Synchronous Interactions of Objects in Object Oriented Petri Nets

Vladimír Janoušek ⋆
janousek@dcse.fee.vutbr.cz

Abstract: This paper presents an additional concept of object interactions in Object oriented Petri nets. Object oriented Petri nets (OOPN) is a formalism developed at the Department of Computer Science and Engineering FEI VUT in Brno during last few years in order to overcome problems with modeling highly dynamic systems by means of Petri nets. An experimental language based on OOPN is called PNtalk. OOPN and PNtalk are comparable with other well known approaches to object orientation in Petri nets. It is based on active objects communicating by a client-server (request-reply) protocol. Objects are described by high-level Petri nets enriched with a message sending mechanism and by a possibility to dynamically instantiate objects and methods. In order to improve its applicability in modeling some non-trivial real-world systems such as flexible manufacturing and workflow systems on an appropriate level of abstraction a new feature was added to OOPN. It allows synchronous interactions of objects and does not violate the pure object orientation of OOPN. This new concept is represented by *synchronous ports*. Its basic principle as well as some methodological aspects are discussed in this paper.

Key Words: Petri nets, object orientation, synchronous interaction

1 Introduction

The development of formalisms based on merging Petri nets and the object oriented paradigm is motivated by a need to deal with systems consisting of some encapsulated active entities which can communicate and which can be dynamically created and discarded during the system life time.

Many systems of this kind can be studied by means of multilevel models, i.e. models allowing several levels of activity. Multilevel modeling in the context of Petri nets can be described as follows. An object o_0 is described by a Petri net. Each token in the net represents some object o_i. Each object o_i is also described by a Petri net n_i. Each token in a net n_i represents some object o_j and so on.

Object oriented Petri nets [Jan94a, Jan95b, Jan96, Jan98] (OOPN) and the PNtalk language encapsulate sets of nets into objects. One of the nets encapsulated in an object o controls the object's own activity, other nets can control processes (inside the object o) which can be dynamically spawned each time the object o receives a message (from another object) requesting some service. This means that objects in OOPN communicate by means of a client-server (request-reply) protocol as it is usual in most widely known object oriented programming languages. The main difference of OOPN/PNtalk is that all objects are implicitly considered to be potentially active and that they can accept several requests simultaneously and process them in parallel.

As for the suitability of OOPN/PNtalk for multilevel modeling, our experience gained from a PNtalk application to flexible manufacturing system modeling [JV98] can be summarized as follows. Although the client-server communication can be successfully used for multilevel

⋆ Department of Computer Science and Engineering, Faculty of Computer Science and Engineering, Technical University of Brno, Božetěchova 2, CZ-612 66 Brno

modeling, in many cases it seems to be a quite low-level form of interaction, i.e. it does not fit to the selected level of abstraction because it is closer to a (computer-based) implementation point of view.

The above discussion implies that some higher-level communication mechanism (e.g. some mechanism allowing synchronous interactions of objects) is needed. A possibility of synchronous atomic interaction in PNtalk was informally mentioned in [Jan96] and formally defined in [Jan98] in the form of *synchronous ports*. The rest of this article is dedicated to an explanation of synchronous ports and their application in multilevel modeling and simulation.

2 Object System

Prior to the explanation of synchronous ports let us mention another formalism for multilevel modeling which was introduced in [Val95, Val98] by Rudiger Valk. We deal with this formalism because some of its methodological aspects will be used as basis for the synchronous ports application.

R. Valk defined a two-level model called the *Object System*. It is a kind of high-level Petri net with tokens representing objects described by P/T-nets. The restriction to two levels of activity seems to be applicable in many cases including workflow and flexible manufacturing systems. Moreover, Valk's object system does not allow dynamic instantiation of objects so it is simple enough to allow for developing analysis techniques [Val98] (note that the OOPN/PNtalk is a much more general formalism than the Object System, but its analysis is in general much more complicated).

