2004

TR-2004002: OOPN: An Object-Oriented Petri Nets and Its Integrated Development Environment

Jinzhong Niu

Jing Zou

Aihua Ren

OOPN: AN OBJECT-ORIENTED PETRI NETS AND ITS INTEGRATED DEVELOPMENT ENVIRONMENT

Jinzhong Niu, Jing Zou
Department of Computer Science
The Graduate Center, The City University of New York
365, 5th Avenue, New York, NY 10031, U.S.A.
{jniu,jzou}@gc.cuny.edu

Aihua Ren
Department of Computer Science and Engineering
Beijing University of Aeronautics and Astronautics
No. 37, Xueyuan Lu, Haidian, Beijing 100087, China
renah@buaa.edu.cn

ABSTRACT
This article proposes a modeling language for developing concurrent software, Object-Oriented Petri Nets (OOPN), which combines the maintainability and reusability of Object Orientation (OO) and the advantages of Petri Nets- a graphical interface and a sound theoretical background. By doing so, each benefits from the other and their negative aspects are at least effectively diminished. The implementation of the OOPN-Integrated Development Environment (OOPN-IDE) and a unique strategy for using OOPN are then explored. Potential applications for this modeling language and possibilities for future work are considered.

KEY WORDS
Petri Nets, Object Orientation, OOPN, Concurrent Software

1. Introduction

Petri Nets have been a popular technique for describing concurrent and parallel systems since their invention in 1962 [1]. They can be used to intuitively describe the dynamic properties of concurrent systems and the restriction for the allocation of system resources. Petri Nets also feature a graphical interface that is easy to represent and understand. Classical Petri Nets, however, are so simple that even the model of a small system needs a considerable number of states and transitions, resulting in a so-called "state explosion". People have been working to improve the representation capabilities of Petri Nets and have already introduced a variety of high-level Petri Nets and simplification methods, but the results are far from inspiring.

With the popularization of Object Oriented (OO) theory over the 1980s, the Petri Net community also appealed to OO concepts in that object orientation features abstraction, encapsulation, and inheritance, which may help to build layered Petri Net models instead of "flat" ones [2]. Until recently, a tremendous number of Petri Net modeling languages have emerged embodying more or less OO concepts, such as G-Net [3, 4], COOPN [5], LOOPN [6], PROT [7], OOCPN [8]. The efforts indeed helped to avoid state explosion to some extent; however, problems still remain. Some languages claim to be object-oriented simply by regarding token as objects, and some otherwise require profound mathematical knowledge.

What's more, only one way of achieving the reciprocity that Petri Nets benefit from was addressed and other aspects were simply neglected. Although OO has actually been the most popular programming paradigm, it is not a one-for-all solution for developing software, especially with respect to concurrent software. The following issues are closely related to the success of software projects [9, 10]:

- Software has become the dominant technology in many, if not most, technical systems, which makes software more and more complex [11]. Scaling-up of a software system is not merely a repetition of the same elements in larger sizes, but an increase in the number of different elements. In most cases, the elements interact with each other in some nonlinear fashion, and the complexity of the whole increases much more than linearly [12]. What makes the situation worse is that objects in the Internet era are more likely to run concurrently or in parallel. It is extremely difficult to accurately represent the communication and synchronization amongst them.

- Encapsulation is a major feature of OO, and only the public methods or attributes of an object are accessible to others; however, OO does not impose constraints upon the access, such as the order of several method invocations, which thus complicates the relationships amongst objects.

- According to OO theory, the state of an object is made up of the values of all its attributes. State transformations occur when any of such values changes; OO however does not provide facilities for representing the state transformations according to the inherent application logic.

- OO brings hierarchy and modularization, which increases the possibility for formal verification by

using a divide-and-conquer strategy, but no popular OO methodology provides such a mechanism.

Fortunately, the integration of Petri Nets and OO can diminish or totally avoid the above negative aspects of OO.

