THE OPEN ARCHITECTURE OF THE PNTALK SYSTEM

Radek KOČÍ, Doctoral Degreee Programme (3)
Dept. of Inteligent Systems, FIT, BUT
E-mail: koci@fit.vutbr.cz

Supervised by: Dr. Zdena Rábová

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 as well as object-orientedness. This paper deals with the open architecture of the PNtalk system and it investigates PNtalk's ability to control all aspects of the object life.

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 [1]. 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. 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 system is approached as a transparent layer on a top of the Smalltalk system.

The paper is organized as follows. Firstly, we think about the object behavior. 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.

2 THE 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 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, 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. 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* (events arising inside the object, e.g. a sequence of commands)
- *outside events* (receiving messages)

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. 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 ensure 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.

3 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[1] 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. It is very useful concept allowing us to split up whole live system into independent parts[2]. The environment defines a life-space of a set of objects and controls these objects.
- `Processor` representing the dynamic part of the system.

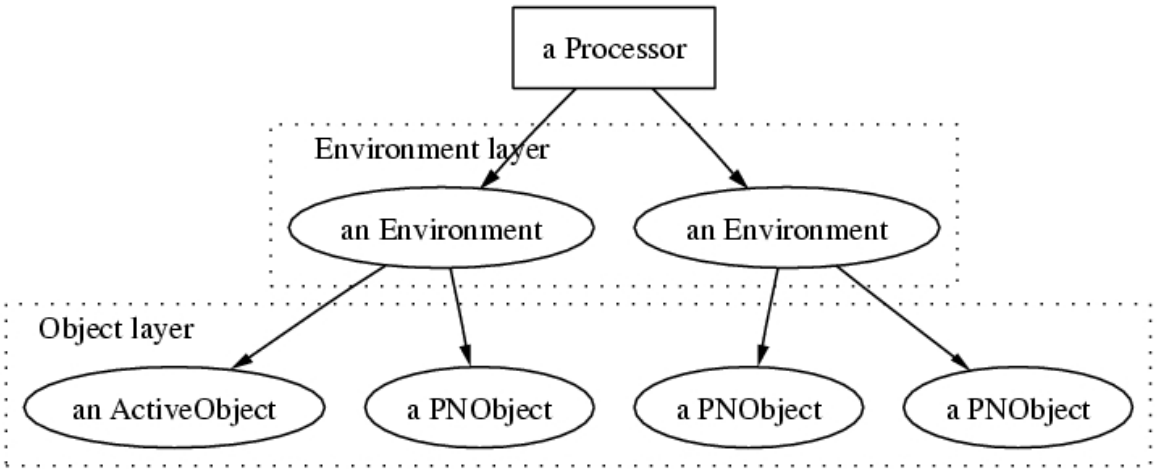

Figure 1: The PNtalk architecture - basic overview

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.

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

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

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`).

4 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 message entry (*outside events processing*) receives messages and ensures consistency of the message and the method 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 – 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 (*inside events processing*) 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). Nevertheless, every operations do in so-called the method context allowing to splite the operations to more events 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. Each indirect 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.

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.

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 and to use special simulation techniques. Thirdly, 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. To summarize, PNtalk is prepared for an open framework for modeling, simulation, and prototyping of the heterogenous, distributed systems or, if it need be, the choosen parts of the systems.

ACKNOWLEDGEMENTS

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”.

REFERENCES

[1] V. Janoušek: *Modelování objektů Petriho sítěmi (in czech)*. Ph.D. thesis at DCSE FEECS TU of Brno, 1998.

[2] 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

[3] 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

[4] E. Kindler et al.: *Special Approach to Reflective Simulation*. In: Proceedings of MOSIS 2001. MARQ. 2001.

[5] 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

[6] 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