TOWARDS AN OPEN IMPLEMENTATION OF THE PNTALK SYSTEM

Vladimír Janoušek[1], Radek Kočí[1]

[1]Brno University of Technology, Faculty of Information Technology,
Božetěchova 2, 612 66, Brno, Czech Republic
{janousek,koci}@fit.vutbr.cz
http://www.fit.vutbr.cz

ABSTRACT

PNtalk is a long term project developing a system which combines Petri nets and Smalltalk in a consistent way by introducing Object Oriented Petri Nets. The objective of PNtalk is modeling, simulation, and prototyping systems. Our current focus is higher level of dynamism and higher level of control over PNtalk features in order to allow heterogeneous modeling and better interoperability with its environment. The most important feature is a possibility of replacing Petri nets by other formalisms in some parts of a model. To achieve these goals, an open approach proposing appropriate level of control over selected features of models is adopted. The open implementation of PNtalk is accomplished by its reflective metalevel architecture.

KEYWORDS

Metalevel architecture, Reflectivity, Modelling, Simulation, Prototyping, Object Oriented Petri Nets

1 INTRODUCTION

PNtalk is a long term project leading towards a tool for modelling, simulation, and prototyping complex systems. As the primary formalism of the PNtalk system, the Object Oriented Petri Nets (OOPNs) have been chosen. OOPNs [6] combine advantages of object orientation and Petri nets. The OOPNs basis consists of high level Petri nets allowing to describe the work-flow of models and their parallelism. Object orientation brings mainly the structuralization of models and communication protocol between parts of the model (objects). OOPNs define objects in a very similar way like other object oriented languages (e.g. Smalltalk [4]) but with one important difference – methods are not described as sequences of commands but by means of high-level Petri nets. At a method call, a new instance (a copy) of the appropriate net is created and made ready for running.

Our current focus is a higher level of dynamism and a higher level of control over the PNtalk features. PNtalk classes are special objects which can be built incrementally during run time (like in Smalltalk). A method is then understood as the least unit constituting the basic block of the class. Along with considering the method to be a pattern and the invoked method to be a copy of that pattern, we can also think about dynamic changes of those structures. By the pattern change, we gain new behavior of its new copies (invocations). The copies originated till this time are not changed. Nevertheless, the copy itself can be modified according to the same principle. The dynamic changes are not restricted just to Petri nets modifications – a possibility to modify or to replace the used paradigm (modeling language) can be very interesting, e.g. by other variant of Petri nets or a sequential computation specified by Smalltalk.

To achieve the above sketched goal the open implementation issue including the reflective and metalevels architectures, appear to be a possible and good way. A suitable environment for an implementation of reflective principles is object orientation because it brings along the clearly defined structures and clearly defined interfaces between those structures. Structures are encapsulated in objects and the only way how to affect other structures is via their interface. The interface is sometimes called a protocol (this terminology has come from Smalltalk). The feasibility of open approaches depends on the degree of their implementation (thus on what the designers do consider as a profitable limit of the open implementation degree). The PNtalk system architecture is based on the idea of having a system as open as possible. The goal of this paper is to discuss advantages and possibilities of such an idea.

2 OPEN IMPLEMENTATIONS

In this section, we try to outline and explain the open implementation approach from several point of view. The most important one are these: why we deal with open implementations and how to make a framework intended for building the open systems.

Recent trend in modern systems for complex application support (e.g. operating systems) is not only to allow applications to use services offered by the given system, but also to offer means to control how these services are provided and processed. This trend can seem to be contrary to the more traditional approach – the black-box abstraction. It says some abstraction (object) should expose its functionality but hide its implementation. The black-box abstraction has many attractive qualities and brings a possibility of portability, reusing or simplicity of the design process. Nevertheless, it does not allow to adapt parts of the system according to changing requirements, to develop applications during its life-time etc. The open implementation principles offer a solution of the problems above. It is needed to remark that these principles should be understood rather like the framework intended for more flexible design and use of black-boxes[1].

