A Case Study: Modelling Alternating Bit Protocol by PNtalk

Bohuslav Křena ⋆
krena@fit.vutbr.cz

Abstract: The alternating bit protocol (ABP) is a simple communication protocol that is able to cover unreliable mediums. ABP has been modelled in many modelling formalisms because it has most of important features of communication protocols although it is very simple. In this paper, we show one of possible models of ABP in modelling language called PNtalk that has been developed in our faculty.

Keywords: Modelling, Alternating Bit Protocol, PNtalk, Object-Oriented Petri Nets.

1 Introduction

We are interested in modelling ABP by PNtalk because we need some new case studies for formal analysis and verification which is our main research interesting. The model of ABP can be also used as an encapsulated core for modelling some advanced communication protocols. The last motivation of this work is to show main features of the language PNtalk.

1.1 Introduction to the Alternating Bit Protocol (ABP)

The alternating bit protocol [2] is a simple communication protocol which is able to cover unreliable mediums. We suppose that a medium is an unbounded buffer that can lose and/or duplicate some of transmitted messages. For simplicity, we do not suppose changes of messages. ABP uses two mediums, one for message transmitting from a sender to a receiver, and second for transmitting an acknowledgement from the receiver to the the sender.

The ABP layer diagram and the diagram of flows is shown in Figure 1.

The Sender works as follows. It accepts a message from the application, sends it together with an additional bit b (0 or 1), and sets its timer to zero. After this, there are three possible ways:

– An occurrence of the time-out signal from the timer causes that the Sender repeats sending the same message with the same additional bit b.
– After the Sender receives an acknowledgement b, it is able to accept next message from the application. The Sender adds a reverse bit ($\hat{\mathsf{b}}$ = 1 - b) to this new message.
– Sender receives an acknowledgement $\hat{\mathsf{b}}$ which it ignores.

The Receiver works as follows. It delivers a message incoming from the Sender to the application, the additional bit b sends via the medium to the Sender, and sets its own timer to zero. After this, there are three possible ways:

⋆ Faculty of Information Technology, Brno University of Technology, Božetěchova 2, CZ-612 66 Brno

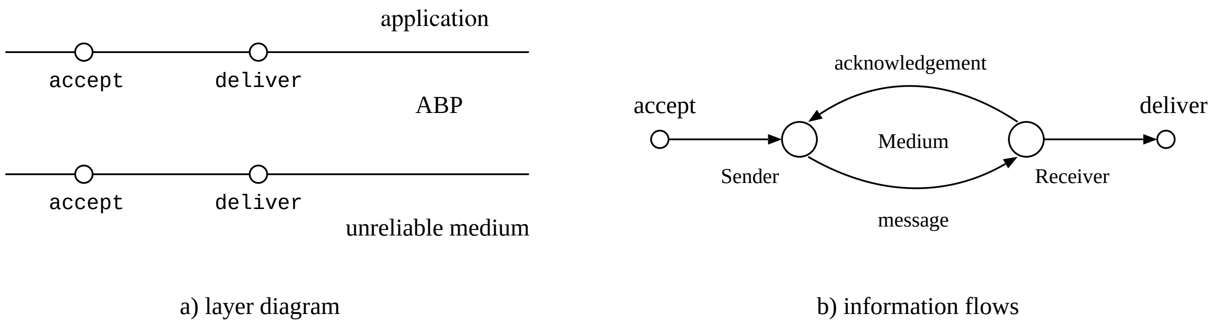

Fig. 1: Alternating bit protocol

– An occurrence of the time-out signal from the timer causes that the Receiver repeats sending the same acknowledgment bit b.
– The Receiver receives a new message with the acknowledgement bit $\hat{b}$ and delivers it to the application.
– The Receiver receives a message with the same acknowledgement bit b which it ignores.

1.2 Introduction to modelling language PNtalk

The modelling language PNtalk [4] is one of possible implementations of the object oriented Petri nets (OOPNs). The OOPN formalism (and therefore also the language PNtalk) joins together Petri nets with the concept of object orientation (OO).

