper, we propose an inspection technique, which is not only backed by the formal mathematical semantics of Petri nets, but also supports inspecting concurrent processes. We also use a case study of an agent based distributed processing system to demonstrate the inspection of concurrent processes.

Keywords: Inspection, Petri nets, Formalized inspection, Requirements engineering, Mathematical analysis of Petri nets1.Introduction

1. Introduction

Software inspection process (or inspection) was proposed back in 1972 by IBM to improve software quality and to enhance programmer's efficiency [1]. Inspection is the most effective method to identify defects during software development life cycle model (SDLC). It is an umbrella activity, which continues throughout the development process from requirements elicitation to testing and implementation. Inspection is used for cost reduction by removing defects and for other benefits that improve functionality for the users [2].

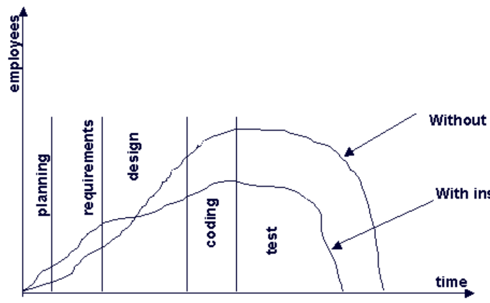


Figure 1. Graph between employee & schedule (borrowed from [1]).

For example, consider a graph plotted for the number of employees required in SDLC against the time required for development in Figure 1. Figure 1 states that companies using inspection based SDLC require fewer employees than those that do not use inspection. Inspection makes the development process more effective by reducing cost and improving throughput [1].

Radicein in [3] showed that the cost of defect removal is much higher in the absence of formal inspection. Further, they proved that the later the defect is caught the greater is the cost paid against that defect. For example, one needs to pay 100, 1000 and 10000 units for each defect caught during inspection, testing and at the time of use by the system users, respectively [3]. This has been depicted in Figure 2 below.

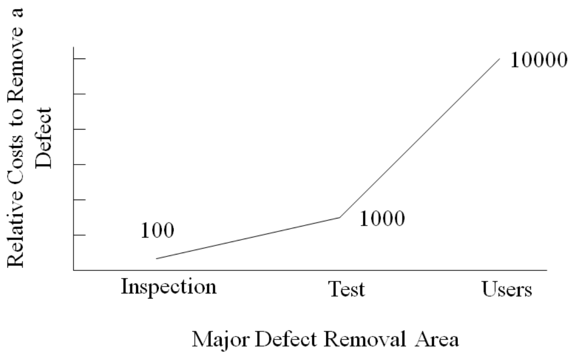


Figure 2. Defect cost relationship (this figure is borrowed from [3]).

Due to the umbrella activity, inspection continues alongwith the SDLC. Because software specification provides building blocks for software, we will consider the use of inspection in requirements engineering. Defects in requirements specification are more harmful than in any other stage in SDLC. Although mathematical requirements use formal verification, they may not perfectly capture the correct objectives [4, 5]. The use of inspection in requirements improves the chances developing the right product [6].

* Corresponding author : mohammadnaeem@gmail.com

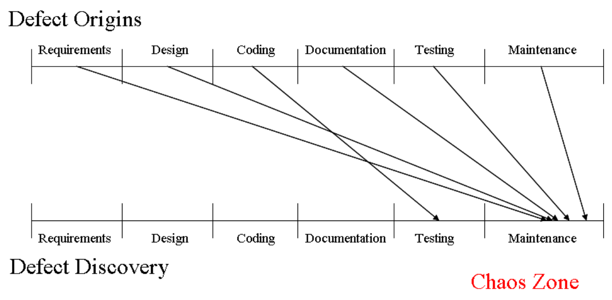


Figure 3. Defect origins and discovery in the absence of inspection [4, 5].

For example, Figure 3 shows that it is important to discover defects at the requirements engineering level. The later we discover the defects in the SDLC, the greater will be the loss. If we get complete, accurate and defect free requirements, it is then a sign of quality product. In the absence of formal inspection in SDLC, more defects will be discovered later (during maintenance phase) which makes them Latent Defects. Latent defects are always harmful for the companies because it causes the loss of time, resources and even the goodwill, just to name a few.