The Object System is tuple $OS = (SN, TN, Y)$ where SN is the *system net*, $TN = \{TN_j | 1 \leq j \leq n\}$ is the set of *token nets*, and Y is the set of *synchronized transition steps*. The system net is a high-level Petri net, allowing only token nets to be its tokens. Its flow relation is restricted in such a way that it behaves only as a token moving net, no token net can be created or removed from the system. The token nets are P/T nets. The set Y comprises the synchronized transition steps that potentially could occur in the object system. A synchronized step s always involves one system net transition and one or more token net transitions.

We say that a token net transition t is *autonomous* if it is not included in any synchronized step s, otherwise it is *synchronized*. Enabledness and firing of an autonomous transition is defined as usual and it affects only the corresponding token net. System net transitions are supposed not to occur autonomously.

A synchronized transition t_i of a token net can *occur* only together with other transitions which are (together with t_i) included in some synchronized transition step s (one of the involved transitions is a transition t belonging to the object net). All these transitions must be simultaneously enabled in their respective nets and the appropriate binding b of variables of the transition t must contain all involved token nets. This means that transitions in token nets can occur only synchronously with such a transition in the object net which moves the token net from one place to another one.

Interested readers can found the formal definition of Valk's object system e.g. in [Val95, Val98] or in [NKKX97].

3 Synchronous ports

Unlike to Valk's Object System the OOPN/PNtalk nets (objects) represented by tokens are not restricted to P/T nets and the number of activity levels is not restricted to two. All nets in OOPN/PNtalk are high-level nets and each token represents an object. Some objects, e.g. integers, are trivial (constants) and other objects are described by Petri nets. Each non-trivial object is described by its class which comprises an object net, a set of method nets, and a set of synchronous ports. Method nets are intended to be dynamically instantiated by a class instance as a reaction to respective messages received by the object. They implement reactions to the incoming messages. When the execution of a method net is finished (this is indicated by marking its return-place) then the value of the token in the return-place is sent back to the client (message sender). Message sending is specified in *transition actions* which can be processed non-atomically (because of waiting for a reply). This way the client-server protocol is defined in OOPN/PNtalk. More information can be found in [ČJV97, Jan98].

Synchronous ports have been introduced to OOPN/PNtalk as an additional form of communication. As opposite to the client-server protocol, this communication is realized as an atomic operation. A synchronous port has a form of a virtual transition (belonging to some object net) with a message pattern assigned to it. It can occur only together with a transition which invokes it. Although the synchronous port invocation is syntactically available via message sending, it is similar to the Prolog's predicate evaluating rather than function or procedure (method) invocation in Smalltalk or C++. This is the reason why it is used in *transition guards* rather than in transition actions. A transition guard (together with other conditions) determines if the transition is enabled. The synchronous communication priciple requires all the involved synchronous ports to be enabled (as though they were transitions). An occurrence of a transition implies that it occurs simultaneously with all the involved synchronous ports (as though they were transitions) in other objects. Note that a synchronous port can have a guard which can call other synchronous ports. Of course the mechanism of dynamic binding which allows for polymorphism applies here as well as in the case of client-server communication of objects. A detailed description of synchronous ports together with their formal definition can be found in [Jan98].

4 An application of synchronous ports

Synchronous ports were originally demonstrated on various mutations of the dining philosophers system in [Jan98]. The following example is probably a bit closer to real-world problems. It is taken from [Val98]. In the example a workflow of the Dutch Justice Department is modeled (the author of the original model is W.M.P. van der Aalst who is a Dutchman). It has originaly been used for a demonstration of modeling and analysis of workflow applications of Petri nets [vdA98].

The example is defined in [Val98] as follows. When a criminal offense has happened and the police has a suspect, a record is made by an official. This is printed and sent to the secretary of the Justice Department. Extra information about the history of the suspect and some data from the local government are supplied and completed by a second official. Meanwhile the information on the official record is verified by a secretary. When these activities are completed, an official examines the case and a prosecutor determines whether the suspect is summoned, charged or that the case is suspended.

