The PNtalk System – a Technique for Object Oriented Modelling

Radek Kočí *
koci@dcse.fee.vutbr.cz

Abstract: The system PNtalk is a language and tool based on the object-oriented Petri nets intended for model a non-trivial distributed software system. The paper presents techniques for object oriented modelling, the system PNtalk and usefulness of this system with applying it to model and creating prototypes of software systems. We consequently introduce the system PNtalk as a possibly technique for object oriented modelling.

Keywords: object-oriented modelling, object-oriented Petri nets, UML

1 Introduction

The PNtalk system [1] is a language and tool based on object-oriented Petri nets (OOPNs) that have been developed at DCSE TU Brno in order to support modelling and prototyping complex concurrent and distributed software system. PNtalk includes the features of Petri nets (formal definition, analyze, native parallelism) and object-oriented design (encapsulation, inheritance, polymorphism, and so on). Thus, the PNtalk system allows for intuitive modelling the key features of such systems, which include message sending parallelism and synchronisation.

Models described by PNtalk can be created and their behaviour simulated by means of a specialised tool [1], which we call the PNtalk system. Currently, we are working on a new version of the PNtalk system, which should allow us to run OOPN-based prototypes in a truly distributed way. Moreover, we have already done some work towards a state spaces-based formal analysis and verification of systems modelled by OOPNs.

Object-oriented modelling generally consists of two parts: modelling the static structure and modelling the dynamic aspects of a system. There are many techniques and tools for these kinds of modelling, the most familiar technique is UML (Unified Modelling Language). UML is a set of techniques (diagrams) describing the modelled system's particular parts. *Class diagrams* are the main technique to describe the structure of the system, the dynamic aspects are modeled with techniques like *Sequence Diagrams*, *State Diagrams* and so on [5]. While class diagrams are well understood, the techniques for dynamic aspects have any disadvantages. The main ones are these: UML lacks a formal semantics and there are not any complex methodology, which would say how to use these technologies. Although thanks to UML we can create a a complex description of the system (somewhile even too complex), UML cannot describe some aspects.

This paper presents two technologies. First of them is UML which has been created to an object-oriented analysis and design of systems. The second technology is the PNtalk system which only aspires to given area. We will deal with a question what PNtalk can offer, compared with UML.

* Dept. of Computer Science and Engineering, TU Brno, Božetěchova 2, 612 66, Brno, Czech Republic

The paper is structured as follows. We first provide an introduction into object-oriented modelling, UML and using UML to a development process. Secondly, we focus on the PNtalk system and its meaning in a development process. We also achieve comparing PNtalk with UML. Subsequently, we show using these technologies by means of an example. We execute an UML modelling of the part of the PNtalk system and develop the corresponding PNtalk-based model, which captures the dynamic behaviour of this system. We finish the paper by some concluding remarks about a development of the PNtalk system.

2 Object-oriented Modelling with UML

A system based on functions, that the system has to provide, is unstable in view of changing user's requierements, while the system based on objects is more resistant. That is what a history and experience say. Hence a development of information systems has aimed at an object-oriented analyze, design and implementation. In the 90th years, there have been many methodologies for object-oriented analysis and design. Each one provides different methods and tools and defines own process of creation of information systems. This state have not been sufficient yet, because it doesn't allow communication between any designers, programmers and other privies to development process in a good way.

In 1995, there was introduced UML (Unified Modelling Language) on the OOPSLA conference. UML arosed as a compilation of two of the most used methodologies – Booch and OMT. UML is not a compact methodology, but it means graphical creation of object models in process of creation of information systems. In this time, UML is a popular object-oriented notation and it could serve as a standard for analytics, designers and programmers, that use the object-oriented technologies. As we have already said, UML is a set of graphical tools, that are used in a development process of information systems. Thus, it isn't a methodology, however, if we want to effective use these tools, we have to have a methodology, which defines *when*, *who* and *how* use individual tools. In this paper, we will not deal with these technologies, we only briefly describe several tools and their applicability in the development process.

First, we take a look at the meaning of a notion *view*. A view of a system is a projection of the system on one of its relevant aspects. Such a projection focuses on certain aspects and ignores others. Therefore it is useful to have different views of a system. UML uses multiple notations (tools) for exhibiting different *views* of the system. We can distinguish four main views:

- The *structural view* describes layout between objects and classes, their associations and their possible communication channels.
- The *behavioral view* describes, how the system components interact, and characterizes the response to external system operations.
- The *data view* describes the state of the system units (objects) as well as their relationships.
- The *interface view* focuses on the encapsulation of system parts, and the possible usage from outside.