The basic idea of an open implementation is to allow a model to inspect inner aspects of objects (introspection), and possibly affect some (eventually all) those aspects (reflection). The classic case of an open implementation is the metalevel architecture partitioning a model into two layers – the basic (or domain) level and the metalevel [12]. All objects describing the domain problem represent the basic level. To each object at the basic level there is a special object (or a set of objects) at the metalevel – metaobject. The metalevel[2] represents a higher sphere of the control and de facto describes information about information. A metaobject offers the protocol for inspecting and changing selected aspects of its basic object. We can say the metalevel objects control the life-time of the basic level objects. In general, there can be more metalevels – each superior level controls some aspects of its sublevel. The meta-level architecture allows not only to affect structures of objects but also to modify their computational behavior, e.g. the way how the objects react to messages, what other operations are to be processed in a consequence of sending or receiving messages, etc.

As an example, we may name the Smalltalk system [4]. Each class has its metaclass which has the same meaning as the class for regular objects. Metaclasses control the class behavior and classes control the object behavior. Thus, Smalltalk has two levels of control (it implies there is one metalevel) – the one for object controlling and the one for class controlling. A deeper introduction to open implementations and metalevel architectures can be found in [12, 15, 9, 13].

[1]Thus, it is not a good idea to reprobate the black-box abstraction. However, in some cases it is useful to have mechanisms to partly evade the black-box concepts.

[2]The word *meta* comes from Greek and means something like *after* or *behalf*. In our conception it can be understood as a denotation of something what stays behind an object and reflects (or describes) its features and properties.

3 OBJECT ORIENTED PETRI NETS

PNtalk and the associated OOPNs 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[3] that offer reentrant services (named by the corresponding message patterns) to other objects (clients). 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.

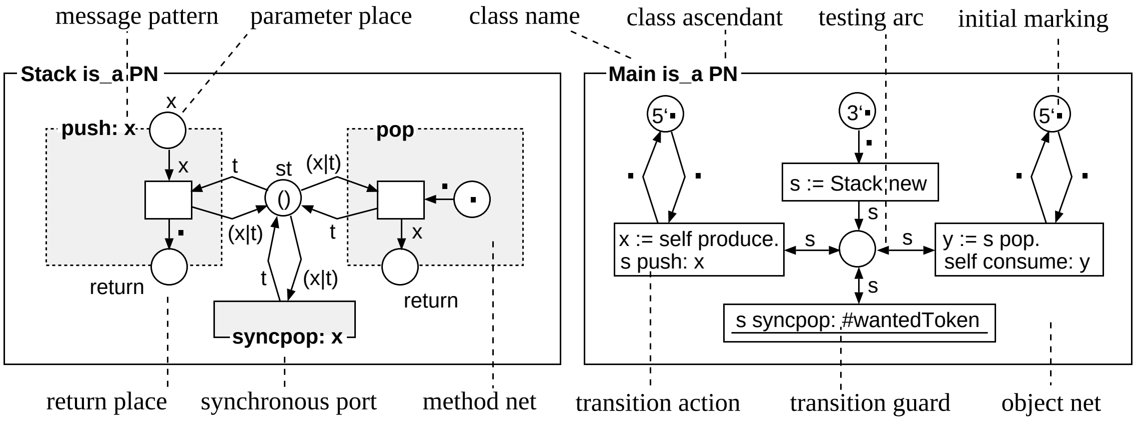

Figure 1. An OOPNs example (Methods *produce* and *consume* are not shown).

An example illustrating the OOPNs formalism is shown in Figure 1. As shown in Figure 1, a place can be inscribed by an initial marking (a multiset of objects) and an initial action (allowing a creation and initialization of complex objects to be initially stored in the place; not shown in the Figure 1). A transition can have a guard restricting its firability and an action to be performed whenever the transition is fired. Finally, arcs specify multisets of tokens to be moved from/to appropriate places as the corresponding transition is being fired.

4 PNTALK SYSTEM ARCHITECTURE

A basic overview of the PNtalk architecture is presented in Figure 2. The architecture introduces a new meta level between the domain objects (i.e. PNtalk classes and PNtalk objects) and Smalltalk. Objects belonging to the metalevel are implemented by classes of Smalltalk. The metalevel comprises metaobjects (i.e. instances of Smalltalk classes) which control PNtalk classes and PNtalk objects. A metaobject of the first kind describes *the structure* of a PNtalk class and it also defines the ways of manipulation with the PNtalk class. Metaobjects of the second kind describe *the computational behavior* of PNtalk objects (instances of PNtalk classes). Each such metaobject consists of a set of net instances (copies) and it also offers the means for structural modifications of the corresponding PNtalk objects. Thus both the PNtalk classes and the PNtalk objects are open to the dynamic changes.