From the analysis of existing inspection methods and techniques we say: It is difficult to find defects from products easily particularly, for requirements because there are no model based (graphical model) inspections. For instance, analysts and designers use UML diagrams to represent process, data, functionality and other aspects of software to get clear picture of the system. Apart from using diagram for the software requirements, diagrams also help in understanding the behavior, inputs and outputs easily. As natural language is ambiguous in nature, its use in either of requirement and inspection may lead to misunderstanding and ambiguity.

By using a method with diagram and mathematical analysis, a software inspector can understand all aspects of the products. Without using a model there are chances of undetected defects even if inspection has been applied. A graphical model with mathematical analysis can catch all defects particularly for concurrent processes because these are difficult to inspect. A Petri net is a powerful tool to model all kinds of processes including concurrency. Furthermore, in case of change request in a product the existing inspection techniques require re-inspecting the complete module, but by using the proposed process, it'll be easier to identify updated part hence easier to re-inspect if needed.

In this paper, we propose inspection by using formal methods based on Petri nets. The rest of the paper is structured as follows: Section 2 provides the information that is important to understand the technical contributions of the paper, whereas Section 3 discusses and compares our approach with state of the art approaches that are closer to ours. In Section 4, we cover the contributions of this paper, while the Section 5 concludes and gives a glimpse of future work.

2. Background

Petri net is a graphical modeling technique, which was discovered by Carl Ada Petri in August 1939 [7]. The graphical models of Petri nets are also supported by the corresponding mathematical equivalences. Petri nets are mostly used to capture the behavior of systems. The behavior can be divided into two parts: one part is a rule stating when a specific transaction will be executable, while the second is the overall behavior, how the executions will occur [8]. Petri net is suitable graphical model for the process of inspection because it has capability to model and inspect different behaviors.

2.1 Concepts and Notations

We use place and transition in the process of inspection, as shown in the left and right of the figure below, respectively.

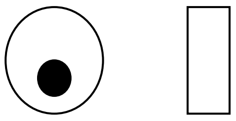


Figure 4. Place (Left) and Transition (Right).

We will refer place as an input or an output and transition by function due to understandable wording in software engineering communities. We will encode input state, output state and function by I_n, O_n and F_n, respectively, where $n \geq 1$.

2.2 Properties of Petri Nets

2.2.1 Concurrency

Petri net is an effective way to model and inspect the concurrency of processes. For example, Figure 5 models the two processes t1 and t2 of concurrent nature. By this property, Petri net can model systems of distributed control with multiple processes executing in parallel.

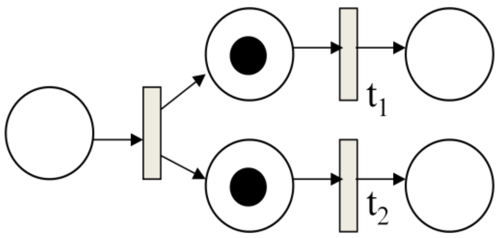


Figure 5. Concurrency in Petri nets.

The concurrent processes contribute towards the increase in the throughput hence, it is important for software companies to consider it in the SDLC.

2.2.2. Priority

Petri nets can be used for the modeling and inspection of the system requiring priority procedure to be implemented. For example, in the Figure below,

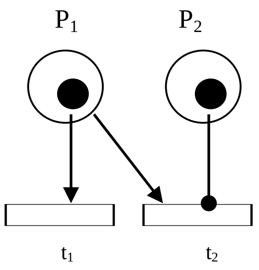


Figure 6. Priority in Petri nets.

t1 has higher priority than t2 so, t1 will be executed first.

2.2.3 Non-determinism

Petri nets provide non-deterministic modeling so, in the situations where a system has to decide from multiple options Petri net can be used to model & resolve conflicts between different processes for instance, in Figure 7 either t1 or t2 can be fired.