UML currently has as many as eleven different notations, which constitute different views of UML designs. In the following, we briefly discuss several notations and their usage to describe certain aspects.

Use case diagrams display the relationship among actors and use cases. A use case is a set of scenarios describing an interaction between a user and a system. The two main components of a use case diagram are use cases and actors. Use case diagrams deal with an interface and behavioral view at the border of the system.

Sequence diagrams can be used to demonstrate the interaction of objects in a use case. They generally show the sequence of events that occur. Sequence diagrams therefore clearly define behavioral aspects but are based on structural and interface views. They do not describe an internal behavioral of objects.

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. They are the central notion for structural aspects.

State Diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

Activity Diagrams are defined as a special case of a state diagrams. They could be useful for describing internal processing of operations or use cases. Activity diagram models a dynamic flow controled by internal stimulations.

Own phraseology of UML does not have enough power in order to it allows us to realize fundamental relationships and, in particular, rules, that are branch of every modelled system. For example, how way do we may define a condition in UML that leastwise one item have to be at a stack close to invocation an operation *pop*? If we do a bit of thinking we will find that for this case (and analogous cases) UML does not offer suitable tools. There has often been used a text comments there, but it steps down formal level of a model and there is a question whether it contribute to better roundness.

Partition of developmental process on two main phases – an analytical and design phase and own implementation phase – is matter of additional problems. Because the models, that are obtained from an analytical phase, are not explicit and complex (and they cannot even be), there has to be a teamwork between analytical and programming parts. However, a experience proclaims the thing, that there are very small tendencies to a teamwork.

In this time, there are tendencies to solve these problems. One of possible ways is a usage of Petri nets, respectively object-oriented Petri nets (OOPNs). In next chapter we will pay attention to the PNtalk system, which is based on OOPNs.

3 Object-oriented Modelling with PNtalk

In this chapter, we demonstrate using PNtalk to model software systems. In next sub-chapters, we give account of fundamental notions, that are very often used. Next, we will briefly pay attention to the PNtalk system and then we will show, how we can use the PNtalk system to model a part of a system.

3.1 What a methodology is

In our paper, we often work, in reference to object-oriented modelling (OOM), with three notations – *a tool*, *a technique* and *a methodology*. Let us put a question now – what do the notations mean and what difference is there between them? A OOM technology is either an integrated set of tools or a single tool to support development process of information systems, while the methodology defines a usage of some technology. In other words, the method consists, apart from the notation themselves, of a set of rules and steps through the *design process*. The design process may be generally defined to be the sequence of specifications developed during the lifetime of the software system. The design process for large systems is very complex and thus the methodological rules deal with many different aspects and activities during design. It follows two esential features for a methodology – it have to be clearly understandable and preferably simplex. Let us remember that the methodology will be used by a man which will try to join it with his/her view of a development process.

3.2 The PNtalk system

We attempt briefly draw near the PNtalk system. 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 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. A transition can have a guard restricting its firability and an action to be performed whenever the transition is fired. Finally, arcs are inscribed by multiset expressions specifying multisets of tokens to be moved from/to certain places by the arcs associated to a transition being fired.

A model in PNtalk is built up of Petri nets organized in classes. Every class consists of an object net modelling the autonomous internal activity of objects of this class and a set of method nets (including constructors) describing how these objects asynchronously respond to certain messages. All method nets of a class share access to the appropriate object net, which is implemented via allowing places of the object net to be manipulated by transitions of the method nets.

A running PNtalk OOPN consists of a set of objects, which are interconnected by references bound to tokens. Using these references, the objects can communicate by means of message sending, either asynchronously via method nets or synchronously via synchronous ports (or primitive non-Petri net-based functions).

3.3 PNtalk as a technique for modelling

As we can see in the previous chapter, there are a number of techniques to model dynamic aspects of a system. If these techniques are used, the designer staticly describes a behaviour of a system in a design phase and he cannot make certain of his partial ideas about the system behaviour. Next, there can be a problem with understanding to models (diagrams), which arise from graphical whatness of these notions. We could describe this situation by following words: “The nice thing about graphical description techniques is that everyone understands them, the bad thing is that everyone understands them in a different way." Therefore, the given notions lack a formal foundation, which would make possible understand the model in a only way and also make possible analysis and verifications of the system.

