Discussion Paper: PiHype Example v0.3

Frank Puhlmann
frank.puhlmann@hpi.uni-potsdam.de

June 22, 2004

Abstract

This short discussion paper tries to model a simple Petri net in the π-calculus. The challenge was given by Wil van der Aalst in a discussion paper [1].

1 Introduction

Wil van der Aalst wrote a discussion paper about the so called "Pi-Hype", introducing seven challenges to the π-calculus advocators [1]. In challenge four, he showed a Petri net, reprinted in figure 1, that in his opinion is hard to model by the use of process algebra. The jumping point is the milestone between c and f, forcing f to wait until both c and e have completed successfully.

He modeled the process without the milestone as $a.(b.c.d|e.f.g).h$. Actually, this is the notation of a classical basic process algebra with parallel extension. In advanced process algebra like CCS [3] or the π-calculus [7], the milestone turns out to be no problem.

It is notable that the π-calculus is based on mobile processes which are connected through communication links. The structure of the π-calculus is basically build on graphs. The difference to Petri nets lies in the ability to dynamically modify the link structure by the use of communication. This is why Robin Milner called his book about the π-calculus "Communicating and mobile systems" [6]. However, this is not even needed for this simple example. It can be used quite well if we want to model the case handling as shown later on.

To start with, I shortly summarize the notation of the π-calculus that we use in this paper:

Empty Process 0. The empty process **0** cannot perform any action; thereby it terminates the process.

Output Prefix $\overline{\mathbf{a}}\mathbf{x}$. The output prefix is used to send the name x along the name a. If the name x is unimportant (a acting as a trigger), than x can be omitted, just writing $\overline{a}$.

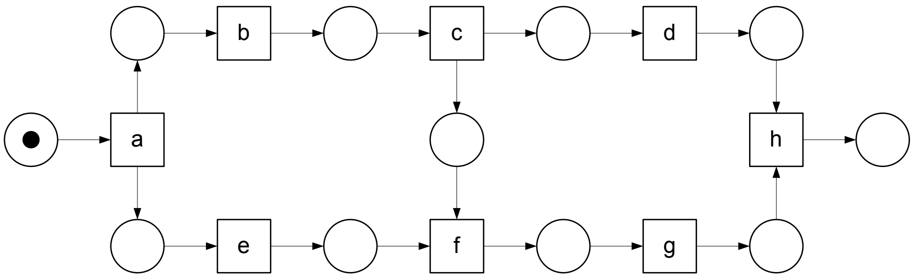

Figure 1: Petri net example from the Pi-Hype discussion paper [1] by Wil van der Aalst

Input Prefix a(x). The input prefix is used to receive the name x along the name a and then x is replaced by the received name. Note that this replacement is not important in the example; I name all received variables the same, but when correctly executing the calculus, they must be renamed because the input prefix creates a new locally bound name. If the name x is unimportant (a acting as a trigger), than x can be omitted, just writing a.

Silent Prefix τ. The silent prefix τ represents a process that does not interact with the environment.

Sum $\mathbf{P+Q}$. The sum represents a process that can either execute P or Q.

Parallel Composition $\mathbf{P \mid Q}$. The parallel composition represents a process that executes P and Q in parallel. P and Q can communicate if one performs an output and the other a corresponding input along the same name. For example, $\overline{a}x \mid a(x)$, represents a process where the left side sends the name x along a and the right side receives x along a.

Restriction $(\mathbf{v}x)\mathbf{P}$. The restriction or new operator $(\mathbf{v}x)P$ behaves as P but the name x is locally bound to P, so it can not be used as a global (free) name anymore. Only P has knowledge of x, but can share this knowledge by the use of communication (called scope extrusion).