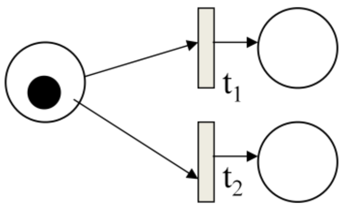


Figure 7. Non-determinism in Petri nets.

2.2.4 Reachability

Reach-ability graph is the property of Petri nets which states whether or not a particular state can be achieved from a respective input state. This property can be used to inspect input and output of a function. For instance, the Figure 8(a) below shows the initial marking of Petri nets. Initially, there were two tokens in the P1 while P2 has no token in it. After firing, there is no token in P1 and two tokens in P2. The Figure 8(b) shows the reach-ability graph illustrating token firing sequence.

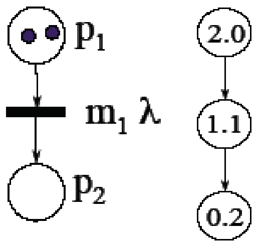


Figure 8. Reachability graph: Initial marking (Left) graph (Right).

2.2.5 Liveliness

The lifetime of a process is shown by the liveliness.

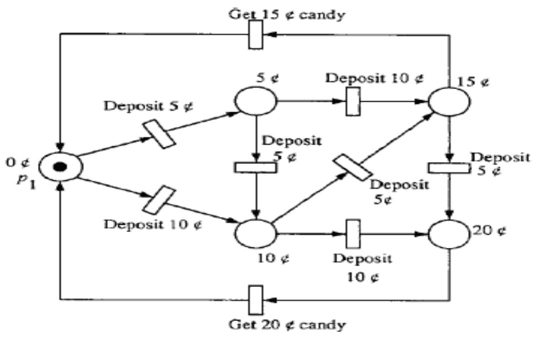


Figure 9. Liveliness of a process by using vending machine.

By this process it would be easy to inspect processes against correct and incorrect inputs. We show the liveliness by using a small example of a vending machine in Figure 9. To get a candy, one has to deposit 15 or 20 cents. If less than the required amount has been deposited then machine would be halted or dead.

2.3 *Validation of Petri Nets*

The validation mechanism of Petri nets used in our paper is inspired by [9]. There are a number of conditions which must hold for the validation of Petri nets. Here we only discuss those conditions that are helpful in our approach of inspection.

1. Before firing a transition tj ∈ T, the following condition must hold

$$M(p_i) \geq W(p_i, t_j), \quad \forall \; p_i \in I(t_j) \qquad (1)$$

If the above condition is satisfied then the transition tj will be enabled. In above Formula 1, pi is an input place of transition tj, while W (pi, tj) is the weight of arc from pi to tj. In the process of inspection M (pi) is also an input with the requirements for the said process and W (pi, tj) is the precondition for the input place pi to the process tj in order to complete the functionality. We can say that to perform the functionality for a process (transition in Petri nets) all input requirements should be completed according to the condition. In the process of inspection this helps to find whether a process with given inputs is executable or not?

2. After firing a transition tj, the next state M'(pi) of Petri net is defined as:

$$M'(p_i) = M(p_i) - W(p_i,t_j) + W(t_j, p_i), \qquad (2)$$

where i =1, 2, 3,, n

Pi is again an input place of transition tj , tokens removed from input place according to the weight of arc are added to the output place according to weight of the arc connecting transition and output place. For inspection, M'(pi) is output place of a process according to the conditions on W (tj, pi) (arc from process to output place), which will consume inputs from M (pi) as per requirements on W (pi, tj) (arc from input to process).

3. Related Work

We categorize this section into: Application of Petri nets which states their use in SDLC which is followed by the discussion on the current approaches of inspection. We will conclude this section by adding the drawbacks of the current inspection techniques. To the best of our knowledge, there is no technique which uses Petri nets in inspection, so we will end up with the discussion about the need of formalized Petri net based inspection.

3.1. Applications of Petri nets

Petri nets are being used in different areas of computer science some of them are listed below.