Each PNtalk model consists of some PNtalk classes (one of these classes is a main class to which instance is created at the beginning). Each PNtalk class consists of one object net and any method nets. All nets mentioned above are high level Petri nets.

We assume readers familiarity with OO and also with basic concept of Petri nets and therefore we do not describe language PNtalk in any details now. We explain an unusual language constructions used in the model in notes in next section.

2 PNtalk Model of the Alternating Bit Protocol

Our model of ABP consists of four PNtalk classes. The main class of the model is the class Application which creates and uses other objects. Class Medium models the unreliable medium, and classes Sender and Receiver models the ABP logic described above.

The complete PNtalk model is shown in following subsections in graphical form. All nets of the model are also described in reasonable details there.

2.1 Class **Application**

The class Application (shown in Figure 2) is the main class of the model.

Note 1. Inside a transition box can be placed a guard expression (above the line) and an action expression (under the line). Guard expressions of all transitions in Figure 2 are empty.

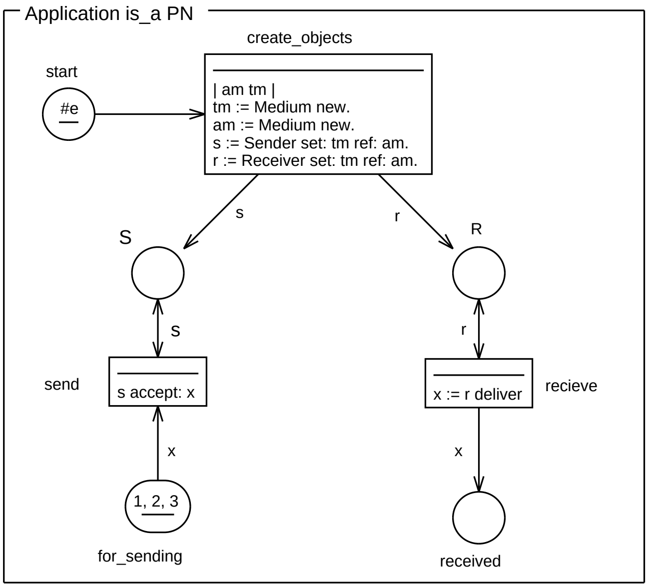

Fig. 2: Class Application – object net

Note 2. Action expressions can contain message calls. For example, the action expression in the transition send means the method accept of an object referenced by the variable s with the parameter x.

At the beginning, all objects (i.e. two instances of the class Mediums, one instance of Sender, and one Receiver) are created by firing the transition create_objects. Next, messages from the place for_sending are sent by the Sender (by the transition send) and received from the Receiver (by the transition receive).

We can observe (if we allow only finite number of losses and duplications in each medium) all messages from the place for_sending are moved to the place received after finite number of steps.

2.2 Class Medium

The class Medium (shown in Figures 3, 4, and 5) models the unreliable medium in accordance with the description in Subsection 1.1.

