Jaroslaw WOJCIECHOWSKI
TECHNICAL UNIVERSITY OF LODZ, DEPARTMENT OF MICROELECTRONICS AND COMPUTER SCIENCE

From formal methods to implementation based on Petri Nets model of concurrent systems

MSc Jaroslaw WOJCIECHOWSKI

Author graduated in 2002 in computer science with specialization in software engineering and network systems. Is occupied in Java technology especially applied to e-commerce. Is working on formal methods in application in applied software engineering.

e-mail: jwojcie@dmcs.pl

Abstract

Purpose of this work is to suggest a path from formal methods to implementation in designing concurrent system, thus helping further stages of systems development to go on. Author focuses on mapping of nonhierarchical Coloured Petri Nets model to class model of the system in Java and C language. Author extends among others formal model with information which would imply generation of class models from formal model, conforming to Java specification and C language, making continuous integration possible. The whole cycle would be presented with changed Petri Nets model of simple concurrent system.

Keywords: Petri Nets, formal methods, mapping, Java class, C language.

Od metod formalnych do implementacji na przykładzie modelu w sieci Petriego systemu współbieżnego

Streszczenie

Celem pracy jest zaproponowanie ścieżki przejścia od modelu formalnego systemu opisanego siecią Petriego do implementacji. Autor skupia się na rzutowaniu niehierarchicznych modeli sieci do modelu klas odpowiadającemu obiektowemu paradygmatowi programowania języka Java i proceduralnemu dla języka C. Autor rozszerza model formalny o informację umożliwiającą dokonanie konwersji do modelu klas i procedur. Cały cykl będzie zaprezentowany na prostym systemie współbieżnym.

Słowa kluczowe: Sieci Petriego, metody formalne, rzutowanie, modele klas w Java, procedury w języku C.

1. Introduction

Techniques for mapping concurrent systems modelled with Coloured Petri Nets to detailed design and coding is presented. Places and transition are augmented with consistent, complementary class model/procedural functions. The validation information will be considered later but not in this article. The information is the extracted from model's xml files and generation is accomplished. Rules for extraction are described by and identification and code generation is done by DOM [5] parser.

2. Coloured Petri Nets

Coloured Petri Nets (CPNs) is one of several mathematical representations of discrete distributed systems. As a modelling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations.

CPN is a language for the modelling and validation of systems in which concurrency, communication, and synchronization play a major role.

Coloured Petri Nets is a discrete-event modelling language combining Petri nets with the functional programming language Standard ML [3, 4].

Standard ML provides the primitives for the definition of data types, describing data manipulation, and for creating compact and parameterisable models.

3. Mapping to Java language

Author assumes that the CPN model is proper and the following mapping does not affect simulation of the CPN model. Mapping algorithm to generate class model [8, 9] from CPN model in Design/CPN application to class diagrams is as follows.

a) One should mark the place with initial class place tag as on fig. 1. The place is chosen as basic to generate implementation class. The comment with class name (*initialClassPlace@classname:object1*) or the colour in place will be the name of the class.

The attributes to the class may be additionally added to initial class place tag as text example.

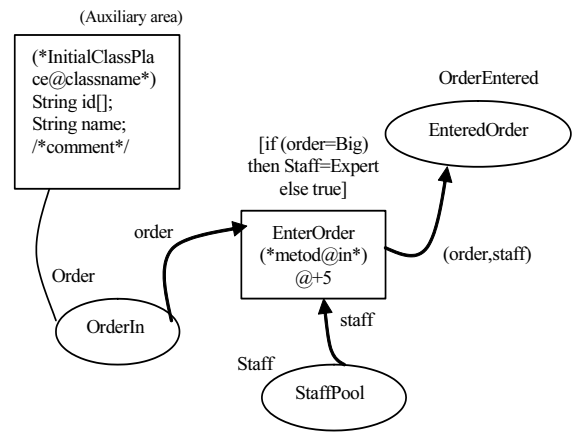

Fig. 1. CPN model annotation example
Rys. 1. Przykład modelu CPN z dodatkowymi oznaczeniami

The initial class place tag is connected with the place with the auxiliary connector (specified in Design/CPN).

b) The methods of the class will be relevant to transitions which are connected to the place with arcs which was marked as initial class place. The guard code on transition will be set as comment in the body of the method.