3.1.1. Software Design

Petri nets are very much helpful to model the systems, especially the interactive system. Petri nets provide a modeling notation which is helpful in capturing dynamic behavior of programs. It helps designing components of software for representing the critical areas [10]. In manufacturing technology deadlocks are very troublesome so, it is very much essential to address this issue. Chao in [11] used Petri nets to provide deadlock control mechanism in sequential processes.

3.1.2. Workflow Management

Petri net is a powerful tool for the analysis of existing systems. It helps to understand difficult workflows. Alongwith designing complex workflow, Petri nets can be used to verify the workflow [12]. Petri net is becoming popular for workflow management systems [13].

3.1.3. Concurrent Programming

Petri nets are being used to verify the process of con-current systems. The application architectures which are based on concurrent processing can be modeled with Petri nets. Barkaoui et al. in [14] have used Petri nets to modularize complex systems to verify the concurrent programming.

3.2. Existing Inspection Approaches

Existing software inspection techniques are listed below [15, 16].

3.2.1. Ad hoc

It is an informal method, which does not require any training. In this technique experienced programmers utilize their experiences to perform inspection.

3.2.2. Checklist-Based Reading (CBR)

This method uses a checklist of questions for inspection. These questions are checked during inspection of systems. It is most widely used method in the development sector.

3.2.3. Abstraction-driven Reading

This technique was developed for the inspection of object oriented code. In this method inspector reads the code in a systematic manner. Each class is inspected alongwith its functions. As natural language abstracts need to be created so in-depth understanding required.

3.2.4. Use Case Reading

In this method, dynamic interaction of objects in object-oriented environment being inspected is checked. This technique helps finding out the usage defects in classes.

3.2.5. Usage-Based Reading

In this method inspectors try to find those defects which effect user. This technique is most effective for design process.

3.2.6. Stepwise Abstraction

This is for poorly documented programs. It is to inspect program functionality.

3.2.7 Scenario-Based Reading

This technique is used for requirement specification. For this method scenarios are created to discover defects.

3.2.8 Perspective-Based Reading

In this technique reviewer has to adopt a prospective. Inspector can adopt designer prospective to verify for the next step of design. User prospective will generate documents understandable to the user.

3.2.9. N-Fold Inspections

This technique is checklist based. N independent teams carry out same inspection with similar check-lists.

3.2.10 Phased Inspection

By this method inspection carried out in all phases of SDLC with a small number of teams. This is also a checklist based inspection.

3.2.11 Traceability-Based Reading

It was developed for design documents to verify the correctness of the system.

3.3 Drawbacks of Existing Inspection Approaches

The existing approaches of inspection do not help understand requirements easily. Furthermore, none of the existing inspection techniques is realized by any modeling technique. The graphical models help inspecting the processes. For example analysts and designers use UML to represent process, data, functionality and other aspects of software to present the system clearly. Using graphical models for inspecting requirements can help to understanding the behavior, inputs and outputs of a process easily. Inspection in the absence of graphical models might leave uncovered defects in the system. A graphical model can catch mostly defects and can help us to clear the ambiguities. In this paper, we are going to use Petri nets as a graphical model for inspection.

4. Petri Net Based Inspection

As already mentioned, we will use Petri net as a graphical model of inspecting the system. Proposed inspection technique can be used in all phases of SDLC, but we are considering inspection for the requirements engineering process. Our approach is able to uncover all type of defects in the requirements e.g. commission, omission, clarity, ambiguity, capacity etc. We use input process, output process and functions in our approach. Our approach consists of two stages:

First stage inspects the individual requirements, while other inspects overall functionality of different modules in a system. Defects found during inspection are written in the defect log and queries related to requirements should be asked to the requirements engineer for complete understanding of system specification.

During the inspection inspector should keep in mind the following types of questions:

1. What, e.g., what types of functions can be per-formed on an input?
2. What if, e.g., what if an unauthorized access of a process happens?

The questions which cannot be answered by the related documents are followed by"➔". For example, a question about the availability of a system after being crashed can be written as: ➔After how much time system should be available, once it crashes.