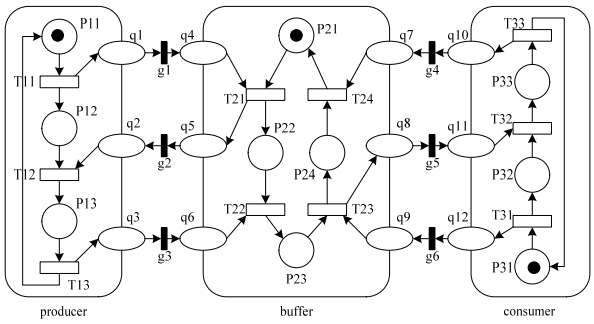
The remainder of this paper contains two major parts: Section 2 presents OOPN as well as how it is used to describe a concurrent system; Section 3 discusses the supporting tool, OOPN Integrated Development Environment (OOPN-IDE) and its implementation. The paper is summarized with a conclusion and prospects for future work.

2. OOPN

OOPN, originated from OPNets [13], features all major concepts of OO while remaining in a standard format. It does not introduce any new facilities so that all analysis methods or theorems regarding Petri Nets are still applicable.

2.1 OPNets

Since the specification of OPNets has already been presented in detail [13], here we will only introduce the modeling language briefly by giving an example. Figure 1 shows the model of the producer / consumer system.



Producer	Consumer
P11: idle	P31: idle
P12: waiting for production permission	P32: waiting for consumption permission
P13: producing	P33: consuming
T11: makes production request	T31: makes consumption request
T12: obtains production permission	T32: obtains consumption permission
T13: finishes production	T33: finishes consumption
Buffer	
P21: empty	T21: responds to production request
P22: waiting for production	T22: receives product
P23: full	T23: responds to consumption request
P24: waiting for consumption	T24: sends product

Figure 1: A producer/consumer model

Following OPNets, OOPN represents *classes* by predicate subnets wrapped in rounded rectangles embodying abstraction and encapsulation. A class may have multiple instances or objects, denoted as *tokens*, and the places where they stay show their current states. The external interface of a class is defined by elliptic *message queues* across the class boundary, which are similar to circular places inside the class. Message passing between objects is represented by connecting message queues with solid black rectangular *gates*.

The internal behavior of a class described by the predicate subnet is similar to that of classic Petri Nets except that at most one state can exist among the predecessors or successors of a transition, which thus prohibits the internal direct interaction between instances of one class. Message transmission is the only way for objects to communicate. Every transition is associated with pre-conditions and actions that are supposed to execute when it is fired. A transition is enabled to fire whenever there is a token in each of its predecessors and the pre-condition is satisfied.

A class may have attributes and each object of this type is associated with its own corresponding values. These values may be read by a transition in the pre-condition and written in the action. Messages transmitted between objects, denoted also as tokens, are structured data and can only reside in message queues. The messages which enable a transition to fire are destroyed after the firing actually occurs; on the other hand, a new message will be created in each successor message queue, whose content is determined by the action of the involved transition.

A class could be *simple* or *composite*. A simple class is defined by a subnet, whereas a composite one is in turn made up of several smaller classes. An OOPN system consists of multiple OOPN classes with instances specified that may pass messages to one another.

Figure 1 defines a producer, a consumer, and a buffer with the capacity of 1. At the beginning, the producer (consumer) makes a request by sending a message to the buffer through *g1* (*g6*). If the buffer is empty (full), it will respond by returning a permission message through *g2* (*g5*). Once the permission is available, the producer (consumer) may proceed to produce (consume). When the action finishes, it again notifies the buffer through *g3* (*g4*). The process may go in the same way periodically.

2.2 The Extensions to OPNets

OPNets were originally designed for modeling Flexible Manufacturing Systems (FMS). To meet the need for developing complex concurrent software, more mechanisms have yet to be supported. We obtained OOPN by extending OPNets as described in the following sections.