Now we will demonstrate how this system (as a representative of similar systems) can be described by means of OOPN/PNtalk using synchronous ports. In fact, it is Valk's approach adapted for OOPN/PNtalk. The OOPN/PNtalk model of this system can be built in several steps (note that the client-server communication is not used in this approach because of its low-levelness with respect to the abstraction level used here):

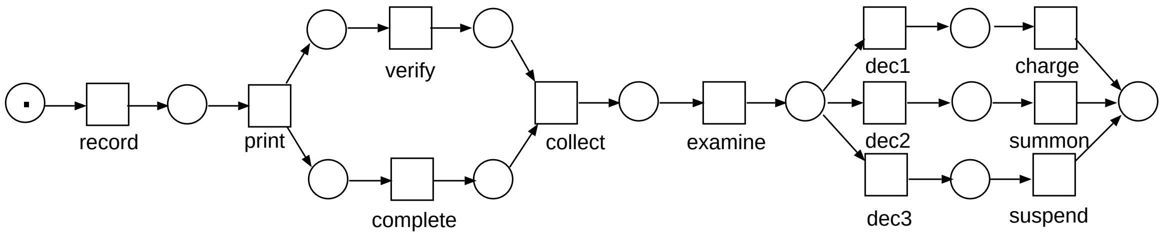

Figure 1: A case specification.

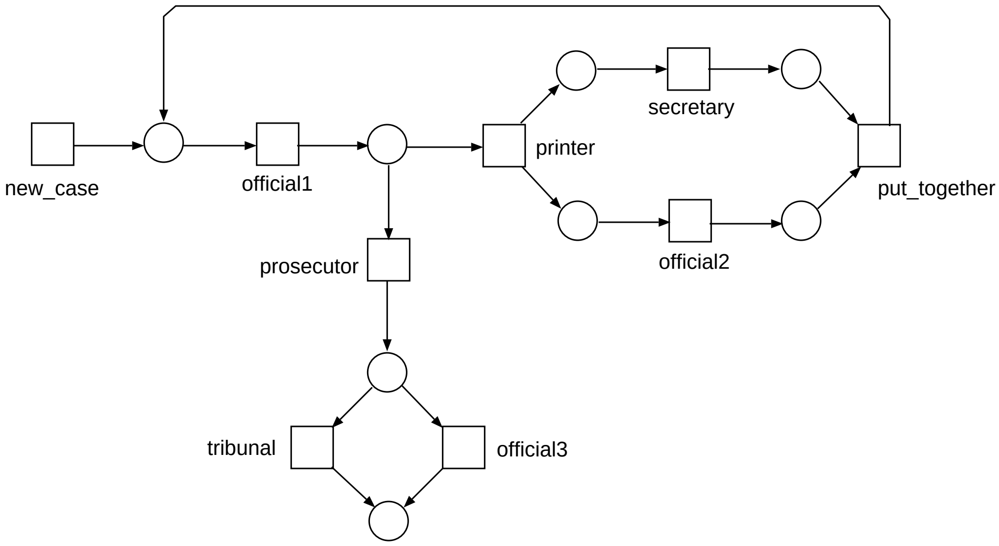

Figure 2: A system structure specification.

Step 1. A task (case) specification (see fig. 1). Required operations and their causality are specified by a Petri net. Each class of problems (tasks or cases) is described by its own net.

Step 2. The physical structure of the system under examination (office, enterprise, manufacturing system) which is intended to solve problems (or execute tasks or deal with cases) is specified by another Petri net. Transitions in this net represent elements which are capable of executing some operations mentioned in the step 1.
In our example, the structure is depicted in fig. 2. We suppose that the first official examines the case which he firstly recorded.

Step 3. A relation of operations from the step 1 to the transitions in the net from the step 2 is specified. In our example, it can be specified as follows: The `official1` can `record` and `examine` the case, the `printer` can `print`, the `secretary` can `verify`, the `official2` can complete the case, somebody (identified as `put_together`) can `collect` the case, the `prosecutor` can do decisisions `dec1`, `dec2`, and `dec3`, the `tribunal` can `charge` or `summon` the case, the `official3` can suspend it.

Step 4. The net which describes the physical structure is refined in order to express executing alternative operations and/or sequences of operations (see fig. 4).