We inspect every requirement separately with all possible inputs and outputs. The defect log is updated in case a defect is found. At the end of our approach, inspection report should be completed. Inspection report can later be used to: 1) Check the completeness and correctness of the design and code because we will have detailed information of all inputs, processes and outputs; 2) Analyze the behavior and functionality of overall system due to the presence of inspection model based on Petri net notations. Furthermore, test case can also be inspected against each requirement's inspection result.

For the application of our approach we borrowed SRS from [17]–a document of social security services of South Africa. The main idea behind that SRS is: The constitution of South Africa makes provisions of financial benefits for their needy citizens. This benefit works as a poverty alleviation program of the government. This scheme is getting mature and the national government has developed norms and standards for the implementation of policies relating to this. Let us now start inspecting the requirements provided in SRS.

4.1 Inspecting Individual Requirements

At this stage we inspect requirements individually. For the elaboration of this stage, we apply the proposed inspection process on the subset of the requirements taken from the SRS provided in [17]. For the application of inspection we move on as:

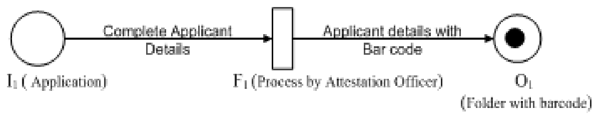
1. The description of a requirement in natural language.
2. Inspection shown in the form of Petri nets which is followed by its corresponding equivalence in the mathematical form.
3. Input, Process and Output explain the main input, functionalities at the processes shown in inspection, and the output of a process involved in the inspection respectively.

In each inspection we show the diagrammatic representation of requirement by using Petri net followed by the mathematical equivalence of the functionalities captured in the diagram. For example, M (pi) = M (Ii) captures the fact that initial marking of pi in Petri net is equal to the input values at input place Ii. The mathematical formulas are followed by the description of inputs and the questions that are not answered by the relevant documents. Listed below is individual inspection applied on SRS for Department of Social Developments System [17]:

4.1.1 Requirement # 1

A grant application is submitted to the Attesting Officer. Attesting officer collects the information from application form; manual information will be replaced with new information submitted by applicant, which are Applicant name, ID number, Date of application (cannot exceed current date), Grant Type, Race, Pay Point Number, Service Point, District Office and Form Number.

Inspection


Complete Applicant Details
Applicant details with Bar code
I1 (Application)
F1 (Process by Attestation Officer)
O1
(Folder with barcode)

As

$$M(p_i) = M(I_1) \quad \text{and} \quad W(p_i, t_j) = W(I_1, F_1)$$

So,

$$M(I_1) \geq W(I_1, F_1)$$

Input

● Complete Applicant detail

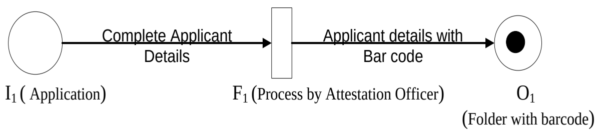
➔ Whether these are complete inputs to enable/perform F_1?

4.1.2 Requirement # 2

a) After ensuring that the application is in order, Attesting Officer opens and bar codes the application. In this process, three barcodes are printed: The first and the second barcodes are attached to the original form and its carbon copy respectively, whereas the third is attached to the folder containing the application; this folder will be stored in the registry.

b) After adding the bar codes, application is moved to the Verification Officer. The amount of time each Officer spends on an application will be monitored.

Inspection


Complete Applicant Details
Applicant details with Bar code
I1 (Application)
F1 (Process by Attestation Officer)
O1
(Folder with barcode)

As

$$M(pi) = M(I1) \quad W(pi, tj) = W(I1, F1)$$

$$M'(pi) = M'(O1) \; W(tj, pi) = W(F1, O1)$$

So,

$$M'(O1) = M(I1) - W(I1, F1) + W(F1, O1)$$

Input

Application for grant

Process $\square$ On F1 Attestation Officer will

- Attach bar codes on original, carbon copy and folder.
- Scan the application

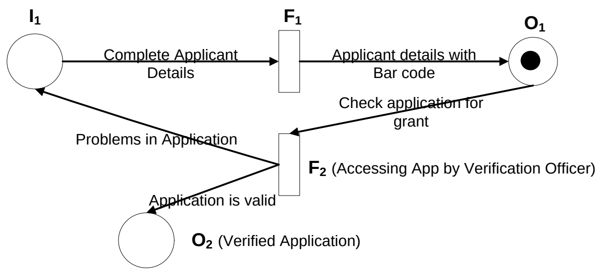
Output

- Folder with barcodes (Which will be the output on O1)
- Scanned application
- What is the format of barcode?
- What kind of values would be saved alongwith scanned document?

