Purposes of the PNtalk System

Radek Kočí *
koci@fit.vutbr.cz

Zdena Rábová*
rabova@fit.vutbr.cz

Abstract: The system PNtalk is a language and tool based on the object-oriented Petri nets intended for model a non-trivial concurrent and distributed software system. They have been developed by our research group at Brno University of Technology. PNtalk benefits from the features of Petri nets (formal nature, suggestive description of parallelism, theoretical background) as well as object-orientedness (abstraction, encapsulation, inheritance, polymorphism, and modularity). The goal of this paper is the reflection on the PNtalk system and on its design. It investigates PNtalk's ability to model, simulate, and prototyping with respecting the open environment allowing heterogeneous modeling (there can be combined more different modeling paradigms), distributed simulation, advanced simulation techniques, and self-modification of the models, simulations, or the system.

Keywords: object-oriented Petri nets, open systems, interoperability, Smalltalk, object behavior, metaobject

1 Introduction

The PNtalk is the project dealing with the modeling, simulation, verification, and prototyping of non-trivial software systems. It has been focused on object-oriented Petri nets (OOPNs) allowing a formal, hierarchy, and object-oriented description of modeled problem. Models described by OOPNs can be created and their behavior simulated by means of specialized tools, that we call the PNtalk system. There have been introduced some experimental implementations of such system and presented some ideas about modeling and simulation using OOPNs for several years. These years and results of the research bring experiences with OOPNs, new problems, and new challenges, namely prototyping, special simulation techniques (e.g. nested simulation), and the distributed environment. The PNtalk system was originally designed as a transparent layer on top of a Smalltalk system. The Smalltalk classes and objects are transparently interchangeable with classes and objects described by Petri nets. Up to now we considered PNtalk to be used for prototyping among others. This means that all models run in real time and communicate with real-world surroundings. In fact, we have also experimented with simulations a bit. Our current prototype has been built up as an open system which can combine different modeling and simulation paradigms of describing models and can also simultaneously support several independent simulations.

The paper is organized as follows. Firstly, we introduce important features of OOPNs and PNtalk system. Then we describe meta-level architecture of PNtalk, notice important parts of it and we outline their purposes. Finally, we summarize current research and its development for future.

* Faculty of Information Technology, TU Brno, Božetěchova 2, 612 66, Brno, Czech Republic

2 OOPNs and PNtalk

Before dealing with OOPNs and their reference to PNtalk, let we question about what is PNtalk. We can take a look at PNtalk from several points of view. PNtalk is the language for describing models based on OOPNs; it is the system for the simulation, analyze, and verification of those models; it can be the technology of an object-oriented design, model, and prototyping of the systems. Due to goals of this paper and topical research of our subgroup, the system for simulation and prototyping will take our attention in this paper. We try to describe the OOPNs and their features that are important for the PNtalk system architecture. Our description will reflect OOPNs defined in [3].

PNtalk and the associated OOPN formalism are characterized by a Smalltalk-like object orientation enriched by concurrency and polymorphic transition execution, which allow message sending, waiting for and accepting responses, creating new objects, and performing primitive computations. OOPNs are based on viewing objects as active servers[1] that offer reentrant services to other objects. Services provided by objects as well as independent activities of particular objects are described by means of high-level Petri nets consisting of inscribed places, transitions, and arcs.

When some object calls for some service of another object, videlicet the object (sender) sends a message to another object (receiver), new process is created in the receiver. This process can be understood the instance of the service description (in object-oriented terminology, the method is invocated). The process is characterized by the sequence of the transition executions, however the notion *process* can not necessary take notice like, for example, a Smalltalk process. We can suppose each transition contains the most one message sending. If some transition sends a message to another object, then it has to wait for response and the processing of this transition is suspended[2] (not the process). As it results from the Petri nets definition, each process can have several transitions that are independent each other, and consequently they can do concurrently. Thus, there is a parallelism within the OOPNs process. This is probably the most important feature of OOPNs from the system architecture point of view.

In next chapters we outline the definitions of the PNtalk system, its specificities, and its purposes in the simulations.