Now, let's take a look at using the PNtalk system to model a structure and a behaviour of the systems. Because the PNtalk system is not only a language for describing the model, but it is also a tool for simulation, it combines a static creation of a system model with an execution of this model. If the model is executed, there is easily an effectible retroaction to change the model, when knowledges getting from a simulation of the model make necessary this change. The PNtalk language is based on the OOPNs, that have a formal foundation [6]. PNtalk can particularly join design and implementation phases. It deals with a behavioral and data view on the system. There is created a model in a PNtalk environment, and this model can concurrently be simply transformed into a functional prototype.

3.4 A PNtalk-based Model

We have chosen a part of the PNtalk system as an example of a usage the PNtalk system in developing process. First, we use UML to model a choice part, then we transform these models (diagrams) into PNtalk-based model and finally we discuss ability PNtalk to be a technique for object-oriented modelling. We will model a PNtalk processor, which is a central part of the PNtalk system and execute a PNtalk-based model. We will keep an experimental implementation from 2000 [3] in view. The PNtalk model specifies an OOPNs. The OOPNs is a set of classes of objects; objects are instances of classes. These classes are translated into *an internal representation* of a PNtalk processor. An internal representation consists of objects corresponding to classes defined by a model. These objects are derived from special classes `PNClass` and `PNSuperClass`, as we can see at figure 1a). This shows a class diagram of given state.

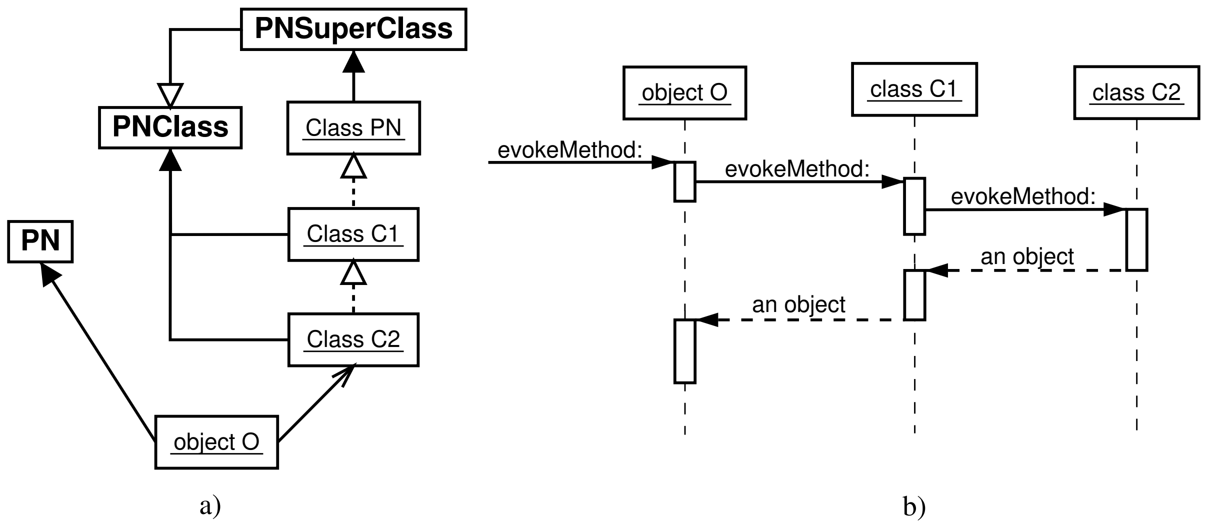

Fig. 1: An OOPN representing a simple system with restartable counters

This diagram is slightly adapted, because it ilustrates relationships between PNtalk processor classes along with an inheritance of OOPNs classes. An arrow-head with a hollow triangle stands for an inheritance of processor classes, the dotted one stands for inheritance of OOPNs classes that are represented by objects in the processor. A filled arrow-head stands for a relationship of an object to its class. Objects whose name starts with *Class* represent classes of an OOPNs model. An object named *objectO* represents an instance of OOPNs class *C2* during a simulation of OOPNs model. An OOPNs class *Class PN*, derived from class `PNSuperClass`, is a special object, which always stays at the top of an inheritance hierarchy of OOPNs classes. Thus, the figure shows one objects (it is derived from a class `PN`), two OOPNs classes (they are derived from a class `PNClass`) and one OOPNs class which is derived from class `PNSuperClass`.