4.1.3 Requirement # 3

Any problems experienced with the grant application will result in the Verification Officer returning the grant application to the Attesting Officer. The reason for returning the application must be recorded for the purposes of monitoring these problems.

Inspection


I1
F1
O1
Complete Applicant Details
Applicant details with Bar code
Check application for grant
Problems in Application
F2 (Accessing App by Verification Officer)
Application is valid
O2 (Verified Application)

As,

$$M(pi) = M(O1) \quad \text{and} \quad W(pi, tj) = W(O1, F2)$$

So,

$$M(O1) \geq W(O1, F2)$$

➔ What information will be required to access application?

If this condition is true

As,

$$M(pi) = M(O1) \qquad W(pi, tj) = W(O1, F2)$$

$$M'(pi) = M'(O2) \qquad W(tj, pi) = W(F2, O2)$$

So,

$$M'(O2) = M(O1) - W(O1, F2) + W(F2, O2)$$

OR

If condition is false

As,

$$M(pi) = M(O1) \qquad W(pi, tj) = W(O1, F2)$$

$$M'(pi) = M'(I1) \qquad W(tj, pi) = W(F2, I1)$$

So,

$$M'(I_1) = M(O_1) - W(O_1, F_2) + W(F_2, I_1)$$

Input

- Application folder with barcodes

Process On F2 Verification officer will

- Access the application.
- Send back rejected application
- Record Reasons of rejection.
- What is the format to record reasons?

Output

- Approved application

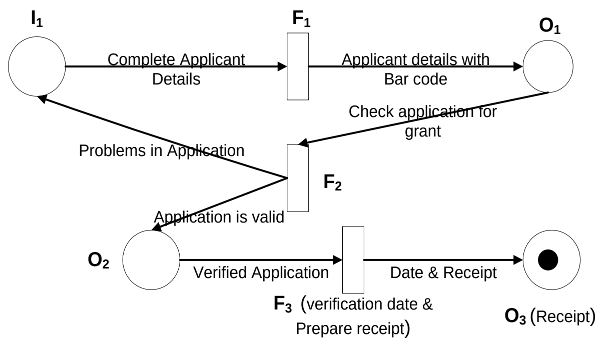
OR

- Application need to reconsider

4.1.4 Requirement # 4

Verification Officer prepares the receipt for the applicant of the valid applications.

Inspection


I_1
F_1
O_1
Complete Applicant Details
Applicant details with Bar code
Check application for grant
Problems in Application
F_2
Application is valid
O_2
Verified Application
Date & Receipt
F_3 (verification date & Prepare receipt)
O_3 (Receipt)

As,

M(pi) =M(O2) W(pi, tj) = W(O2, F3)

M'(pi) = M'(O3) W(tj, pi)= W(F3, O3)

So,

M'(O3) = M(O2) - W(O2, F3)+ W(F3, O3)

Input

- Approved application

Process On F3 Verification officer will

- Write down verification date.
- Prepare receipt for applicant.

Output

- Receipt for applicant with barcode

➔ What information will be written on receipt?

4.2 *Inspecting Overall Functionality*

Inspection of the requirements that apply on system as a whole can be dealt at this stage, e.g., non-functional requirements and other features like con-currency and priority. Inspection of the properties that applies to system as a whole helps discovering the defects like improper synchronization, deadlock, starvation, lost signals, and race conditions etc.

To check the application of combined inspection, let us use an example of an agent based distributed processing which requires the interaction of multiple processes at a time. In this case source platform creates two agents for service and data discovery, respectively. These agents are transmitted on the network simultaneously. We assume that these agents have already been inspected separately.