3 The Reflection on Object Behavior

We can say each object has some behavior. From the general point of view we can distinguish two kinds of the behavior: let's call them the domain behavior and the computational behavior. The domain behavior represents own behavior of object, thus operations (or methods) that are offered by the object. The computational behavior defines the form of message processing and it is usually hidden in the language environments (e.g. C++, Simula, Java). However, there can arise the need to change this one (q.v. chapter 4.3), thus the object would offer means for manipulation with the computational behavior.

In addition to many other things we ask for full controllable objects too. It means we would be able to start the object, to stop it, to store it, to restore it, and so on. Due to that request whole object is controlled by just the real Smalltalk process providing the operation processing.

[1] The active server means the object can have own activity which starts as soon as the object is created (it is called the *object net* in OOPNs).

[2] Take a notice the message sending is rather the request for message processing by the receiver and the receiver processes the message within a newly created process.

It follows the message passing between objects must be replaced by delegation and sequential processing is replaced by event processing with respecting two basic classes of events:

- *inside events*. They represent events arising inside the object, e.g. firable transitions etc.
- *outside events*. They represent receiving messages from the object's neigbourhood or from the inside of the object.

To ensure the features described above we define two special notions that define whole object behavior: the domain object and the metaobject. The domain object provides the domain behavior and the metaobject specifies the computational behavior as well as the control mechanisms of the object's life. The object of first type exists in programmer's world (and it need not be real) while the object of second type exists in the system's world. Simultaneously we have to classify the messages into two groups:

- *the domain messages*. They are messages defined by OOPNs classes. They are not implemented directly but they have the special invocation form enabling to simply control the object's computational behavior. They are always delegated.
- *the control messages*. They enable to control the object and to process the events. The control messages can be received either directly or by means of the special direct control message (thus, that received message is delegated). Every domain messages are accessible just via the special control messages.

In order to unsure the control mechanisms of the object so all control messages can not be delegated. Hence we define two types of messages in accornadce with the message processing:

- *direct messages* are processed in the classic message passing way. They ensure the basic controlling of the object. A part of the control messages are direct messages.
- *indirect messages* are processed by means of the delegation. They ensure the domain behavior of the object and partially the object controlling. The indirect message is received by means of the special direct message. All domain messages and a part of the control messages are indirect messages.

4 The System Architecture

Whole system is divided into two parts: so-called static and dynamic part. The static part consists of objects describing classes of OOPNs. The dynamic part consists of objects representing live OOPNs[3] objects (instances of OOPNs classes) and objects needed for simulation run. We will call all these object as metaobjects in order to we mark them off from OOPNs objects and classes that are represented by those metaobjects. All metaobjects are ordered in the system hierarchy. Each metaobject has its place in the system and there is strictly defined the communication protocol between appropriate components. Because of the static part is not interesting for our paper, we will not next deal with it. The system structure of the dynamic part is shown in the Figure 1. We can see three basic types of metaobjects there:

- `PNObject` representing the OOPNs object.
- `Environment` representing a group of OOPNs object collaborating in the simulation
- `Processor` representing the dynamic part of the system

[3] We suppose the system architecture will allow to abstract away from the OONPs formalism and to use the another formalism or its modification.