- The parameters of the method are the colours from the places which are connected to this transition but directed to the transition.
- If we want to restrict the amount of parameters of the methods than we mark the arc with tag (*method@in*). The returning value from the method is the value of arc directed from the transition or denoted by (*method@out*) as on fig. 2.
- If we do not want all the transitions to be taken as methods than we mark the transition with (*method@nottaken*) (Conditions on arcs will not be taken into account in this article).

c) The same steps from a) to b) we proceed with another place which will be the initial class place.

Syntax for denoting other dependencies i.e. the problem of polymorphism in object-oriented paradigm will be considered in different publication.

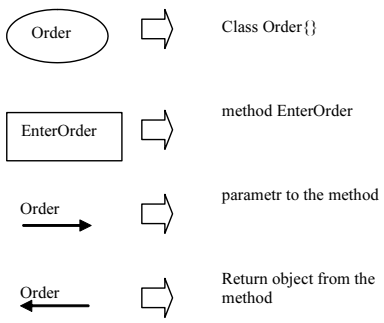

Fig. 2. Translation schema
Rys. 2. Schemat rzutowania do modelu klas

4. Procedural vs. OO

This pseudo code shows the difference between OO approach and procedural. The type of language for C is procedure oriented and for Java is object oriented. Class is equivalent to structure and method of the class to function outside the structure.

```
class Order                Struct Order
{                          {
float:price;               float:price
enterOrder(Staff);
}                          }
                           enterOrder(Order,Staff);
```

This is taken into account when switching between Java and C structure of program is needed.

5. Mapping to C language

Algorithm for marking is the same. Author suggests using the same marking for places, transitions in OO style. It will be the option of parser Java/C language generator to choose the implementation (time considerations is not taken into account in this article). The algorithm will treat the transitions connected with the palaces marked with initial class place tag as function with appropriate rule i.e. class is treated as structure as first parameter to function which is relevant to transitions in CPN model. Number of parameters to function and returning value the same treated as in mapping to Java algorithm 3 b).

Number of functions and structures goes with the same approach as in 3 c).

6. Prerequisite to building formal model in mapping to Java/C language

Crucial role of successful and accurate mapping is to analyze and mark CPN model taking into account object-oriented approach described in par. 3. To achieve this there is only needed marking with initial class tag for each functional area of the CPN model. This marking will not affect the model and the way it is simulated in Design/CPN simulator.

Rule 1: if the model is sophisticated and no transition is good for marking as the methods in OO approach than make a submodel using substitution transitions (CPN Hierarchy) and mark this transition with (* method@in *) tag.

7. Representation of the CPN model in xml

This is a fragment of representation of CPN model shown in par. 2 in xml [1, 2], taken from Design/CPN export functionality. Keywords marked bold are used in par. 8 to generate class model and procedural representation in C.

```
<workspaceElements>
<cpnet>
<generator tool="Design/CPN" version="4.0.5"/>
<page id="id75">
<trans id="id78">
  <name id="id123">
  <text>Enter Order (*method@in*)</text>
  </name>
        <cond id="id124">
        <text>
        [if (order = Big)  then staff = Expert else true]
        </text>
        </cond>
        <time id="id127"> …
        <text>@+5</text>
        </time>
</trans>…
<arc id="id135" orientation="ptot">…
        <placeend idref="id106"/>
        <transend idref="id78"/>
        …
        <annot id="id137">
        <text>order</text>
        </annot>
</arc>
<place id="id106">
        <!-- 3 regions(s) were found -->
        <!-- Region 1 of node -->
        <type id="id108">…
        <text>Order</text>
        <name id="id109">
        …
        <text>Order In</text>
        …
</place>
<aux id="id133">
        <text>
        (*InitialClassPlace@classneme:Order*)
        String id[];
        String name;
        </text>
</aux>
<aux-conn id="id162" orientation="nodir">
        …
        <node1 idref="id133"/>
        <node2 idref="id106"/>
        <!-- no text was found -->
</aux-conn>
…
</page>
</cpnet>
</workspaceElements>
```

