

# NOPL Compilation to Generate C# Code for Notification Oriented Paradigm Framework

Paulo Renaux

**Abstract**—The solidified imperative and declarative paradigms are well known for their advantages and their disadvantages, thus the proposal of a Notification Oriented Paradigm. It is possible to implement a NOP application either in some existing languages (e.g. C++, Java, C#, VHDL) or in the NOP specific language, which requires translation to be compiled. This document will briefly describe the technologies used, such as NOP, LingPON (NOP Language), and NOP C# Framework applications. This document will describe with more emphasis how a new layer was added to the NOP Compiler and generated to allow a NOPL code execute as a NOP C# Framework application.

**Index Terms**— Notification-Oriented Paradigm; Compilation; Software Development; C#.



## 1 INTRODUCTION

This document will present the implementation and results for development of a compiler that allows a LingPON (Linguagem PON – Language for Notification Oriented Paradigm) code to be rewritten as a C# code, such conversion allows a NOP application in C# to be compiled as an executable code.

This document will firstly describe the topics required for better its better comprehension, such as NOP and NOPL. After describing these topics, this document will describe how the previously created compiler now can output a code which can be compiled into an executable application.

### 1.1 NOP

NOP (Notification-Oriented Paradigm) is a programming paradigm that aims to overcome the difficulties that are present in Imperative Paradigms, such as coupling and redundancy, and in Declarative Paradigms, such as inference mechanism processing overload and also coupling. [1],[2]

#### *NOP Elements*

A NOP application consists on the following elements: Rule, Premise, Condition, Action, Instigation, Fact Base Element, Attribute, and Method.

A Rule is composed by a condition and an instigation. The rule’s action will occur when it’s condition is met; the condition is met when each of its premises are

satisfied. The rule’s action can trigger several instigations to occur sequentially.

A Fact Base Element (FBE) is an abstraction of a real world object. The FBE has each of its characteristics and states translated into an Attribute. Each of the FBE’s possible actions are translated into Methods.

When an attribute value changes, a notification is sent to all premises in which they are evaluated. When a premise is valid, it notifies a condition.

The aforementioned elements communicate themselves via notifications for the following mechanism:

1. An attribute notifies a premise;
2. A premise notifies a condition;
3. A condition, as part of a rule, activates an action, or, in a more complex system, a condition notifies an action.
4. An action notifies an instigation;
5. A method, as part of an instigation, can be called by one;
6. As implementation of a method, an attribute status can be changed, closing the loop.

## 1.2 LingPON

LingPON (Linguagem PON or NOP Language) is a programming language that defines each FBE (Fact Base Element), Rules, and their related elements. After FBEs and Rules are defined in LingPON, these are translated as a NOP application in C++.

NOP applications were firstly implemented in C++ and nowadays can be implemented in Java and C# to generate an executable application.

## 1.3 PON Compiler

As aforementioned, FBEs, Rules, and their related elements are translated to an executable NOP application. This process occurs by following a process in which a LingPON code is translated into a stream of tokens (corresponding to the lexical analysis), semantically analyzed to produce a data structure, known as symbol table. Such symbol table, when complete, contains all information needed to produce the application in targeted code. [3]

For the NOP Compiler, a set of C++ classes contain a data structure to represent and storage of information relevant for the compilation process. [3]

## 1.4 NOP C# FRAMEWORK

The NOP C# Framework can be defined as a set of C# classes which abstract the NOP mechanism and elements in a manner that allows development of applications following this programming paradigm using said language.