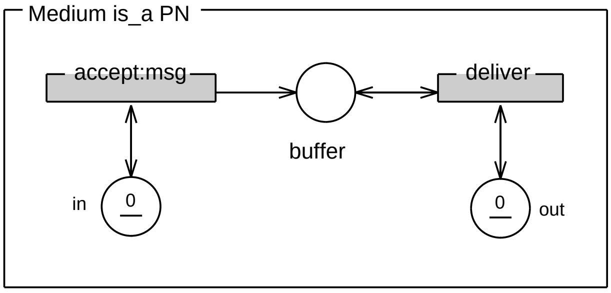

Fig. 3: Class Medium – object net

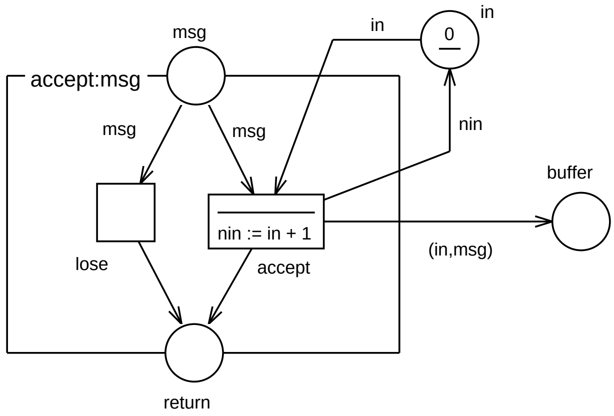

Fig. 4: Class Medium – method net accept

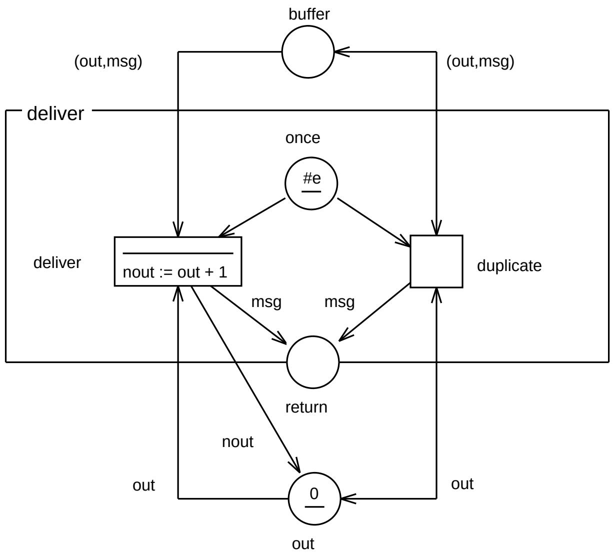

Fig. 5: Class Medium – method net deliver

Note 3. An instance of object net represents state of an object. Methods (represented by gray boxes in a object net) are used for communication with surroundings and changes of an object state (i.e. marking of places in an object net).

Note 4. On the top of the box border of a method net are parameter places (it the method has some parameters) and on the bottom of the box border is one return place. A method is finished by putting a token into the return place.

The Medium can accept a message (by the method accept) or deliver a message (by the method deliver). Carried massages are placed in the place buffer. The order of messages is guaranteed by adding an increasing integer to each message. Losses of messages can be done by the the transition lose in the method accept and duplications can be done by the by the transition duplicate in the method deliver.

This model of the medium makes the state space of the ABP model infinite because the number of duplicated messages is not bound. It is convenient to limit number of message duplications in order to make formal analysis and verification easier. However, this is not necessary because formal analysis and verification is also possible for systems with unbounded, lossy FIFO channels [1].

2.3 Class Sender

The class Sender models part of ABP that is responsible for accepting messages from the application and sending them via the Medium to the Receiver.