8. Object model of class in Java of functional entity of CPN model

This class model represents container which holds information about classes and methods to be generated.

```
Class Class_{
        String name;
        String comment;
        Method methods[];
        int id_aux_cpn;
        int id_place_cpn
        };
Class Method{
        String in[];
        String name;
        String out;
```

```
        String comment;
};
```

Below is presented a list in Java language which holds all marked classes according to specification in par. 3 and 5.

```
List l=new Arraylist();
l.add(new Class_);
```

9. Extracting dependencies using DOM

DOM parser builds in memory the complete structure of the document to manipulate an XML document. Once the document is loaded, its data can be manipulated using the DOM. The DOM treats the XML document as a tree.

Below the pseudo code for extracting dependencies is presented:

```
Initialize list classes_list to keep object model of classes
Load xml document

For all elements of tag aux
begin
  for each element get value of node text which contains
"InitialClassPlace" annotation

    Take identifier of auxiliary region and find identifier of place
    from the auxiliary connection
    begin1

      find auxiliary connection in CPN model which points to a
      place that is a starting class container

      begin2
        find place tag for "InitialClassPlace"
        get class name
        initialize class_object type Class_

        begin3
          get method of class from transition connected to found
          place
          initialize class Method_object type Method

          begin4
            get parameters of method from places directed to
            found transition

            get return value for the method from place which is
            directed from transition to place

          end4
        end3
      end2
    add class_ object to classes_list
    end1
end

if classes_list is not empty and user marked 'to Java'
then
        print Java code
else if classes_list is not empty
        print C code
```

10. Code generator for Java/C

a) Java

```
class Order{
/*
String id[];
String name;
*/
OrderEntered enter_order(Staff staff,Order order){
/*[if (order = Big)  then staff = Expert else true]*/
}
}
class Staff{};
class OrderEntered{
Staff s;
Order o;
};
```

b) C language

```
struct Order{
/*
String id[];
String name;
*/
};
struct Staff{};
struct OrderEntered{};
OrderEntered enter_order(Staff staff,Order order);
OrderEntered enter_order(Staff staff,Order order){
/*[if (order = Big)  then staff = Expert else true]*/
}
```

11. Conclusions

In this article a technique for mapping concurrent systems modelled with Coloured Petri Nets to design and coding to object oriented design in Java and procedural approach with C language was proposed. Although this is starting point to consider detailed design and verification.

12. Future work

Considering transformation of CPN model to design without mapping tags will be next step of work. This should be considered deeper since it gives exaggerated code because common places to particular transitions would have the same methods.

Consider working on new syntax for denoting other dependencies i.e. the problem of polymorphism in object-oriented [6, 7] paradigm, time considerations, and validation information allowing checking model time constraints with implementation.

Consider generating representation to pseudo code, which would be parsed by back end parser generator for appropriate implementation.

13. Literatura

[1] Meta Software Corporation, Design/CPN tutorial, Cambridge

[2] Węgrzyn A.: Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci petriego, Oficyna wydawnicza Uniwersytetu Zielonogórskiego 2003

[3] Claude Girault, Rudiger Valk: Petri Nets for System Engeeniering, Springer 2002

[4] Jensen K., Kristensen L.M., Wells L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems, Department of Com puter Science University of Aarhus.

[5] Document Object Model, DOM parser http://www.w3.org/DOM/faq.html

[6] Gosling J., Joy B., Steele G., Bracha G. "The Java Language Specifcation", Second Edition. Addison-Wesley, 2000

[7] Agha G., De Cindio Eds F. Concurrent Object-Oriented Programming and Petri Nets. Lecture Note in Computer Science. Spring-Verlag, 1998.

[8] Puczyński M., Węgrzyn M., Implementacja hierarchicznej sieci Petriego w C++, XIII Krajowe Sympozjum Koła Zastosowań Cybernetycznych, WAT, Warszawa, 13.11.1997

[9] Bukowiec A., Węgrzyn A. Metody zintegrowanego projektowania sprzętu i oprogramowania z wykorzystaniem nowoczesnych układów programowalnych, XVII Krajowe Sympozjum Koła Zainteresowań Cybernetycznych, WAT, Warszawa, 08.11.2001, ss.7-12

Artykuł recenzowany