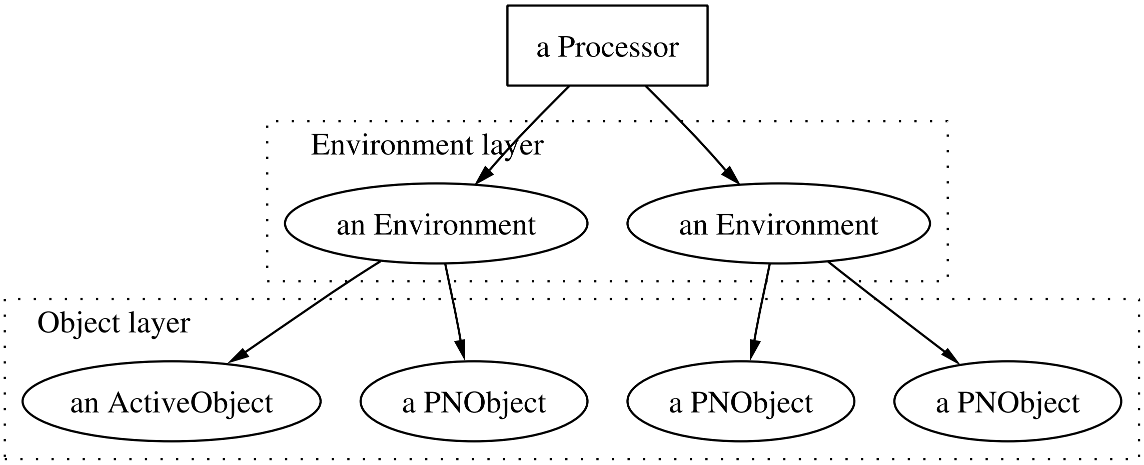

Fig. 1: The PNtalk architecture - basic overview

The interfaces of all basic metaobjects are alike, each metaobject can add new operation (e.g. the `PNObject` adds operations that are needed to receive the domain messages) or redefine present operations. Thus, three metaobjects described above have the same principle of control: just the controlling process, event processing and two types of messages with the difference the metaobject *does not* have the domain messages. All messages of metaobject are control messages and only if the metaobject represents some domain object then it must define the control message for receiving domain messages of the domain object.

The metaobject processes all inside events by step-by-step method. The object may do next step (thus process some *inside event*) only if the object receives appropriate control message (e.g. step). *Outside events* are processed without the necessity of the special control message[4]. Nevertheless there must be ensured synchronization of events and communication by means of delegation in good way.

4.1 The Delegation and Synchronization

As there has been already said, the classic sequential processing has been replaced by event processing and it is processed in just the real process. It also requires doing atomic events and synchronization of events (received message can not affect other events or they order). Hence the object processing is divided into two parts: *the life process* and *the message entry* (or *inside events processing* and *outside events processing*). Let us recall the message can be the control message or the domain message with receiving the domain message and the indirect control messages by means of the special direct control messages.

The message entry receives messages and ensures consistency of the message and the method[5] which has been invocated in virtue of the entry of the message. Because of the message is delegated, the sender can not wait for the message response. Consequently, the message unique identification is returned as the result of message delegation after message entry and the message is stored into the input queue. The real result is carried back to the sender via special control message[6].

[4] Only if we can delegate the control message, we have to send it via the special direct control message.
[5] Note the method can be real (by the control message) or virtual (by the domain message).
[6] After method finishing the metaobject gives notice to the sender about it and carries a method result together. The sender can process this event at any time with destroying invocated method *after* this processing.

The life process works in the cycles with the meaning the cycle represents the processing of just the event. Each cycle gets the message from input queue, takes information about type of message (control or domain), creates an event (or selects one of inside events) and processes it.

The metaobject is realized as a pair (*master*, *slave*). The master represents the metaobject from external point of view while the slave ensures the operations that are processed in the life process. The operations can be defined either for the control messages (then they are implemented as methods of the slave) or for the domain message (then they are represented by special system objects)[7]. Nevertheless, every operations do in so-called the method context allowing to splite the operations to more events[8] and consequently to process the operation in more steps.

The life process lives in context of the master receiving the direct control messages. The delegated messages are received by means of the special direct control message. The every operations offered by the slave are thus processed in virtue of the delegated messages. The majority of messages are delegated and processed in context of the life process. These messages wait for processing in the shared queue placed in the master. It implies the synchronization of delegated messages are ensured by the serialization by means of the queue. Each message can be processed in more steps (thus it can be defined as a sequence of events), but there can be processed just the one event for the time step.

4.2 The Metaprotocol

The metaprotocol is very important component defining the control operations on the object. It allows the object manipulating together with the concept of delegation and serialization. Apart from other things, it offers special control messages allowing to interfere in object's state or object's work. It follows that we can save the state of live object including semi-finished messages and sequentially we can restore this object and its state in simple way. It also allows the object migration or, in more exactly words, it is one of the necessary features for migration.