We depict functionality by a part of the PNtalk processor – calling methods of OOPNs objects. Because classes of a PNtalk-based model are translated into internal-representation objects in implementation environment, there cannot be applied natural inheritance of given environment there. On that account there must be implemented own inheritance hierarchy direction. Let us specify a letter `O` as an OOPNs object, which is derived from a OOPNs class `C`. Remember these two elements are objects in the PNtalk processor internal-representation. Now, if we attempt calling an OOPNs method `M` of an object `O`, the object `O` devolve this requirement upon a class `C`. The class `C` look for an object, which represents a called method `M`. If this object have been found, the class `C` give back one; if does not, the class `C` generates a fault. This have been a simple variant without an inheritance. As soon as we add an inheritance, the scenario of an unsuccessful search is changed. The class attempt devolve a requirement upon a superior class (if exists), which process this requirement in the same way like there have been already said. Only if a top class in inheritance hierarchy does not find a called method, there is generated a fault.

If we take a look at the figure 1b, we can see a sequence diagram of an object behaviour based on external events. A filled arrow-head stands for an invocated event, the dashed one stand for a returned result of event processing.

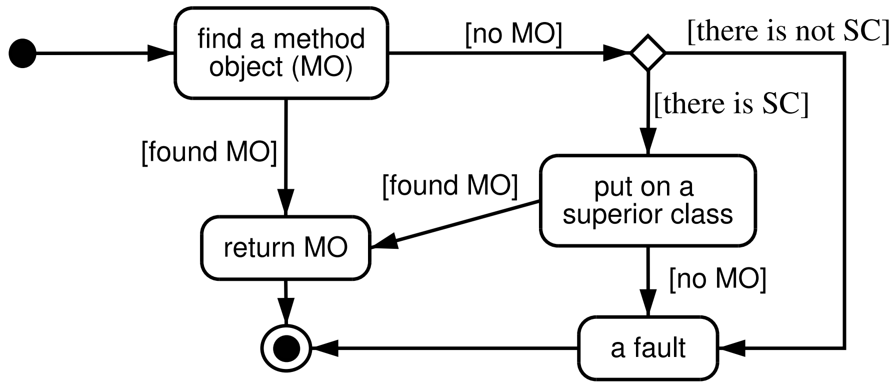

Fig. 2: An activity diagram of a method invocation

Now we focus on an internal behaviour of an object representing a PNtalk class close to a method invocation. This is depicted at a figure 2 subscribing with a description which has been already depicted there. The letters `SC` at the picture means a shortcut of *a super class*.

We have dealed with UML modelling of our chosen part of a system so far. Now we pay attention to PNtalk-based modelling. We have attempt to create a PNtalk-based model according to previous diagrams. Thus, our model consists of two classes – `PNSuperClass`

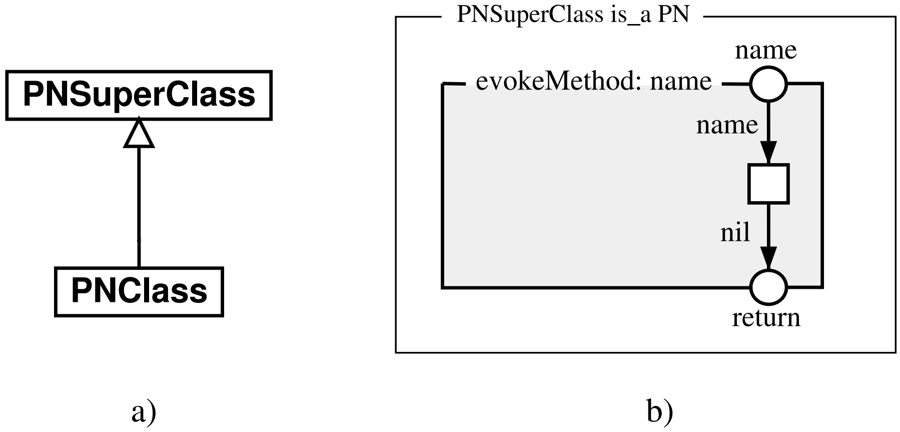

Fig. 3: a) A class diagram; b) OOPNs representing a class PNSuperClass of the PNtalk system

and `PNClass` (see a fig. 3a). Both classes have a method *evokeMethod:* with one parameter (a name of a method). This method process a requierement to a method invocation of OOPNs object (and its relevant OOPNs class). If a looking for an object representing a called method is unsuccessful, there is not generated a fault but the relevant method *evokeMethod:* gives back a special object `nil`. The class `PNSuperClass` represents a top class of an OOPNs inheritance hierarchy (see a fig. 3b) of which other OOPNs classes are derived. It is not a part of an OOPNs model, hence it always gives back an object `nil` close to a method invocation. The class `PNClass` (see fig. 4) represents a PNtalk class which is a part of an OOPNs model, hence it can contain methods. If any method is invocated, the method *evokeMethod:* of a relevant class `PNClass` is called. The method is processed as follows. First, it have a go at look for a method object (a transition `t1`). If this search is successful, the method give back a found object (a transition `t2`). In other case, the method is calling the same method in a superior class (a transition `t3`). If the superior class is derived from class `PNSuperClass`, the search of a method object is unsuccessful (its method *evokeMethod:* give back an object `nil`).

