GENERATING PETRI NET MODELS FOR BASIC JAVA PROGRAM

V. L. Narasimhan (Supervisor) and A. T. Aung (Student)

School of Electrical & Electronic Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
e-mail: eeeurop@ntu.edu.sg

Keywords: Petri Net, Modelling, Java Program

Abstract

This paper will examine the technique of transforming a Java program into the Petri Nets GUI for later design analysis. Here, the Java program is trimmed to a basic minimum constructs in a preliminary design stage. The transformation is to be written in Java language.

1 Introduction

Industrial Java programs have been too long and complex so that to analyze these kinds of programs, a human reader needs software supports. Petri Net is one of the best tools suitable for this kind of analysis. Currently, in the market, no such tool exists. In this project, by trying to transform a ‘basic’ Java program into a Petri Net, the feasibility of creating this kind of tool has been studied.

2 Theory of Petri Net

Petri Nets have come into academic focus when Carl Adam Petri wrote his PhD on the subject in 1962. A Petri net can be used in the analysis of the flow of information as an abstract modelling tool. [1] Petri Nets can be represented in mathematical as well as graphical forms. [3]

Petri Nets study can be divided into two theoretical directions: pure theory and applied. Pure Petri Net theory studies the techniques, concepts and mathematical foundations needed for applications of Petri Nets, while applied theory focuses upon the utilization of Petri Net models in the analysis of systems. [2]

A Petri Net is composed of four parts: a set of places \( P \), a set of transitions \( T \), an input function \( I \), and an output function \( O \). The input and output functions relate transitions and places. [2] Mathematically, Petri Net can be represented as a mathematical structure or as a bipartite directed multi-graph, where the latter graphical representation is more obvious in terms of analysis and more explicit to the visually-oriented human reader.

The following is an example of a Petri Net graph:

![Example Petri Net Graph](image)

Figure 1: An Example Petri Net Graph

This graph can be represented as a mathematical Petri Net structure as follows:

\[
C = (P, T, I, O)
\]

\[
P = \{P1, P2, P3, P4, P5, P6\}
\]

\[
T = \{T1, T2, T3, T4\}
\]

\[
I (T1) = \{P1\} \quad O (T1) = \{P2\}
\]

\[
I (T2) = \{P1\} \quad O (T2) = \{P3\}
\]

\[
I (T3) = \{P2\} \quad O (T3) = \{P5\}
\]

\[
I (T4) = \{P3, P4\} \quad O (T4) = \{P6\}
\]

This Petri Net structure is 4-tuple. The given example is a foundation structure for the Petri Net. One of the most important concepts of Petri Net in terms of simulation is the tokens and their marking in the places of the Net. The marking of a Petri Net can be defined as a topological distribution of tokens across the Nets at certain time. ‘Firing of a transition’ (referred to Section 3) may change the marking of the Petri Net. Therefore, some definition of Petri Net structure includes an
extra dimension called the initial marking M0. [3] The Petri Net structure then becomes 5-tuple: C = (P, T, I, O, M0).

When tokens are involved in a Petri Net, the need for defining the token-capacity of each place is apparent. Theoretically, each place can hold infinite numbers of tokens. Practically, this token-capacity of a place can be defined to a finite number in accordance with the modelled system’s requirements.

3 Rules and Representations of Petri Nets

Applications of Petri Nets are mainly in the modelling of the systems. Since the behaviour of many systems can be characterized by their states and changes of states, the Petri Nets, intrinsically built from states and transitions, are perfect tools for system modelling.

Again, the working of Petri Nets can be mostly understood by the analogy of the real-world system. According to Murata’s Paper, places can be subdivided into input and output places. [3] Input places are likened to preconditions, input data, input signals, etc. Output places are likened to post-conditions, output data, output signals, etc. Transitions are representations of event, computation steps, processor, etc.

In a Petri Net graphical representation, arrows can be seen to show the direction of information flow. In Petri Nets’ terminology, these arrows are called arcs, mainly serving the functional relationship direction indicators between places and transitions or vice versa.

To capture the dynamic nature of modelled systems, tokens and their markings add simulational capability to the Petri Nets under the constraints of the ‘firing rules of transitions’.

A transition T is said to be enabled when every input place has at least a token each. Enabled transition can fire, i.e. each token from all of its input places is removed and one token each is added to all of its output places. [3]

On a Petri Net graph, tokens are represented by small dots in the circles which represent the places of a Petri net.

Applications of Petri Net can be found in manufacturing system, hardware modelling, modelling and analysis of distributed software systems, concurrent and parallel program analysis, etc.

4 Sequential Programming and Petri Nets

Peterson’s Paper [1] on Petri Nets has discussed about software modelling with Petri Nets for sequential programs. In this paper, a brief review upon translation of flowcharts into Petri Nets has been illustrated.

Mainly, the nodes of the flowchart can be replaced by places; while the corresponding arcs, replaced by transitions on the Petri Net.

A sequential program is divided into sections, where section boundaries are mostly decision-making statements or conditions. These section boundaries become transitions. The statements in each section are mostly initialization statements or assignments. These sections form into separate states of the program, i.e. they become places in the resulting Petri Nets.