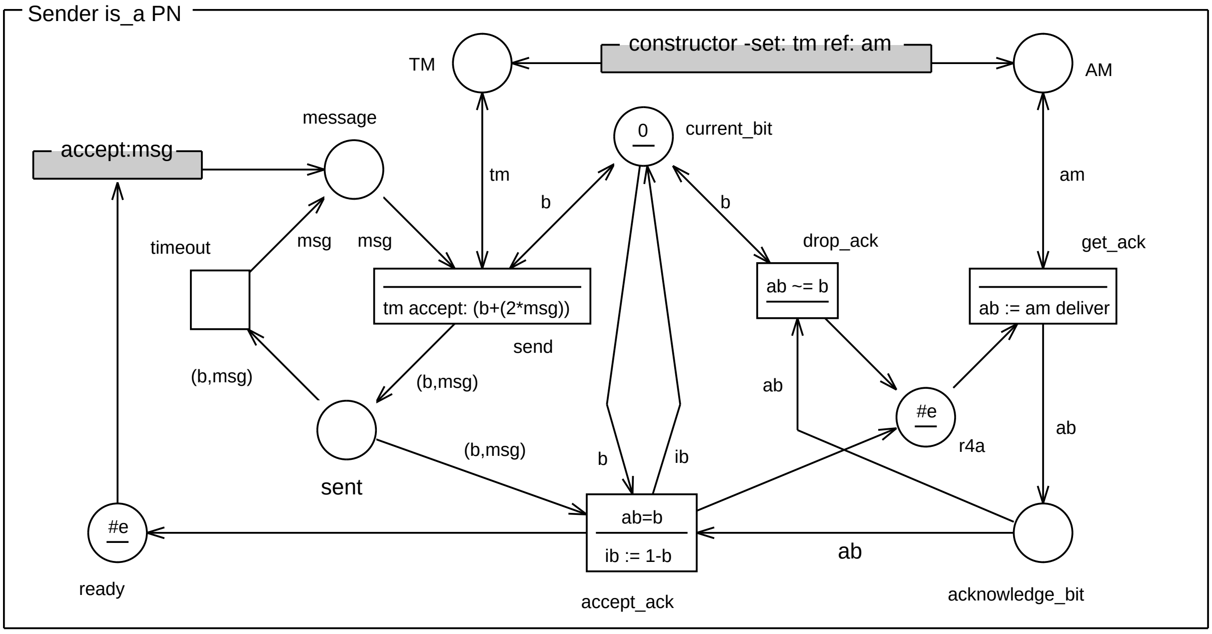

Fig. 6: Class Sender – object net

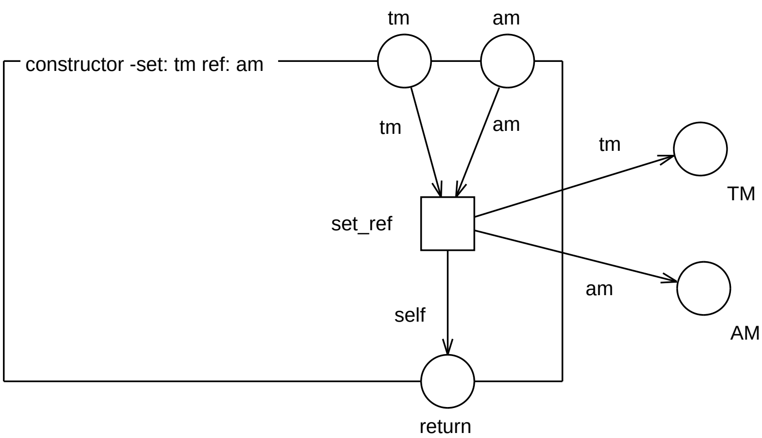

Fig. 7: Class Sender – non-implicit constructor net

Note 5. A non-implicit constructor (e.g. constructor set: ref:) is a special method which creates a new instance of the class and sets it initial marking.

The sender model consists of the object net and two method nets. First one is a non-implicit constructor trans: tm ack: am that allows the Application to set references to both mediums and second one is a method accept that is a first service access point of ABP.

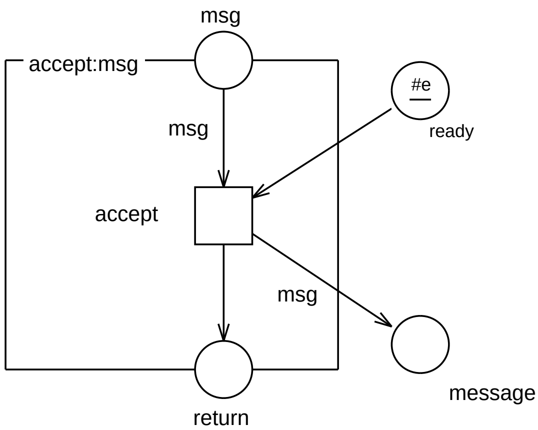

Fig. 8: Class Sender – method net accept

After accepting, message is placed to the place message from where it is send together with a control bit via the medium tm. Carbon copy of the message is put into the place sent where it is allowed for re-sending if the timeout signal occurs.

The Sender gets an acknowledgement by the transition get_ack. Correct bit is accepted by the transition accept_ack (after its firing is the Sender ready for accepting next message) and wrong bit is ignored by the transition drop_ack.

The place r4a (it means ready for acknowledgement) guarantees that in the place acknowledge_bit is at most one token at the time. It prevents loss of order of acknowledgements.