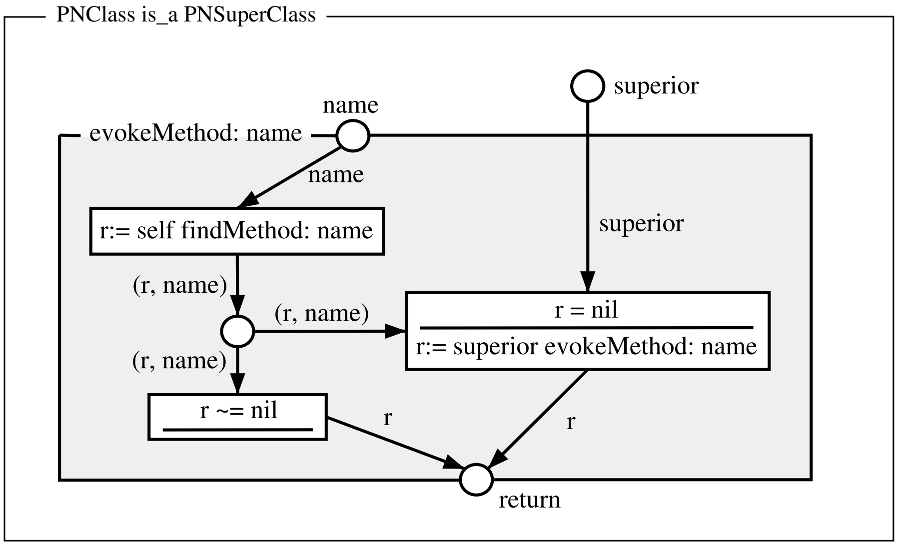

Fig. 4: OOPNs representing a class PNClass of the PNtalk system

Let's think about a development process as a sequence of an analysis, design, prototype creation, and finally implementation of a system. PNtalk presses for consolidation of a design phase and prototyping. As in every similar tools the most problem at its usage is by estimation to abstraction level. If the model contains very complicated objects (classes), their behaviour will be not providing an easy survay. On the other hand, very simply objects result in great numbers of them in the model. Primarily there have to be found essential objects of a modelled system and their relationships. Further to we can successfuly employ resources of UML such us *Use Case*, *Sequence*, *Class* diagrams. Because the PNtalk system could be taken in a design phase and the model created by PNtalk can serve as a prototype, there can be partially eliminated some problems, which rise from an insufficient teamwork between a design and an implementation phase.

4 Conclusions

In this paper we have briefly presented UML [5] and the PNtalk system [1]. We also have dealed with object-oriented modelling of software systems based on these technologies. Our goal have been to show possibilities of the PNtalk system to be an object-oriented technique for a design and a prototypes creation of software systems. Althrough the PNtalk system cannot fully comprehend a development process of systems, it can partial bridge over a design and implementation phase. A model described by PNtalk is good understandable and have a formal foundation, which make possible an analysis and verification of this model. It also can serve as a functional prototype, some parts of the system are likely to be implemented in other languages than only in PNtalk (e.g. a user interface can be implemented in Smalltalk). Thus, our goal is to take PNtalk in object-oriented techniques (such as UML) that are intended for a development of systems.

In the future, if we can use the PNtalk system for a design and the prototyping of systems in a good way, we have to accomplish these points: (a) to have a full-functional version of PNtalk; (b) to support a distribution environment; (c) to join the PNtalk system with another environmets (such as Smalltalk, CORBA and so on) and mainly (d) to design a methodology for PNtalk-based modelling.

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. R. Kočí. *Rozšiřitelný simulátor objektově orientovaných Petriho sítí (in czech).* Thesis at DCSE FEECS TU of Brno, 2000.
4. R. Kočí. *Prospects of the PNtalk system in the Smalltalk environment.* In: Proceedings of 7th Conference Student FEI 2001, TU of Brno Brno, 2001, pages 338–342.
5. The Rational e-development company. Unified Modelling Language resource center. *UML documentations.* URL: `http://www.rational.com/uml/`
6. V. Janoušek. *M*odelování objektů Petriho sítěmi (in czech). Ph.D. thesis at DCSE FEECS TU of Brno, 1998.