4.3 The Interoperability

The present architecture is limited to itself, videlicet it has own special computational behavior which is not compatible with other systems. Nevertheless, there will arise the situations when some object (sender) living outside the system can communicate with the object (receiver) living inside the system but the sender does not need know about it[9] or the object can send a control message without necessity to know implementation details. It follows the computational behavior should not be visible at domain level but the system should be to adapt this behavior between any objects. The system lacks a means for transparent communication (the interoperability) between objects of different types, and for communication with objects on different levels (the domain level or the meta level). Hence we introduce new special layer enabling interoperability and special access to objects – so-called proxy-objects.

Using proxy is standard Smalltalk technique to control message passing. A proxy is obviously implemented in such a way that it handles the exception *messageNotUnderstood*. The handler (implemented in the proxy) decides how to react to the message. The proxy-object in

[7] However, both types of operations are always evoked by means of the slave.

[8] The event is undestood either the processing of the method of the slave (for control message) or e.g. the firing of the transition (for OOPNs).

[9] According to polymorphism the object can communicate with any objects (receivers) by the same form and the receivers have to understand this form.

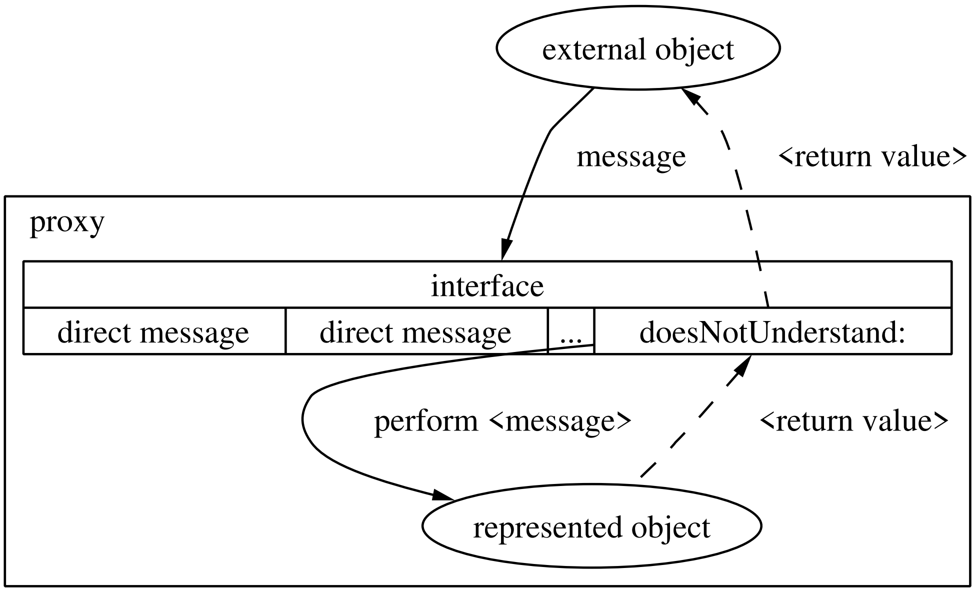

Fig. 2: The Proxy object - basic overview

the PNtalk architecture has a few significant properties, mainly to ensure uniform communications between objects of different types or in a different levels without the need of message sending modifications.

The basic overview of proxy using is shown in the Figure 2. An object (potential sender of a message) always refers to another object (potential receiver) by means of a proxy-object devolving incoming messages on the receiver in the suitable form. There are several kinds of proxy-objects allowing the communication in the domain level or in the meta (control) level. The behavior of the proxy-object also depends on a kind of the sender and the receiver. The proxy-object does not define the computational behavior of the receiver, but it ensures message receiving by the receiver in the right way and it conforms the response to sent messages in accordance with the requirements of the sender. Further, it can serve like a reference to far object in distributed environment too.

The present concept of object controlling and proxy-objects together make the basis of full-distributed work (i.e. collaborating of far objects including the possibility of object migration). Nevertheless, there are still several open questions for future research, such as own transmit, stopping message passing during the transmit, and so on.

4.4 The Environments

The notion *environment* is very useful concept allowing us to split up whole live system into independent parts. These parts can be understood as simulations[10]. In other words, the environment defines a life-space of a set of objects. It controls cooperation of objects in the environments. Furthermore it offers a notion of time. Thus, it allows grouping the objects to the relative independent units. Every environment has its own scheduler. In many cases it is obvious to interchange the notions environment and scheduler mutually. The priciple of the environment scheduler (thus the control of objects placed in the environment) is outlined by the algorithm 1. Objects can migrate between environments and can communicate with objects of other envi-

[10] The simulation is mainly characterized by the time axis. The independent simulations thus mean simulation runs with different time axes.