2 Petri net example

We can model the static Petri net from the example given in [1] (see figure 1) as an instance oriented process view, mapping the transition to processes as shown in figure 2. We initially omit the link between C and F. Every transition, that does the work in a classical workflow net, is now shown as a separate process, labeled from A to H. Every of those processes represent an activity in the workflow domain. The processes are very simple. A contains an AND–split, B to G just receive a link from its predecessors and after performing some silent actions, the activity, they call their successor. H contains an AND–join. By the notation of the π-calculus the abstract process can be written as:

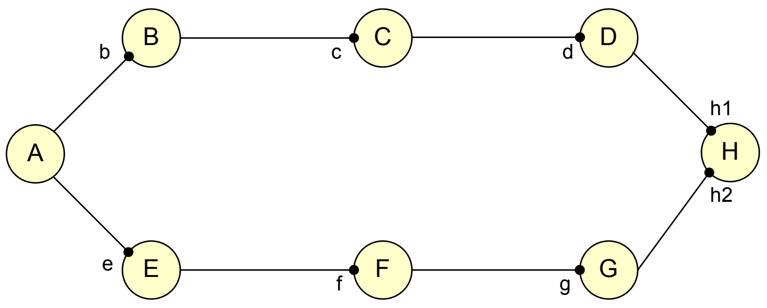

Figure 2: Simple example as a process view

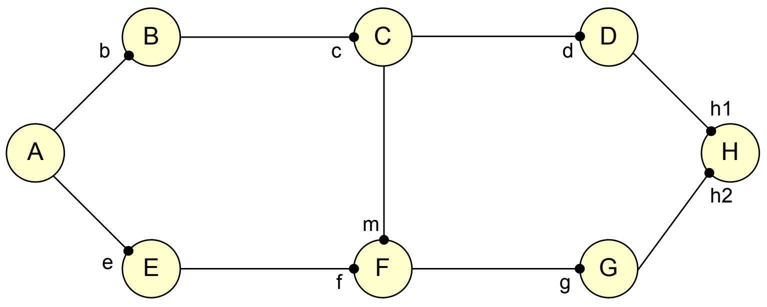

Figure 3: Complete example with milestone

$$PN = A \mid B \mid C \mid D \mid E \mid F \mid G \mid H$$

All processes (here representing activities) are defined initially as concurrent. The control flow is handled by each process itself. Process A invokes in parallel two links enabling B and E to start their work. After finishing, they enable C and F, which in continuation enable D and G. H acts as an AND–join, waiting on two names. The following equations contain the missing process definitions in π-calculus notation:

$$A = \overline{b}.\mathbf{0} \mid \overline{e}.\mathbf{0} \quad \begin{array}{lll} B = b.\overline{c}.\mathbf{0} & C = c.\overline{d}.\mathbf{0} & D = d.\overline{h1}.\mathbf{0} \\ E = e.\overline{f}.\mathbf{0} & F = f.\overline{g}.\mathbf{0} & G = g.\overline{h2}.\mathbf{0} \end{array} \quad H = h1.h2.\mathbf{0}$$

Now we can add the missing link between C and F. Because communication occurs synchronous in the π-calculus, we need to keep C running until F is ready to go on. The only task C continuous to do, after performing the activity, is to send a name m to F. This is shown in figure 3.

Process C now acts as a milestone, keeping the information that the work of C has finished. F must now act as an and–join, waiting for f and m. The updated, complete process description is as follows:

$$A = \overline{b}.\mathbf{0} \mid \overline{e}.\mathbf{0} \quad \begin{array}{lll} B = b.\overline{c}.\mathbf{0} & C = c.(\overline{d}.\mathbf{0} \mid \overline{m}.\mathbf{0}) & D = d.\overline{h1}.\mathbf{0} \\ E = e.\overline{f}.\mathbf{0} & F = f.m.\overline{g}.\mathbf{0} & G = g.\overline{h2}.\mathbf{0} \end{array} \quad H = h1.h2.\mathbf{0}$$

We can combine the process descriptions in one line, thereby the empty processes $\mathbf{0}$ is omitted for space reasons:

$$PN = (\overline{b} \mid \overline{e}) \mid b.\overline{c} \mid c.(\overline{d} \mid \overline{m}) \mid d.\overline{h1} \mid e.\overline{f} \mid f.m.\overline{g} \mid g.\overline{h2} \mid h1.h2$$