Possible questions at this moment could be:

- What should be the behavior of system when both Agents reach source at same time?
- What if one of them trapped?

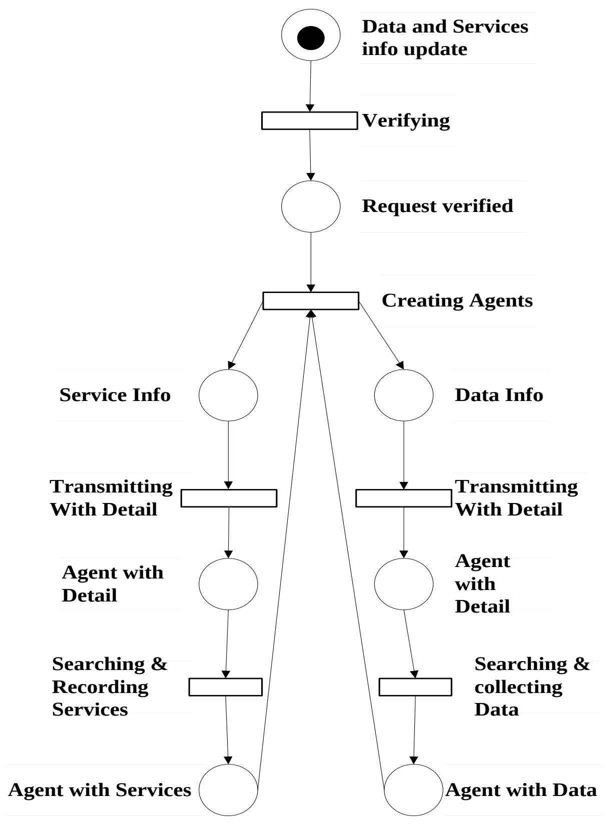


Figure 10. Inspection of overall system.

4.2.1 Analysis of System

System's functionality inspected by Petri net can be analyzed and audited by mathematical properties i.e. reach-ability, liveliness etc. We will consider both in our approach. This property tells the sequence of transitions on given marking in Petri net. Having a sequence means that the desired state is reachable from the current state. In inspection audit and analysis this reachability can help us to find whether this module or component can perform all tasks without any defect on the given input or not. In the following Figure 11, we show the reachability graph for the analysis of the agent based module inspected and shown above in Figure 10. While doing the analysis we use vector = (Data and Services info update, Request verified, Data Information, Service Information, Agent with Detail, Agent with data, Agent service).

We present the analysis at five different stages. On each stage, 1 and 0 represent the existence and absence of token in the corresponding vector attribute, respectively. The analysis data captured in Figure 11 is explained as:

Stage 1	Initially, there is a token in Data and Services information update,
Stage 2	After verification, token reaches to the next place, i.e., request verified,
Stage 3	When an agent is created then token is placed to both data information and service information places.
Stage 4	When an agent is transmitted with detail then token passes to agent with detail position,
Stage 5	After searching, token is moved to agent with data and agent service positions.

This analysis verifies the inspection process; it is reachable from the place Data and Services Information Update to the Software Agents with Data and Services.

5. Conclusions and Future Work

This paper presents the first step towards Petri net based inspection of software systems. The graphical models used in the paper are also supported by their corresponding mathematical formulas. The proposed approach is equally important to discover defects in the requirements separately and on the system requirements that can be implemented on as a whole. To the best of our knowledge, this is the first attempt to provide formal semantics of inspecting concurrent processes in software systems. In future, we will focus on the automation of our approach and will try to use extended case studies. There are some simulation and analysis tools like ALPHA/Sim, AlPiNA, ARP, Artifex, CoopnBuilder, COSABPM, CPN-AMI, CPN Tools, ePNK [18] available for Petri nets simulation, we will also see how our approach of inspection can benefit from those tools.