Algorithm 1 The object controlling by the environment

1. $E = \{o \,|\, o\, is\, an\, object\, participating\, the\, simulation\}$
2. Q is the queue of planned events in the time
3. T is the simulation time
4. $\forall\, o \in E$: the mesage step is send to o
5. if $\forall\, o \in E$ the possibility of step does not exist
   (a) Q is not empty $\Rightarrow T = q : q$ is a next planned time from Q
6. The steps 4 and 5 are being done in cycles

ronments. We can view the environment as a means determining the simulation or a part of the simulation.

5 Conclusions

In the paper, we have briefly presented the PNtalk system. PNtalk is the open simulation and development environment based on object-oriented Petri nets above all. Nevertheless, our approach brings a possibility to modify this formalism, to chage it to another one, or to adapt the formalism for the needs of experimental aims or real aplications. These changes can be done incrementaly, i.e. for choosen parts of the models. Secondly, the presented system architecture allows to create more independent simulations, to migrate objects to another simulation, and to communicate between far objects. Thirdly, the system ensures an interoperability with Smalltalk, consequently the model can consists of objects placed in the PNtalk as well as in the Smalltalk. It allows to create the simulation and verification model with simple changing to prototype. To summarize, PNtalk makes an open framework for simulation, prototyping, and verification of the heterogenous, distributed systems or, if it need be, the choosen parts of the systems. Although the current design and implementation do not allow to use the system in full extent till now, the open architecture of the system ensures very flexible development in this area for future research.

This work has been done within the project CEZ:J22/98: 262200012 "Research in Information and Control Systems" and also supported by the Grant Agency of Czech Republic grants No. 102/01/1485 "Environment for Development, Modelling, and Application of Heterogeneous Systems".

Bibliography

1. M. Češka, V. Janoušek, and T. Vojnar: *PNtalk – A Computerized Tool for Object-Oriented Petri Nets Modelling*. In F. Pichler and R. Moreno-Díaz, editors, *Computer Aided Systems Theory and Technology – EUROCAST'97*, volume 1333 of *Lecture Notes in Computer Science*, pages 591–610, Las Palmas de Gran Canaria, Spain, February 1997. Springer-Verlag.
2. V. Janoušek: *Reflective Approach to Petri Net Simulation.* MOSIS'97, MARQ Ostrava, pages 209–304.
3. V. Janoušek: *Modelování objektů Petriho sítěmi (in czech).* Ph.D. thesis at DCSE FEECS TU of Brno, 1998.
4. R. Kočí: *Rozšiřitelný simulátor objektově orientovaných Petriho sítí (in czech).* Thesis at DCSE FEECS TU of Brno, 2000.

5. R. Kočí, T. Vojnar: *A PNtalk-based Model of a Cooperative Editor*. In Proceedings of the 35th Spring International Conference on Modelling and Simulation of Systems – MOSIS 2001, Hradec nad Moravici, Czech Republic, CZ, MARQ, 2001, p. 165-172, ISBN 80-85988-57-7
6. R. Kočí: *The PNtalk System - a Technique for Object Oriented Modelling*. In: Proceedings of XXIIIrd International Autumn Colloquium, Ostrava, CZ, MARQ, 2001, p. 151-158, ISBN 80-85988-61-5
7. E. Kindler et al.: *Special Approach to Reflective Simulation*. In: Proceedings of MOSIS 2001. MARQ. 2001.
8. R. Kočí, Z. Rábová: *The PNtalk System and Interoperability*. In: Proceedings of International Conference MOSIS '02, Ostrava, CZ, MARQ, 2002, p. 73-80, ISBN 80-85988-71-2
9. V. Janoušek, R. Kočí: *PNtalk - An Open System for Prototyping and Simulation*. In: Proceedings of The 28th ASU Conference, Brno, CZ, FIT VUT, 2002, p. 133-146, ISSN 1102-593X