5 Object Oriented Programming and Petri Nets

The principles and practices of object-oriented programming have been dated back since 1960s, the same period when Petri Net was born into academic world. Unlike sequential programming, object oriented programming is based on objects which are at a certain state at a certain time. These states can be changed by sending or receiving ‘messages’.

Petri Net, which is designed to capture the states of the system and transitions of the system, is obviously become a natural model for any OOP language. It can be easily deduced that states of an object can be represented as places in Petri Nets. The message which is sent or received can be modelled as a transition.

6 Java and Petri Nets

Java is an object oriented programming language invented in the mid 1990s at Sun Microsystems. Most of its syntax is derived from another popular OOP language C++.

Java has been designed so that multithreading and networking becomes much easier and simpler than other languages. Therefore, Java has been now increasingly and widely use in concurrent programming, parallel and distributed programming.

As amount of work in these fields increases, the complexities of concurrency and distributed objects are a challenge to the system analysts and designers. Most of them use software tools to aid their analysis. UML and Petri Nets are among the top candidates.

The existing Petri Nets tools, most of them are written as a Java applet or GUI application, are manual in the following aspect. They have given the user GUI facility to draw Petri Nets, which the user must have been derived from his system priory. Also, simulation capability is added in the way that the user can manipulate the tokens and check the firing sequence.

In this way, system analysts and designers can utilize them in their analysis of respective systems. One important point in this usage of Petri Nets is the users are mostly designers of the systems, i.e. they have known foundation designs from the start; they are using these tools to improve the performance and/or to check the errors and discrepancies.

Here comes an important question. What if the design of the system has been unknown? Can we still use Petri Nets in this case?
7 Petri Nets as a Design Recovery Tool

Suppose we have to analyse an old program with several million lines of code and no blueprints are available for this legacy code. It is practically impossible, at least in terms of time and efficiency, for a human reader or programmer or analysts to check and comprehend all the lines of code, let alone improve it.

If there is a tool which can read the code and transform it into a visual representation, then this would mean the task can be accomplished by the tool doing transformation and leaving only analysis to the human reader.

The objective of the UROP project is to check the feasibility of this kind of tool. More specifically, by building the simplest software tool which can transform ‘basic’ Java program into Petri Nets, the project aims to lay foundation for the general tools which have ability of automatic transformation of programs into respective visual representation for further analysis. Here, Petri Net is chosen as final visual representation because of its inherent ability for analysis and its solid mathematical background.

8 Foundation for Implementation

The term ‘basic’ in “basic Java program” can be defined using the following assumptions:

1. The given program is correctly compiled, i.e. error-free and ‘grammatically correct’.
2. Up to one basic construct ‘if-else’ structure is considered.
3. Source program has only primitives and base classes provided by sun.
4. The program has a main method, i.e. application type.

As a preliminary version, minimum complexities are preferred. After workability has been proved, extension must be considered. Therefore, the basic ‘transform program’ must be built upon flexible design and structure for accommodating further add-on expansions.

Therefore, transformation is done in two-stages process. First stage is to transform the given basic program into an intermediate representation. Second stage is to translate this intermediate representation into Petri Net GUI.

As a guideline for transformation process, the following assumptions have been set:

1. Object with its field creates a state, and the object will have many states throughout program execution. Each state is represented as a place in a Petri Net.
2. If one or more of its field value or values change, then its state changes. State changes can be triggered by message sent or received, events driven, so messages and events are transitions.

After these assumptions, the format of intermediate representation can be decided. Since the program has been logically grouped into places and transitions, the intermediate representation will be a data structure which will store these logical groups, relationship between these groups in a format easily accessible from a GUI-creating program.

The second stage has been quite straightforward when an intermediate data structure has been created. A GUI program must draw places, transitions and connecting arcs according to the data stored in the structure.

9 Implementation Ideas

The language used for the ‘transformer program’ is Java programming language itself.

The implementation can be summarised into the following flowchart:

![Figure 2: Implementation Flowchart](image)

For transforming input program into intermediate representation, the process is quite similar to earlier stages of compilation process. The main idea behind is to create a table-driven parser, i.e. trying to achieve flexibility in terms of complexity in implementations. Actually, algorithms are already present for parsers, intended for the construction of compilers. But, direct adoption of these algorithms are obviously impossible since, in our case, for example, the grammatical check is not needed since the program is already compiled and assumed to be correct.
Main base classes used for implementations are java.io.StreamTokenizer, java.util.StringTokenizer and java.lang.String.

From an intermediate form to Petri Net GUI, the process needs dynamic modification of GUI components. The process will mainly use javax.swing components.

10 Final Remark

Due to the time constraint of UROP project, the full implementation of the program is still to be worked out. Since the implementation process has been built upon a firm theoretical foundation and a practical research, the complete program can be realized in two weeks of intensive programming.

Acknowledgements

First of all, the author wishes to thank Dr. V. L. Narasimhan for his kind and careful supervision throughout the execution of the UROP project.

I would like to thank all those individuals from the School of Electrical and Electronic Engineering, Nanyang Technological University who have given me the opportunity to explore into an academic research exercise.

References