As aforementioned, a NOP application can be implemented in C# via a framework. Implementing an application using the C# framework requires the code to follow a syntax, grammar and definition order. An example of syntax change between LingPON and NOP in C# would be the definition of an FBE is presented in the following figures 1 and 2. Such conversion implies that a compiler can be used to convert from LingPON to NOP Framework in C#, as described in the next section.

```

fbe Test

    // attributes
    // methods

end_fbe

```

Figure 1: declaring an FBE in LingPON

```

using System;
using System.Collections.Generic;

class Test : FBE
{
    // attributes
    // methods
}

```

Figure 2: declaring an FBE in Framework C#

## 2 IMPLEMENTATION

To implement this LingPON compiler for a NOP C# application, the following steps were followed:

1. have knowledge of the compiler's data structure;
2. have a base code to adapt (the base codes to adapt were the source codes for a compiler outputting Java code and for a compiler outputting C++ code);
3. have knowledge of how each data structure translates to a C# NOP application;
4. implement each adaptation;
5. verify that data structure was fully output to a NOP C# application code;
6. verify that the NOP application code can be compiled;
7. verify that the NOP application has the expected output.

### 2.1 Steps 1 to 4

Knowledge of the compiler data structure was obtained by analyzing its source (.cpp) and header (.h) code. The base codes to adapt were acquired in the NOP compiler project repository.

Knowledge of how each data structure translates to a C# NOP application was obtained by analyzing the example and source code acquired from the NOP Frameworks repository under C# NOP Framework.

The adaptations on the base codes were done by analyzing how information was extracted from a data structure (e.g. name of an attribute type) and afterwards changing the information value if required before printing it to the target source code.

### 2.2 Steps 5 and 6

Using C# comments to view non-formatted acquired information of if such information was wrongfully acquired, it is possible to fix the compiler code to correctly acquire and format such information.

Using Visual Studio 2017 to compile the output code it is possible to verify grammatical and semantical errors such as attribute visibility among classes and inheritance among classes. When such error occurs, it is fixed by changing the compiler code responsible for it, such as a missing left bracket or a misnomer.

### 2.3 Step 7

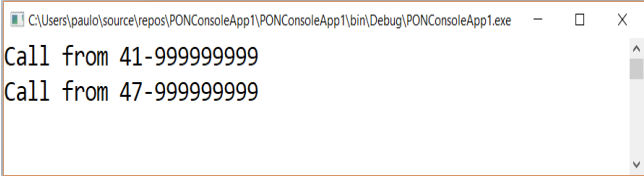
Using the expected output obtained by executing the base example, it is possible to compare it with the output generated given example as a C# NOP application.

If an error caused by the output occurs due to the generated code, the corresponding code segment in the LingPON C# compiler is fixed.

In this step, an error occurred outside the generated code, being fixed manually in the framework.

## 3 RESULTS

Summarizing the aforementioned process to obtain results, to verify that the C# code was generated correctly, it should be able to be compiled and its execution output matches the expected output, as seen in Figure 3.



```

C:\Users\paulo\source\repos\PONConsoleApp1\PONConsoleApp1\bin\Debug\PONConsoleApp1.exe
Call from 41-999999999
Call from 47-999999999
  
```

Figure 3: Generated Program Output

As verified that both output matches, it is possible to confirm that the NOP compiler for C# is able to generate correctly a code in C# for NOP applications.

## 3 CONCLUSION

In conclusion to this process of research and iterations of development, testing and output analysis, implementing such compiler required knowledge of how any compiler and how the source language (LingPON) translates into a data structure to be converted into another language.

Analyzing the results, it is possible to ensure that the analysis components of the compiler and the code generating components are working for outputting a source code into a functional C# NOP Application.

### ACKNOWLEDGEMENTS

The author would like to thank the course colleagues Leonardo T. Silio and Adriano F. Ronszcka, which contributed with their prior knowledge of the LingPON compiler; and also the course professors, who made the project for this report possible.

### REFERENCES

- [1] Mendonça, Igor T.M, "Metodologia de Projeto de Software Orientado a Notificações". UTFPR. 2016.

- [2] Ronszcka, Adriano F. “Contribuição para a Concepção de Aplicações no Paradigma Orientado a Notificações (PON) sob o viés de Padrões”. UTFPR. 2012
- [3] Ferreira, Cleverson A. “Linguagem e Compilador para o Paradigma Orientado a Notificações (PON): Avanços e Comparações”. pp. 89-90. UTFPR. 2015.