Note, adding of an control bit to a message is realized by mathematical operations on integers. Therefore all transmitted messages must be integers. It is caused by used implementation of PNtalk system which does not allow tuples as method parameters in a simple way. However, this restriction does not limit possibilities of formal analysis and verification of the model.

2.4 Class Receiver

The class Receiver models a part of ABP that is responsible for getting and acknowledging messages via the Medium and delivering them to the application.

The model of receiver consists of the object net and two method nets. First one is a non-implicit constructor trans: tm ack: am that allows the Application to set references to both mediums and second one is a method deliver that is a second service access point of ABP.

A message is put to the place got_msg after accepting it from the medium tm by the transition get_msg. The message can be then droped by the transition ingore_msg (if a control bit is unexpected) or accepted by the transition accept_msg (if a control bit is expected). An accepted message is delivered to the application and then an acknowledgement is sent by the transition acknowledge via the medium am to the Sender.

If the Receiver does not get a new message before the timeout occurs it will send the acknowledgement once more.

Note, the Receiver can fire the transition timeout and therefore it can also generate a false acknowledgement before receiving any message. It does not matter because the

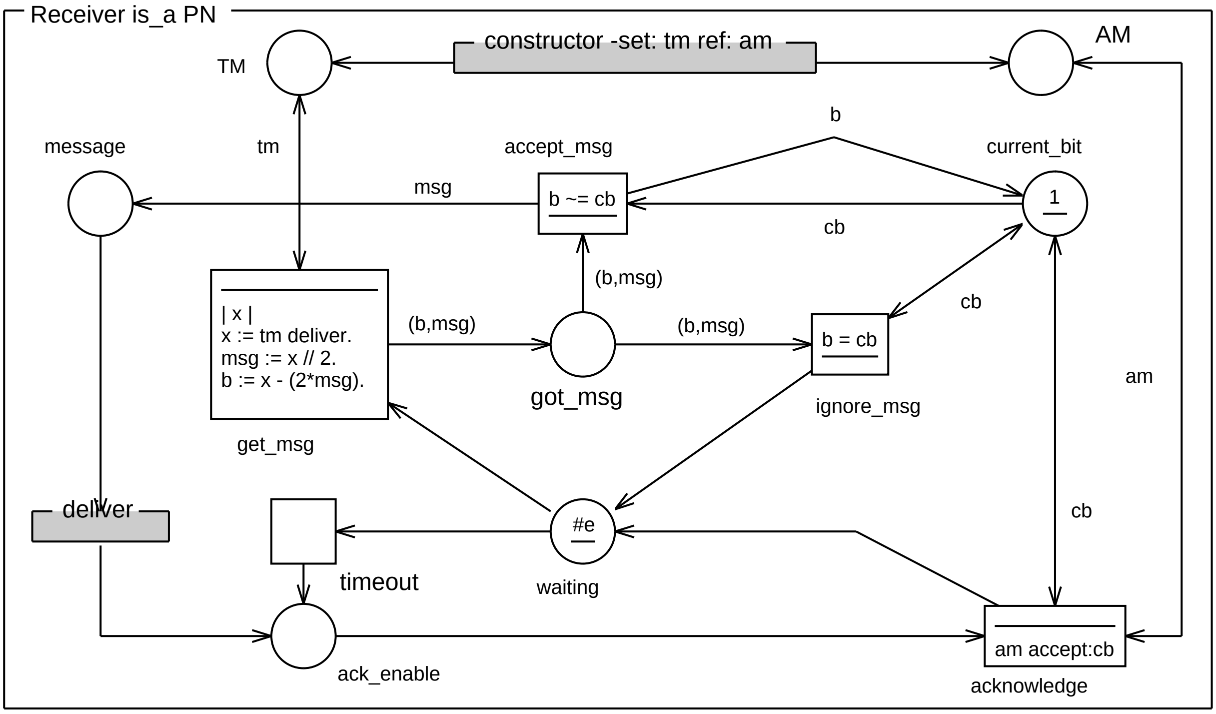

Fig. 9: Class Receiver – object net

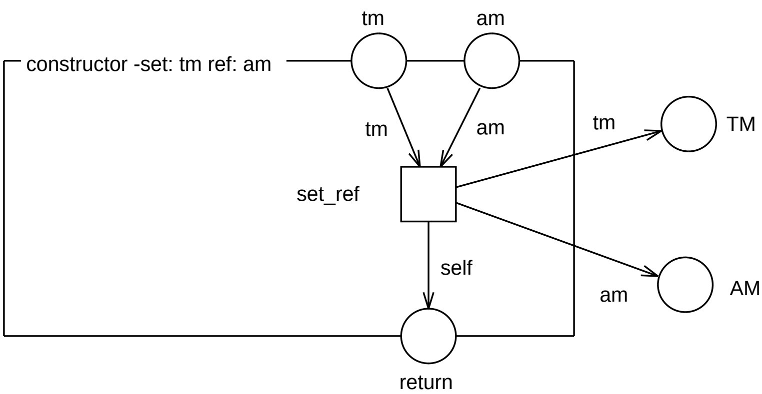

Fig. 10: Class Receiver – non-implicit constructor net

Sender ignores such acknowledgements. This feature can be eliminated by duplicating the transition get_msg for the first firing.

3 Conclusion