Step 5. All the transitions representing operations in the nets from the step 1 are declared as synchronous ports (see fig. 3), all the transitions representing the execution of operations

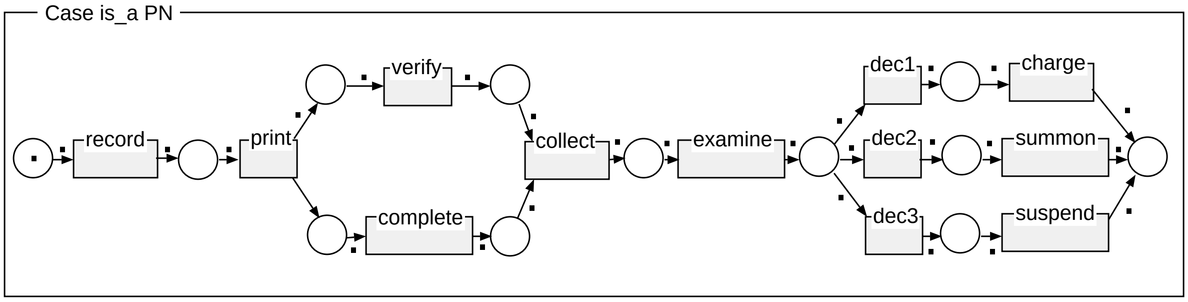

Figure 3: The class `Case`.

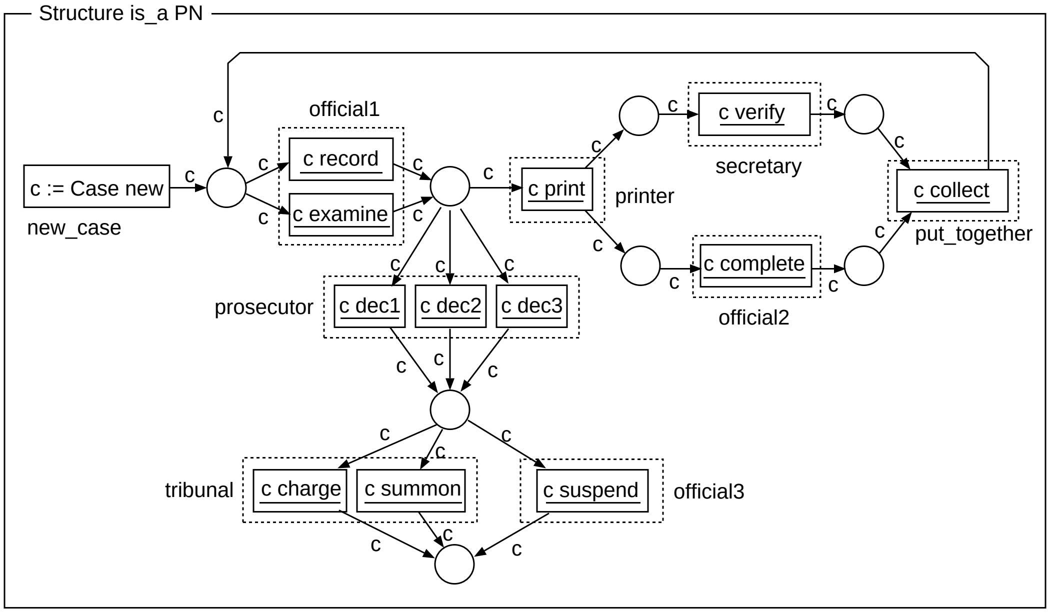

Figure 4: The class `Structure`.

are specified as synchronous port callers (see fig. 4 – note that the transition guards are underlined).

Step 6. To allow simulation, model initialization mechanisms and a model environment must be specified. Moreover, time restrictions can be added to some transitions.

Note that if there is a non-trivial operation which cannot be modeled by a single transition in a net from the step 1, then it has to be modeled by two transitions which represent the begining and the end of the operation. Refinement of the net from the step 2 in the step 4 has to reflect this situation.

Also note that thanks to the PNtalk's syntactic sugar allowing disjunction in transition guards it is possible to specify invocation of several alternative synchronous ports in a guard of a single transition. This leads to more a straightforward implementation of the system structure and its connection to synchronous ports of a case as it is shown on fig. 5.