We can also prove, by using reduction, that this process always terminates:

$$(\overline{b} \mid \overline{e}) \mid b.\overline{c} \mid c.(\overline{d} \mid \overline{m}) \mid d.\overline{h1} \mid e.\overline{f} \mid f.m.\overline{g} \mid g.\overline{h2} \mid h1.h2$$

$$\overset{\overline{b},\overline{e}}{\Rightarrow}$$

$$\mathbf{0} \mid \overline{c} \mid c.(\overline{d} \mid \overline{m}) \mid d.\overline{h1} \mid \overline{f} \mid f.m.\overline{g} \mid g.\overline{h2} \mid h1.h2$$

$$\overset{\overline{c},\overline{f}}{\Rightarrow}$$

$$\mathbf{0} \mid \mathbf{0} \mid (\overline{d} \mid \overline{m}) \mid d.\overline{h1} \mid \mathbf{0} \mid m.\overline{g} \mid g.\overline{h2} \mid h1.h2$$

$$\overset{\overline{d},\overline{m}}{\Rightarrow}$$

$$\mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \overline{h1} \mid \mathbf{0} \mid \overline{g} \mid g.\overline{h2} \mid h1.h2$$

$$\overset{\overline{h1},\overline{g}}{\Rightarrow}$$

$$\mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \overline{h2} \mid h2$$

$$\overset{\overline{h2}}{\Rightarrow}$$

$$\mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0}$$

$$\Rightarrow$$

$$\mathbf{0}$$

Exercise for the interested reader: Give the formal description of the Petri net and then prove soundness as defined in [10].

3 Case handling

We could go on further by considering the Petri net example as a workflow net. The token contained in the input place of transition a can then be a case. This case (the token) is cloned at the the transition a and moved to the input places of b and e. This process continuous until the tokens reach the transition h, get merged and the merged token is finally moved to the output place of h (the sink). This is where the case is completed.

Now we can model the same in a process oriented view. Thereby the first process, A shares the name x of a case. This name is initially bound to a process X, representing a case (along with the case data), shown in figure 1a. Now that A has knowledge of x, A can work on the case and after finishing the activity, sends the name x to B and E. The scope of x is extruded, so that B and E can access the case process X. Since A has come to inaction, it has no further access to x. This is shown in figure 4b. After completing their work, B and E send x to their successors and so on (figure 4c).

This can be written in the π-calculus notation as follows, we again omit the empty processes $\mathbf{0}$ for space reasons:

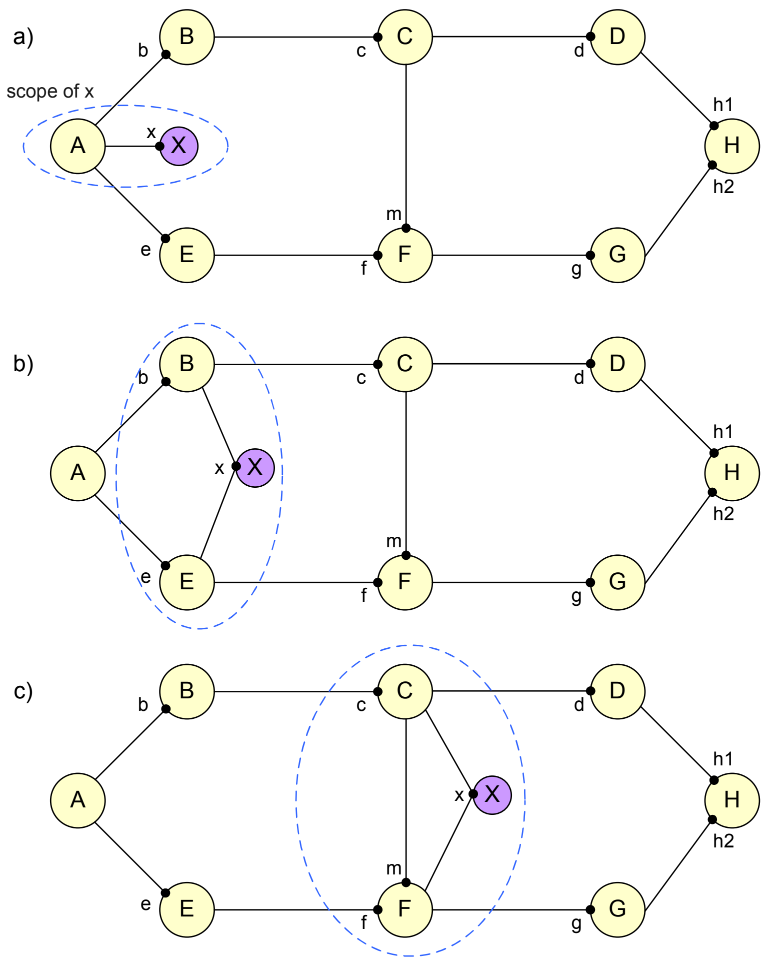

Figure 4: Complete example with a case moving through the activities

$$A = \bar{b}x \mid \bar{e}x \qquad \begin{array}{l} B = b(x).\bar{c}x \\ E = e(x).\bar{f}x \end{array} \quad \begin{array}{l} C = c(x).(\bar{d}x \mid \overline{m}) \\ F = f(x).m.\bar{g}x \end{array} \quad \begin{array}{l} D = d(x).\overline{h1}x \\ G = g(x).\overline{h2} \end{array} \quad H = h1(x).h2$$

Note that in D and G it is sufficient if only D transmits x to H, because it represents a link to the case. H can only start if D and G have triggered it, therewith D must have been transmitted x.

The process with a new case X can than be defined as follows:

$$PN = (\mathbf{v}x)(X \mid A) \mid B \mid C \mid D \mid E \mid F \mid G \mid H$$

The case process X "moves" through the workflow, by extending the web of links between the processes. Every process that performs an activity is guaranteed to work on the same current data of the case. The access must be controlled by the case process X, that is not considered here.

4 Conclusion

I must admit, that I'm not an advocator for anything. That's mostly because I only have very basic knowledge about Petri nets and the π-calculus. I have currently started to look around of how to model processes, esp. in the e-business domain. Thereby I came along a lot of topics like procedural languages, rule based process descriptions, graph grammars, intelligent agents, process algebra and of course workflow nets.

While searching for some process algebra related stuff, I came across the discussion paper from Wil van der Aalst. I thought a bit about the challenges and came to the conclusion that the Petri net challenge misses the π-calculus. That's why I searched for answers, but finally Wil van der Aalst mailed me, that there is no response yet. So I wrote down my thoughts in this discussion paper, hopefully to get some comments on my ideas.

The more interesting questions lie of course in the benefits of the π-calculus over other approaches. Maybe the mobile part might be helpful in the e–business domain, where there is a rapidly changing, unpredictable and open environment with a significant possibility that actions can fail in an unpredictable way (as often used by the agent advocators). Still, in all earlier documentation of the π-calculus the processes are called agents.

Further readings

I conclude this discussion paper with some literature I've read on process algebra.

- The foundations of process algebra can be found in [2].
- In his Turing Award lecture, Robin Milner [4] draws the path from sequential over communication up to mobile process algebra.
- The π–Calculus was early described in [7]. The first part contains easy to follow examples, whereas the second part contains more formal descriptions.
- Robin Milner wrote a tutorial [5] and a single book [6] about the π–Calculus. The book contains a nice introduction to automata theory and labeled transition systems as well as good examples. Beside the introduction, the key concepts are difficult to read and comprehend. The earlier papers mentioned above are much easier to understand.
- Another good introduction to the π-Calculus can be found in [8].
- The current state of the art regarding the π-calculus can be found in [9]. The authors see their book as a natural complement to Milner's book.

References

[1] W.M.P. van der Aalst. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype". `http://tmitwww.tm.tue.nl/research/patterns/download/pi-hype.pdf` (April 8, 2004).

[2] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1990.

[3] Robin Milner. *Communication and Concurrency.* Prentice Hall, New York, 1989.

[4] Robin Milner. Elements of Interaction. Turing Award Lecture. *Communications of the ACM*, 36:78–89, January 1993.

[5] Robin Milner. The polyadic π–Calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246, Berlin, 1993. Springer-Verlag.

[6] Robin Milner. *Communicating and Mobile Systems: The π-calculus.* Cambridge University Press, Cambridge, 1999.

[7] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part I/II. *Information and Computation*, 100:1–77, September 1992.

[8] Joachim Parrow. An Introduction to the π–Calculus. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[9] Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes.* Cambridge University Press, Cambridge, paperback edition, 2003.

[10] Wil van der Aalst and Kees van Hee. *Workflow Management.* MIT Press, 2002.