Metaobjects representing PNtalk objects are characterized by the following important features. Firstly, while the Smalltalk metalevel architecture is based on classes (i.e. objects are instances of their metaobjects), the PNtalk metalevel architecture is based on objects (i.e. objects

[3]The term *active server* means an object that can have its own activity which starts as soon as the object is created (the activity is described by the *object net* in OOPNs).

are not instances of metaobjects, but metaobjects implement corresponding domain objects). Secondly, the meta objects controlling the PNtalk objects are *actors*. They implement PNtalk concurrency model and they serialize all accesses and requirements to the PNtalk object.

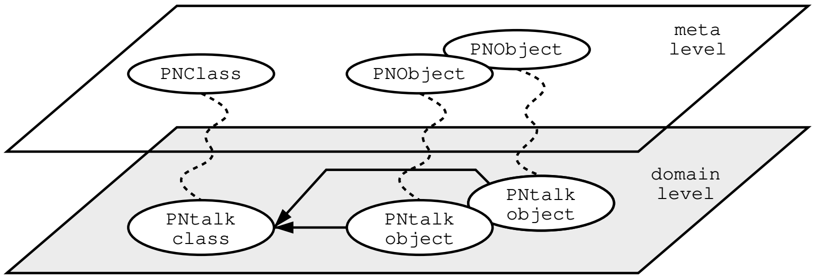

Figure 2. The PNtalk metalevel architecture.

Since this paper concentrates on motivations and features of the PNtalk metalevel architecture, we will not deal with the PNtalk architecture in details. A more detailed view on the PNtalk system and the PNtalk project can be found in [7, 8, 16].

5 ON THE OPEN PNTALK SYSTEM APPLICABILITY

There is a question what is the purpose of a system with a robust degree of the open approach moreover affiliated with the Object oriented Petri nets (OOPNs) formalism. The goal of PNtalk project is not only to make a tool intended for modelling and simulation but also to make tool allowing the developed model to be integrated to a real environment. Such a model can then serve as a part of the prototype or the target application. When we take in account the reflective features, we can use this system as a framework for interactive application development. The framework allows us to build models and prototypes, to develop new paradigms, to combine different paradigms for model specification, including automatic adaptation of paradigms or models evolution during run time. In other words, our goal is to develop a lite-form of an operating system (a lite-form in the sense of security features) primarily intended for a work with models and simulations.

One of possible application domain is artificial intelligence, especially the area of intelligent multi-agent systems. The process of the agent reasoning can be characterized by its inner structures changes. Moreover, the whole structure of a multi-agent system can be highly dynamic. The present goals of the PNtalk project are to check the promising features of open systems discussed above in practise and to check application of such system in the area of intelligent agent modelling and prototyping or the complex system design.

6 RELATED WORK

The PNtalk project is closely related with similar works. The idea of merging Petri nets and objects has been found in the first half of 1990s independently by several researchers. The probably best known issues have been introduced by Lakos [10], Sibertin-Blanc [17], Moldt [14], and Valk [20]. The Object oriented Petri nets and PNtalk language and system was developed in 1994 and published in [3, 5, 6].

The idea of object-oriented computational reflection (including structural and behavioral changes of objects at run time) was proposed in 1970s [4] but roots of this concept are much older (Lisp, Universal Turing machine). As the examples of recent activities the following projects can be cited – Kiczales [9] and Maes [12] introducing aspect-oriented programming, Actalk [2] introducing concurrency via reflection in Smalltalk, Apertos [21] and TUNES [18] are attempts to develop a highly flexible operating system using computational reflection. As to the reflection in modeling and simulation, the most important (from the PNtalk point of view) are Barros [1] and Uhrmacher [19] introducing reflectivity to DEVS [22] in order to allow for structural changes of models. These approaches are important especially for modeling and simulation of intelligent agents.

Our current attempts to incorporate reflection to the Petri nets also are not alone. Lakos [11] presents a reflective approach to Object Petri Nets implementation. He emphasizes the advantages of that approach in a clean, flexible and efficient implementation, and also in a possibility to investigate alternative scheduling schemes, interaction policies, etc.