2.2.1 The Whole Life Cycle of Objects

OPNets disallow dynamic construction or destruction of objects during runtime so as to reduce the analysis complexity of FMS models, which however is not sound for generic software systems. OOPN supports the following two special transitions: *source* and *sink*. The former, as depicted in Figure 2a, has no predecessor and only one successor state; the latter, in Figure 2b, alternatively, has only one predecessor and no successor state. When the generator fires, a new object will be generated and put in the following state, whereas when the tomb fires, the involved object in the preceding state will be destroyed. In this way, objects could be dynamically generated or destroyed.

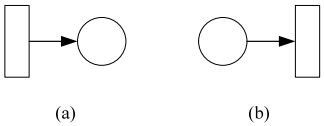


Figure 2: Source transition and sink transition

2.2.2 Inheritance

Inheritance is usually regarded as the unique characteristic that distinguishes OO [10], but OPNets provide no support for this mechanism. Based on the common practices in programming languages and the graphical structure of Petri Nets, we introduced single inheritance in OOPN.

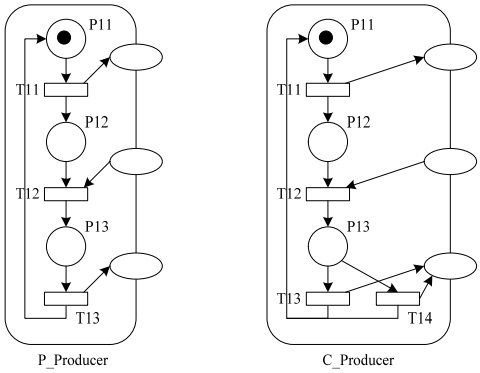


Figure 3: Inheritance in OOPN

Like various programming languages, where inherited methods are automatically available in a subclass, every OOPN subclass receives the Petri Nets structure of its superclass, and hence the internal behavior and external interface. Since every OOPN object is also regulated by meaningful sequential behavior, more detailed specification is necessary to define the semantics of inheritance rather than merely specifying a superclass. As shown in Figure 3, subclass *C_Producer* extends superclass *P_Producer* by introducing a new transition *T14* so that at state *P13*, *C_Producer* instances could randomly choose to do either of two kinds of production. In this case, besides the parent-child relationship, the connections amongst the involved states, transitions, and message queues should also be redefined.

2.2.3 Use Java for Semantics Definition

Although the OOPN graphical structure could model the skeleton of a real application very well, a scripting language is necessary to describe the detailed semantics. OOPN utilizes Java to define:

- Pre-conditions and actions of transitions and gates
- Class types for message tokens in message queues
- OOPN Class attributes, which could be variables of simple Java data types, e.g. *int*, or class objects, e.g. *java.util.Vector*

2.2.4 Inhibitory Arcs

Inhibitory arcs are introduced to obtain the capability of a so called *zero test* so that the firing of a given transition can be forbidden when its preceding place is not empty.

What's more, inhibitory arcs could only lead from a message queue to a transition in the OOPN rather than from a state, otherwise a transition may fire when no object is involved, which is against the object-centric view of object orientation.

3. OOPN-IDE

To support OOPN modeling, an integrated development environment, OOPN-IDE, has been implemented in Java, in which the specification of concurrent software can be designed, verified, and implemented. The three phases, in terms of OOPN, respectively refer to building up an OOPN model, dynamic and static verification upon the model, and generating Java code automatically from it.

3.1 Java Library OOPN Extension Package

The core of OOPN-IDE is the class architecture for implementing OOPN, which turns out to be the *Java library OOPN extension package*. The classes for OOPN components and their inheritance relations are depicted in Figure 4.

The *OOPNComponent* is the root class in the Petri Nets structure family and *Token* in the instance family. An *OOPNClass* object stands for an occurrence of an OOPN class, since a class may appear more than once in a system. *OOPNClass* uses hash tables to store the internal structure and external interface of an OOPN class. *OOPNNode* presents a node in classical Petri Nets, which could be connected by *Arc*. *OOPNClassInstance* and *Message* are designed respectively for OOPN class instances and message instances. All these classes feature

both graphical characteristics and Petri Nets semantics so that they can be used not only for editing models interactively but also for the execution of the real application system.