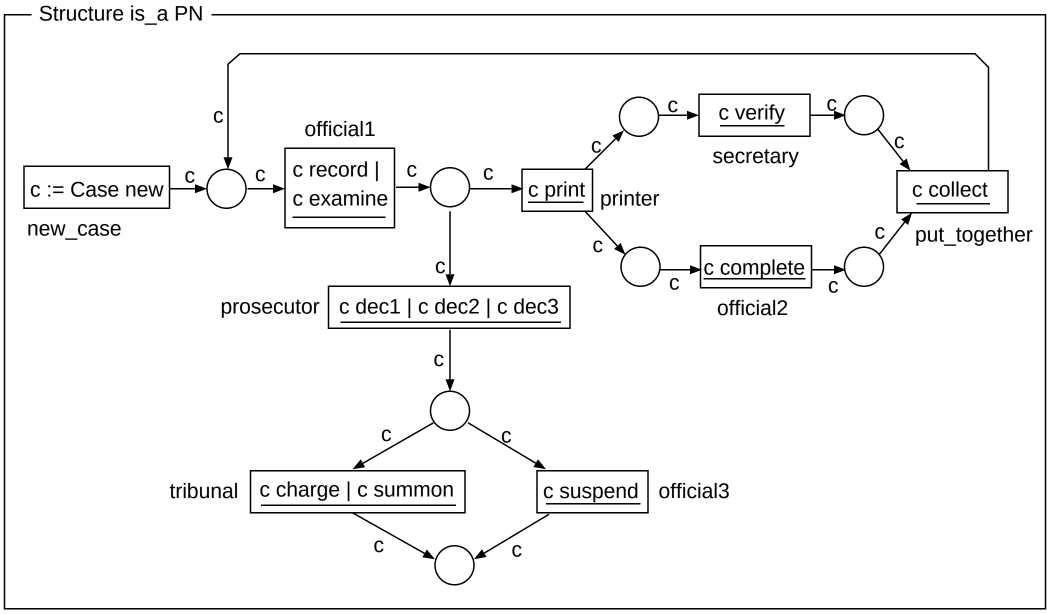

Figure 5: The class `Structure` – an alternative implementation.

5 Conclusions

The article has demonstrated synchronous ports as a new feature of OOPN/PNtalk allowing synchronous interaction of objects. It has been applied in two-level modeling of workflow where particular cases as well as the structure processing them are modeled separately by PNtalk objects. Despite these objects communicate via a mechanism which is syntactically available as message sending (allowing polymorphism although in the presented example this feature has not been used) this mechanism makes these objects tightly coupled because it leads to a synchronous execution of transitions in various objects. This means that an occurence of some event in some object is conditional on synchronous occurence of another event in another object and vice versa. This holds independently to which object owns respective synchronous port and which object invokes this synchronous port. As a consequence, the presented example could be modified in such a way that the case would call synchronous ports of the structure (this would require the structure to pass its *self* to the case when it is created or when it comes to the structure).

Note that the presented example represents a very simple application of synchronous ports. Neither parameters nor guards of synchronous ports (which could potentially invoke other synchronous ports) were used here. All these possibilies are described in [Jan98] and will be more studied in the future (mainly from a methodological point of view). Nevertheless, the simple example has demonstrated one possible way of applying OOPN/PNtalk in multilevel modeling. PNtalk has been used as a very high level implementation language here. Note that Valk [Val98] proposes a special formalism and a special method of modeling two-level activities. On the other hand, PNtalk is a universal programming language. So it is clear that not all its features are supposed to be applied in each model. Moreover, a serious application of OOPN/PNtalk in workflow modeling will probably require a special formalism and appropriate tool together with a translator to the PNtalk language. We dealt with modeling and simulation only. Neither analysis, nor optimization, nor scheduling problems have been considered here. All these topics are considered to be investigated as a part of our future research activities.

At the very end let us mention that synchronous communication of objects implies a need to solve some problems connected to applying OOPN/PNtalk in hybrid models where a model in PNtalk has to communicate with other models implemented in other languages and environments. Since objects in other object oriented languages obviously do not communicate the way PNtalk does a communication protocol for synchronous interactions has to be defined for these languages. Probably only some object oriented extension of Prolog could communicate with OOPN without the "impedance" problem. On the other hand, the client-server communication (which is the basic communication concept of OOPN/PNtalk) is fully compatible with other object oriented languages. This mechanism was not discussed in this paper because it is suitable for models on lower levels of abstraction than we dealt with here.