7 CONCLUSION

In this paper, we presented the metalevel PNtalk system architecture, especially motivations and abilities that stem from. The present architecture is faced with two main problems. Firstly, the architecture allowing dynamic changes presented above has to be implemented. Secondly, the system lacks a recommendation how to use it, eventually case studies of typical problems and their solving. So the methodology has to be evolved. We are now in a stage of finishing a prototype implementing the presented architecture. We believe that the prototype will allow us to gain some experience needed for the development of the methodology and a further development of the PNtalk system itself.

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. 102/04/0780 "Automated Methods and Tools Supporting Development of Reliable Concurrent and Distributed Systems".

REFERENCES

[1] F. J. Barros (1997). Modeling formalisms for dynamic structure systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, v.7 n.4, 501-515.

[2] J. Briot (1999). Actalk : A Framework for Object Oriented Concurrent Programming - Design and Experience. In *Object-Based Parallel and Distributed Computing II - Proceedings of the 2nd France-Japan Workshop (OBPDC'97)*. Herms Science.

[3] M. Češka, V. Janoušek and T. Vojnar (1997). PNtalk – A Computerized Tool for Object-Oriented Petri Nets Modelling. In *Computer Aided Systems Theory and Technology – EUROCAST'97*, volume 1333 of *LNCS*, 591–610, Spain. Springer-Verlag.

[4] A. Goldberg and D. Robson (1989). *Smalltalk 80: The Language*. Addison-Wesley.

[5] V. Janoušek (1995). PNtalk: Object Orientation in Petri nets. In *Proc. of European Simulation Multiconference ESM'95*, Prague, Czech Republic, 196-200.

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

[7] V. Janoušek and R. Kočí (2002). PNtalk - An Open System for Prototyping and Simulation. In *Proceedings of The 28th ASU Conference*, FIT VUT, Brno, CZ, 133-146, ISSN 1102-593X.

[8] V. Janoušek and R. Kočí (2003). PNtalk: Concurrent Language with MOP. In *Proceedings of the CS&P'2003 Workshop*, Warsawa, PL, UW, 271-282, ISBN 83-88374-71-0.

[9] G. Kiczales, J. Lamping, A. Menhdhekar, Ch.Maeda, C. Lopes, J.M. Loingtier and J. Irwin (1997). Aspect-Oriented Programming, In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, Springer-Verlag, Berlin, Heidelberg, and New York, 220–242.

[10] C. A. Lakos (1995). From Coloured Petri Nets to Object Petri Nets. In *Proceedings of the Application and Theory of Petri Nets 1995*, vol. 935, Springer-Verlag, Berlin, Germany, 278–297.

[11] Ch. Lakos (1996). Towards a Reflective Implementation of Object Petri Nets. In *Proceedings of TOOLS Pacific*, Melbourne, Australia, 129-140.

[12] P. Maes (1987): *Computational reflection.* Technical report, Artifical Inteligence Laboratory, Vrije Universiteit Brusel.

[13] H. Masuhara, S. Matsuoka, T. Watanabe and A. Yonezawa (1992). Object-oriented concurrent reflective languages can be implemented efficiently. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, SIGPLAN Notices, volume 27, number 10, ACM Press, New York, NY, 127–144.

[14] D. Moldt (1995). OOA and Petri Nets for System Specification. In *Application and Theory of Petri Nets; Lecture Notes in Computer Science*, 16th International Conference, Italy.

[15] R. Pawlak (1998). Metaobject Protocols For Distributed Programming. *Technical report*, Laboratoire CNAM-CEDRIC, Paris.

[16] PNtalk Project Home Page, http://www.fit.vutbr.cz/˜janousek/pntalk

[17] C. Sibertin-Blanc (1994). Cooperative Nets. In *Proceedings of Application and Theory of Petri Nets*, vol. 815, Springer-Verlag, Berlin, Germany, 471–490.

[18] The TUNES Project for a Free Reflective Computing System, http://tunes.org

[19] A. M. Uhrmacher (2001). Dynamic structures in modeling and simulation: a reflective approach. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Volume 11, Issue 2, 206–232, ISSN:1049-3301.

[20] R. Valk (1998). Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science*, Vol. 1420, Springer-Verlag, Berlin, 1-25.

[21] Y. Yokote (1992). The Apertos reflective operating system: The concept and its implementation. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, SIGPLAN Notices, volume 27, number 10, ACM Press, New York, 414–434.

[22] B. Zeigler, T. Kim and H. Praehofer (2000). Theory of Modeling and Simulation. *Academic Press*, 2nd edition, 510 pages, ISBN: 0127784551.