There are two different but related concepts of class:

- *OOPN model class*: or simply OOPN class, such as *producer* in Figure 1
- *OOPN Java class*: the Java implementation of OOPN model class

In OOPN-IDE, every OOPN class can be compiled into its corresponding OOPN Java class, which extends the OOPNClass.

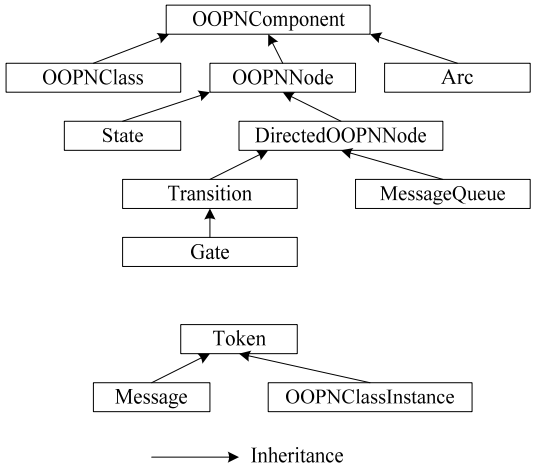


Figure 4: Java classes for OOPN model

For example, the following is part of source code generated automatically by OOPN-IDE for the producer class in Figure 1:

```
package opn.opnclass;
...
public class prod extends OOPNClass{
  public Message q3;
  ...

  void setupGraphStructure(){

    setClassName("producer");
    setShape(50,50,359,424,20,20);
    setColor(new Color(255,200,0));
    ...
    State s = null;
    addComponent(s = new State("P13"));
    s.setRadius(30);
    s.setColor(255,0,0);
    s.setLabelLocation("N");
    ...
    Transition t = null;
    addComponent(t=new Transition("T13"));
    t.setColor(0,0,255);
    ...
    addComponent(new MesQueue("q2"));
    ...
    Arc a = null;
    OPNNode b = null, e = null;
    addComponent(a = new Arc("T13---q3");
    b = getNode("T13");
    a.setPre(b);
    e = getNode("q3");
    a.setPost(e);
    b.addPostCom(a);
    e.addPreCom(a);
  }
}

public class prod_in extends OOPNClassInst{
  ...
  public boolean T13_preCond(){
    // NB: your code follows.
    return true;
  }
  public void T13_action(Message q3){
    // NB: your code follows.
  }
}
```

More details about Java library OOPN extension package and code generation are available in [9].

3.2 System Topology

As shown in Figure 5, OOPN-IDE adopts the client/server architecture, where the front end provides a graphical interface for interactive development, and the back end synchronizes multiple users' working on the same project. The distributed architecture is adopted based on the observation that concurrent and distributed systems are also likely to be developed concurrently by team members at different locations.

At the back end, access control and consistency control are imposed so that no invalid actions are requested for execution, e.g. a user is trying to add an arc to a node whereas the latter is just deleted by another user.

OOPN model classes and their corresponding Java classes, once verified and compiled, will be exported respectively to the *OOPN Class Library* and *OOPN Java Class Library*. Users may import existing model classes for inheritance or simply build up a composite class, and thus achieve reusability.

At the front end, two ways are available for users to build up OOPN models: graphical mode and text mode. Under text mode, a XML scripting language is used and the input should be checked for syntactical errors; under graphical mode, different graph components could be easily put together by drag-and-drop.

The network communication module wraps up the low-level communication details over the Internet and provides a concise API for both the front end and back end.