We have presented a model of the ABP in the PNtalk modelling language here. We are interested in modelling ABP by PNtalk because we need models for our research in the field of formal analysis and verification of OOPNs [5].

Used implementation of PNtalk system does not allow synchronous ports. If we use synchronous ports instead methods it could reduce size of a state space of a model.

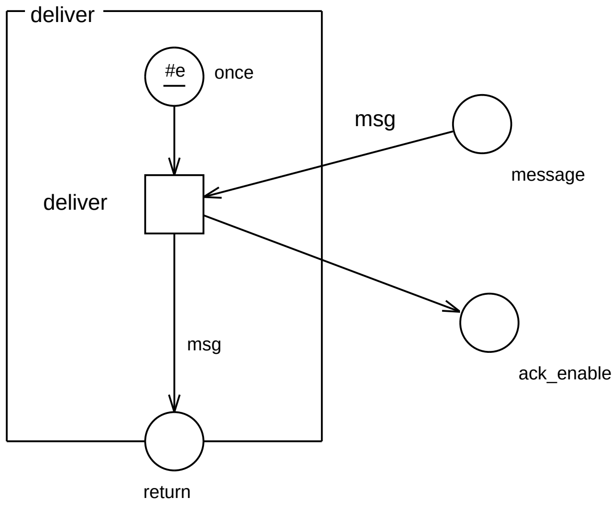

Fig. 11: Class Receiver – method net deliver

It is also possible to use this case study as a base for modelling some other communication protocols including real communication protocols as TCP-IP by PNtalk and/or OOPNs. This case study could be useful for investigation of methodology and tool support for model construction.

I thank to my supervisor Milan Češka and my colleagues Vladimír Janoušek and Tomáš Vojnar for their help and good commentary.

This work has been supported by the Ministry of Education of the Czech Republic within the project FR1948/2002/G1 "Metody formální analýzy a verifikace v objektově orientovaných Petriho sítích".

Bibliography

1. P. Abdulla, A. Bouajjani, B. Jonsson. On-the-Fly Analysis of Systems with Unbounded, Lossy Fifo Channels. in Proc. 10th Intern. Conf. on Computer Aided Verification (CAV'98), LNCS 1427, Vancouver (Canada), June 1998. URL: http://www.liafa.jussieu.fr/ abou/publis.html
2. L. Brim. Komunikace a paralelismus. Education material used within the course "Komunikace a paralelismus", Faculty of Informatics, Masaryk University, Brno, 2001.
3. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Vol. 1: Basic Concepts. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992.
4. V. Janoušek. *Modelling Objects by Petri Nets.* PhD thesis, Department of Computer Science and Engineering, FEECS, Brno University of Technology, Czech Republic, 1998.
5. T. Vojnar. *Towards Formal Analysis and Verification over State Spaces of Object-Oriented Petri Nets.* PhD thesis, Department of Computer Science and Engineering, FEECS, Brno University of Technology, Czech Republic, 2001.