On the Object Oriented Petri Nets Model Transformation into Java Programming Language

Radek Kočí

Brno University of Technology, Faculty of Information Technology
Czech Republic
koci@fit.vut.cz

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

ICSEA 2024, 1.10.2024, Venice, Italy

Motivation

Simulation-Based Design

- reduce the gap between real needs and specified needs to sofware system under development
- combination of semi-formal and formal models
- formal and executable models showing a sketch of the system to help visualize what the system will do

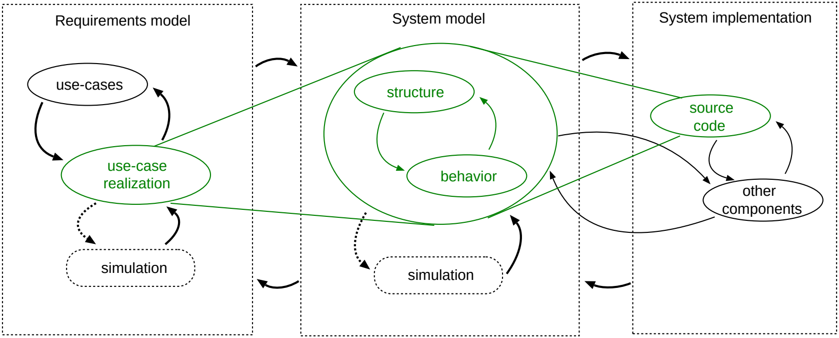
Model continuity

- elimination of the overhead caused by creating models at different level of abstraction
- continuous incremental development of models
- models can work in live system
- no need of implementation or code generation

Basic requirements for software design

Points that have to be met to create the correct and reliable software system

1. understand the goals of the software project and precisely specify the specific requirements whose implementation meets the declared objectives
2. validate that the requirements specification is in line with the objectives
3. based on a validated specification, create a system design that reflects the conditions of a particular implementation environment
4. verify that the system design complies with the requirements
5. implement the verified design
6. verify that implementation is consistent with the design
7. verify accuracy and reliability of implementation under real conditions



- design models complement and extend each other in the development process
- no need to transform or create new models
- if the nature of the resulting application permits, it is possible to maintain the models in the target system

How to meet points 5, 6, and 7 (implementation and verification)

- at the end, we have functional models that fully reflect the system requirements
- these models can serve as implementation models, i.e., become part of the target system
- if this is inappropriate or impossible (e.g., for performance reasons), we must implement or exploit the ability to generate code
- consistency with the design does not need to be checked, as the same set of models is still being developed

Why Petri nets?

Formalism of OOPN (Object-Oriented Petri Nets)

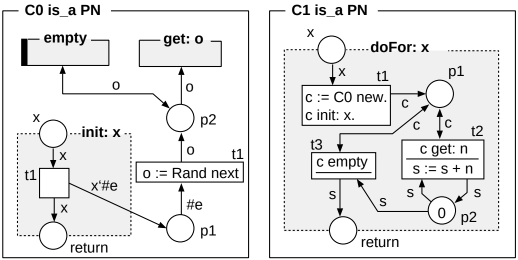
⇒ clear formal syntax

⇒ clear semantics

⇒ usable by developers having no power mathematical backgroud

Petri Nets Models

- we could use Petri nets in the same way with no need to get executable form
- Petri nets are also a simulation model
- Petri nets can be executed in real environment
- Petri nets are formal, can be transformed

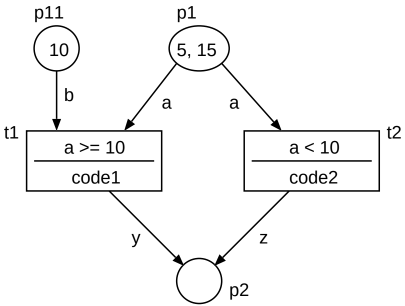


- classes, object nets, method nets
- nets: places and transitions
- a transition ≈ a component that can be instantiated (fired) multiple times for different variable bindings

Element Transformation

- OOPN class ≈ Java class
- object net ≈ constructor
- method net ≈ method
- place ≈ a special Java class
- transition ≈ an instance (component) of the special Java class