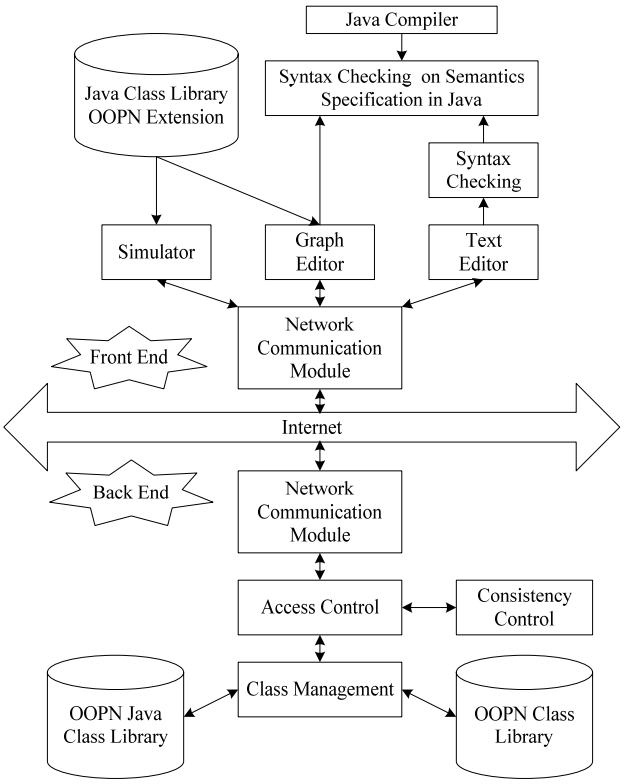


Figure 5: System topology

3.3 OOPN Model - The Real Application

A major difficulty in the traditional application of Petri Nets to concurrent software development is that there is a huge gap between the model and the target real system. In other words, even though a perfect model is built and thoroughly verified, errors could probably be introduced during the translation from the Petri Net language to a regular programming language. A unique strategy is hence used in OOPN-IDE.

Similar to the cases in which programmers use arrays, queues, or stacks in their programs, the net structure of OOPN model and the associated logic are explicitly kept in the real system generated by the OOPN-IDE. In such a way, an OOPN model is actually the prototype of the real application, which could be enriched incrementally until a full-fledged system is obtained.

The advantages of this method are obvious:

- No more brutal translation is needed and a run-able ongoing development version of the target system could always be available.

- The graphical interface of OOPN may present a vivid overview of the running system, thus debugging becomes much easier.

In OOPN-IDE, users may also choose not to retain graphical notations of the OOPN model in the target system to achieve better performance, although the OOPN logic still works.

4. Conclusions and Future Work

The OOPN modeling language presented in this paper concisely and intuitively conveys all major OO features while it still possesses the classical Petri Net structure. The advantages of OOPN in more detail are as follows:

- The representation power is increased while the advantages of Petri Nets, e.g. graphical structure and operability, still hold.

- The concept of OOPN class brings abstraction, encapsulation, and inheritance, which together enhance the reusability of OOPN models dramatically.

- Composite classes make it possible to build up a hierarchical OOPN system and thus a divide-and-conquer analysis method can be employed more or less avoiding state explosion.

The supporting tool of OOPN, OOPN-IDE, has been developed to model, simulate, and implement concurrent software systems in an integrated environment. Since OOPN models may be seamlessly embedded in the target system, simulation is actually equivalent to debugging or execution of the real system.

Future work about OOPN and OOPN-IDE could be done at the following three aspects:

- Interoperability
  Petri Nets can do a good job in representing certain tasks, operations, or certain services, but not some others, e.g. data storage. To achieve applicability to a variety of systems, Petri Nets facilities and non-Petri Nets components should be interoperable and be able to be integrated seamlessly. Component technology is a promising “glue” between them.

- Scalability
  Experience tells us that concurrent systems are very likely to be distributed. Although Petri Nets also embody distribution in the inherent locality of their transition firing rule, the following questions have yet to be answered:
  - How do we distribute different parts of a system over a collection of computers?
  - How and when are these parts initiated?
  - How do the components locate and communicate with one another?

- Friendliness
  Petri Nets provides an intuitive way to describe concurrent systems, however some people might still find it too formal and difficult to learn and understand. A user-friendly layer could be utilized to veil the underlying Petri Net model.