This work was supported by the Grant Agency of the Czech Republic under the contract 102/98/0552 "Research and Applications of Heterogeneous Models" and by the Universities Development Fund of the Czech Ministry of Education under the contract G1-1092 "Temporal Properties of Models Described by Object-Oriented Petri Nets".

References

[ČJ96] M. Češka and V. Janoušek. Object Orientation in Petri Nets. In *Object Oriented Modelling and Simulation, Proceedings of the 22nd Conference of the ASU*, pages 69–80, Clermont-Ferrand, France, September 1996. University Blaise Pascal.

[ČJV97] 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. A Selection of Papers from the 6th International Workshop on Computer Aided Systems Theory*, volume 1333 of *Lecture Notes in Computer Science*, pages 591–610, Las Palmas de Gran Canaria, Spain, February 1997. Springer-Verlag.

[ČJV98] M. Češka, V. Janoušek, and T. Vojnar. Object-Oriented Petri Nets, Their Simulation, and Analysis. In *Proceedings of 1998 IEEE International Conference on Systems, Man, and Cybernetics, SMC'98*, IEEE Catalog Number 98CH36218, pages 256–261, Hyatt La Jolla, San Diego, California, USA, October 1998. Omnipress.

[Jan94a] V. Janoušek. Merging Petri Nets and Objects. In *Proceedings of International Winter School SOFSEM '94*, pages 45–50, 1994.

[Jan94b] V. Janoušek. OOPN: A High-Level Language for Modeling and Simulation. In J. Štefan, editor, *Proceedings of MOSIS '94*. MARQ Ostrava, 1994.

[Jan95a] V. Janoušek. Functional and Object Oriented Structuring of Petri Nets. In *Proceedings of International Conference on Computer Science*, pages 44–47, Ostrava, Czech Republic, September 1995. Technical University of Ostrava.

[Jan95b] V. Janoušek. PNtalk: Object Orientation in Petri Nets. In *Proceedings of European Simulation Multicoference ESM '95*, pages 196–200, Prague, Czech Republic, 1995. Czech Technical University.

[Jan96] V. Janoušek. The PNtalk Language and System. In J. Štefan, editor, *Proceedings of MOSIS '96*, pages 233–238. MARQ Ostrava, 1996.

[Jan97] V. Janoušek. Objektov orientovan modelovn paralelnch systm. In J. Štefan, editor, *Sbornk ASIS '97*, pages 365–371. MARQ Ostrava, 1997.

[Jan98] V. Janoušek. *Modeling Objects by Petri Nets*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering and Computer Science, Technical University of Brno, Czech Republic, 1998. (In Czech).

[JV97] V. Janoušek and T. Vojnar. PNtalk Project, 1997.
`http://www.fee.vutbr.cz/˜janousek/pntalk/pntalk.html`
`http://www.daimi.au.dk/˜petrinet/tools/db/pntalk.html`.

[JV98] V. Janoušek and T. Vojnar. Modelling a Flexible Manufacturing System. In J. Štefan, editor, *Proceedings of MOSIS '98*, pages 195–200. MARQ Ostrava, 1998.

[NKKX97] K.-P. Neuendorf, D. Kiritsis, T. Kis, and P. Xirouchakis. Two-Level Petri Net Modelling For Integrated Process and Job Shop Production Planning. In F. DiCesare, M. Silva, and R. Valette, editors, *Proceedings of the Workshop on Manufacturing and Petri Nets, the 17th International Conference on Application and Theory of Petri Nets*, pages 135–150, Toulouse, France, June 1997.

[Val95] R. Valk. Petri Nets as Dynamical Objects. In *Proceedings of Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, 1995.

[Val98] R. Valk. Petri Nets as Token Objects. In M. Silva J. Desel, editor, *Application and Theory of Petri nets, 18th International Conference, ICATPN '98*, pages 1–25. Springer-Verlag, 1998.

[vdA98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 1998.