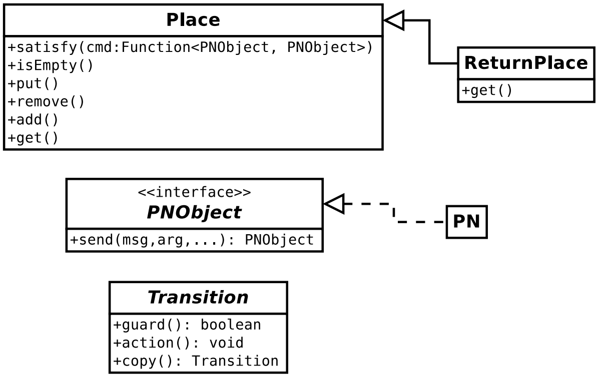
Example of the OOPN class C1:



```
public class C1 extends PN {
   protected Place p11;
   protected Place p1;
   protected Place p2;
   public C1() {
      p11 = new Place(this);
      p1 = new Place(this);
      p2 = new Place(this);

      class T_1 extends Transition { ... }
      T_1 t1 = new T_1();
      class T_2 extends Transition { ... }
      T_2 t1 = new T_2();

      t1.precond(p11, p1);
      t2.precond(p1);
      p1.add(5;
      p1.add(15);
      p11.add(10);
    }
}
```



OOPN is typeless

- the interface PNObject is the common type for all variables (objects)
- message passing is done specially (the method send)

```
class T_1 extends Transition {
    private PNObject a;
    private PNObject b;
    public boolean guard() {
        // guard1: a >= 10
        if (p11.isEmpty()) return false;
        if (p1.isEmpty()) return false;
        a = p1.satisfy((o) -> o.send(">=", 10));
        if (a == null) return false;
        b = p11.remove();
        p1.remove(a);
        return true;
    }
    public void action() {
        // code1: y = a + b
        PNObject y = a.send("+", b);
        p2.put(y);
    }
    public Transition copy() {
        T_1 t = new T_1();
        t.a = a;
        t.b = b;
        return t;
    }
}
```

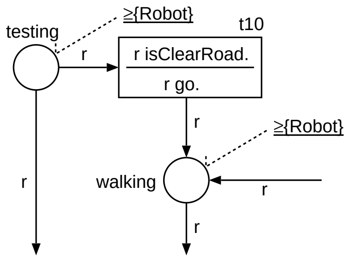
When an object is added to a place

- all connected transitions (components) are checked for fireability
- if the transition can be fired (its guard returns true)
  - a copy of the component is created
  - the copy is executed in the different thread (throught the executor)

```
void add(PNObject obj) {
  synchronized(monitor) {
    Integer c = content.get(obj);
    c = (c != null) ? c + 1 : 1;
    content.put(obj, c);
    for (Transition t : observers) {
      if (t.guard()) {
        Transition tt = t.copy();
        PNSystem.execute(() -> tt.action());
      }
    }
  }
}
```

Constraints

- constraints over OOPN allow, among others, to specify the type of objects in places
- it is possible to generate the code more precisely



```
class T_1 extends Transition {
    private PNObject r;
    public void action() {
        r.send("go");
        walking.put(r);
    }
    ...
}
```

```
class T_1 extends Transition {
    private Robot r;
    public void action() {
        r.go();
        walking.put(r);
    }
    ...
}
```

```
public PNObject m(PNObject p1) {
    ...
    Place ret = new ReturnPlace();
    ...
    // Transition::action -> ret.put(result);
    ...
    // Blocking method, waits for putting an object to the place
    return ret.get();
}
```

Conclusion

Present state

- We have implemented experimental simulator in Smalltalk (not suitable for wider use).
- We have partial experimental implementations of transformations into C++ and Java languages (done by our master students)

Future work (in progress)

- Completion of the simulator and editor implementation in Java.
- Completion of the code generation into Java including optimization (typing, atomic components not requiring threads, . . .).
- The goal is to create a comprehensive tool for modeling, designing, and verifying software systems with the possibility of direct deployment (with a lightweight version of the virtual machine for running models) or direct transformation into a programming language for more efficient running.

Thank you for your attention!