Upon achieving the above goals, the development of concurrent systems will be significantly facilitated.

Until now, OOPN and OOPN-IDE have been applied to the development of real-time operating systems, agent systems, grid computations, and cluster computations; and promising progress has already been made [14-17].

Acknowledgements

The research on object-oriented Petri Nets and the implementation of OOPN-IDE was sponsored by China Natural Science Fund.

Special thanks go to Professor Danny Kopec at Brooklyn College, CUNY, for all of his invaluable help with the preparation and organization of this paper.

References

[1] T. Murata, Petri Nets: Properties, Analysis and Applications, *Proc. of the IEEE, 77*(4), 1989, 541-580.

[2] Michael Zapf, Armin Heinzl, Techniques for Integrating Petri-Nets and Object-Oriented Concepts, *Working Papers in Information Systems*, University of Bayreuth, 1999.

[3] Yi Den, S. K. Chang, Jorge C. A. de Figueired, Angelo Perkusich, Integrating Software Engineering Methods and Petri Nets for the Specification and Prototyping of Complex Information Systems, *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, Chicago, USA, June 1993, 203-233.

[4] Yun Wu, Hanyu Yang, Aihua Ren, Wenlong Yang, The Development of Integration of OO and Petri Nets, *China Computer World*, Sep. 20th, 1995.

[5] Olivier Biberstein, Didier Buchs, An Object Oriented Specification Language based on Hierarchical Algebraic Petri Nets, *Proc. of ISCORE-1994*, Amsterdam, Holland, 1994, 47-62.

[6] C. D. Keen, C. A. Lakos, A Methodology for the Construction of Simulation Models Using Object-Oriented Petri Nets, *Proc. of the European Simulation Multi-conference*, 1993, 267-271.

[7] Sarah L Englist, *Colored Petri Nets for Object-Oriented Modeling*, Ph.D. Dissertation of University of Brighton, June 1993.

[8] Jianling Hu, *Theory and System Modeling of Object-oriented Colored Petri Nets*, Ph.D. Dissertation of Institute of Mathematics of Chinese Academy of Sciences, July 1996.

[9] Jinzhong Niu, *Research and Implementation of An OO Petri Nets Based Integrated Development Environment for Concurrent Software*, Master Thesis, Beijing University of Aeronautics and Astronautics, 1999.

[10] Xiyao Cai, Ping Chen, *Object Theory* (Xi'an, China: Xidian University Publishing House, 1995).

[11] Richard H. Thayer, Software System Engineering: A Tutorial, *IEEE Computer, 35*(1), April 2002, 68-73.

[12] Frederick P. Brooks, No Silver Bullet: Essence and Accidents of Software Engineering, *IEEE Computer, 20*(4), April 1987, 10-19.

[13] Yang Kyu Lee, Sung Joo Park, OPNets: An Object-Oriented High-Level Petri Net Model for Real-Time System Modeling, *Journal of Systems and Software, 20*(1), Jan. 1993, 69-86.

[14] Aihua Ren, Yuedong Du, A Multi-processor Real-time Operating System Based on Petri Nets, *China Journal of Software, 12*(7), 2001, 1064-1073.

[15] Aihua Ren, *A Research on OOPN based Concurrent Software Developing Methods*, Doctoral Thesis, Beijing University of Aeronautics and Astronautics, 2001.

[16] Yuedong Du, *Research On Object-Oriented Petri Nets-Based Real-time Software Development Method And Its Supporting Operating System*, Master Thesis, Beijing University of Aeronautics and Astronautics, 2001.

[17] Hui Jiao, *Research and Implement on Object-Oriented Petri Net-Based Distributed Computing and High Performance Computing*, Master Thesis, Beijing University of Aeronautics and Astronautics, 2002.

[18] Aihua Ren, Jinzhong Niu, Yongming Zhang, A Concurrent Software Modeling Method Based on Object-Oriented Petri Nets, *Journal of Beijing University of Aeronautics and Astronautics*, April 1998, 491-494.