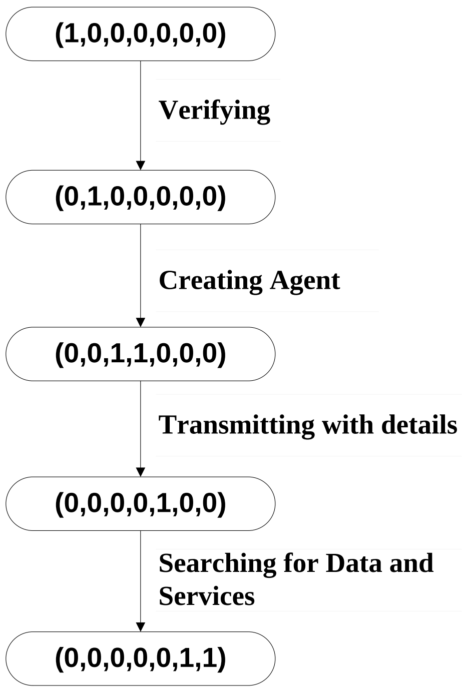


Figure 11. The analysis of system.

References

[1] M. E. Eagan, Advances in Software Inspections, IEEE Transactions on Software Engineering **12**, No. 7 (1986) 744.

[2] T. D. Crossman, Some Experiences in the Use of Inspection Teams in Application Development, In Application Development Symposium, Monterey, CA (1979).

[3] R. A. Radice, Software Inspections, Methods & Tools. **10**, (2002) 7–20, http://www.methodsandtools.com/PDF/dmt0202.pdf.

[4] Jones, Estimating Software Costs,Tata McGraw-Hill Education (India), 2ndEdition (2007), http://books.google.co.uk/books?id= pdcsm KljT9QC.

[5] C. Jones and O. Bonsignour, The Economics of Software Quality. Pearson Education (2011).

[6] D. L. Parnas and M. Lawford, IEEE Transactions on Software Engineering **29**, No. 8 (2003) 674.

[7] C. A. Petri and W. Reisig, Petri Net, Scholarpedia, **3**, (2008) 4: pp.7–20, http://www.methodsandtools.com/PDF/dmt0202.pdf.

[8] J. Desel and G. Juhás, What is a Petri Net?" Informal Answers for the Informed Reader, In Unifying Petri Nets, Springer-Verlag Berlin Heidelberg (2001) pp. 1–25.

[9] C. Cassandras and S. Lafortune, Introduction to Discrete Event Systems, SpringerLink Engineering, Springer (2007).

[10] U. Farooq, C. Lam and H. Li, Towards Automated Test Sequence Generation, In 19th Australian Conference on Software Engineering, (2008) pp. 441 –450.

[11] D. Y. Chao, Journal of Information Science and Engineering **25**, No. 6 (2009) 1963.

[12] W. M. P. van der Aalst, The Journal of Circuits, Systems and Computers **8**, No. 1 (1998) 21.

[13] K. Grigorova, Process Modeling using Petri Nets, In International Conference on Computer Systems and Technologies (2003).

[14] K. Barkaoui and J.-F. Pradat-Peyre, Verification in Concurrent Programming with Petri Nets Structural Techniques, In The 3rd IEEE International Symposium on High-Assurance Systems Engineering. Washington, DC (USA) IEEE Computer Society (1998) pp. 124–133.

[15] D.A. McMeekin, A Software Inspection Methodology for Cognitive Improvement in Software Engineering. Ph.D Thesis, Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology, Australia (2010).

[16] R.O. Oladele, Reading Techniques for Software Inspection: Review and Analysis. Technical Report. Department of Computer Science, University of Ilorin, Ilorin, Nigeria (2010).

[17] Software Requirements Specification (SRS) Document: Workflow Monitoring, Social Security SRS document of south African Social Security Agency. Developed for Social Development Eastern Cape Govt. Online at: http://www.sassa.gov.za/Portals/1/Documents/7f4f791d-a512-4abc-99ff-887586ff431a.pdf, (2005).

[18] A. Comprehensive Database of Petri Nets Tools, http://www.informatik.uni-hamburg.de/TGI/ Petri nets/tools/db.html (February 2014).