

Using the Pi-Calculus for Formalizing Workflow Patterns

Frank Puhlmann

Hasso-Plattner-Institute at the University of Potsdam

<http://bpt.hpi.uni-potsdam.de>

(Joint work with Mathias Weske)

Outline

- Introduction
- The Pi-Calculus
- Pattern Representation
 - Basic Control Flow Patterns
 - Advanced Workflow Pattern:
 - Multiple Merge
 - Discriminator
- Conclusion

Introduction

- Motivation
 - The pi-calculus, a process algebra, has been discussed as the formal foundation for workflow (e.g. The Third Wave, PiHype)
 - However, no formal investigations on the capabilities of the pi-calculus regarding the workflow domain have been made so far
- Goal
 - Show the capabilities of the pi-calculus to describe the behavioral perspective of workflow
 - Investigate the representation of Workflow Patterns in the pi-calculus

The Pi-Calculus

- The pi-calculus consists of names and processes:
 - Names represent existing concepts like links, pointers, references, identifiers, etc.
 - Each name has a scope

- Processes are defined as:

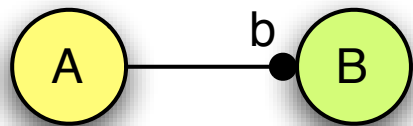
$$P ::= M \mid P|P' \mid \nu zP \mid !P$$

- The summations:

$$M ::= \mathbf{0} \mid \pi.P \mid M + M'$$

- And the prefixes:

$$\pi ::= \bar{x}\langle y \rangle \mid x(z) \mid \tau \mid [x = y]\pi$$



$$A = \bar{b}\langle x \rangle.\mathbf{0} \quad B = b(x).\tau_B.\mathbf{0} \quad P = A|B$$

Basic Control Flow Patterns I

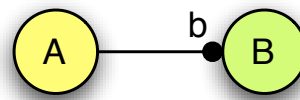
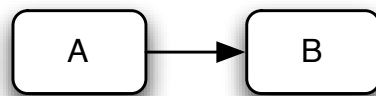
- Each workflow activity is mapped to a pi-calculus process with pre- and postconditions
- Event-based Workflow routing:

Precondition
=
Wait for Event

$$x.\tau.\bar{y}.\mathbf{0}$$

Postcondition
=
Generate Event

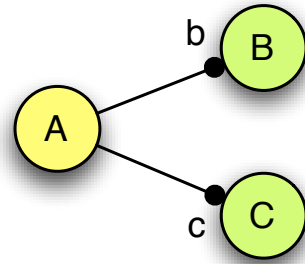
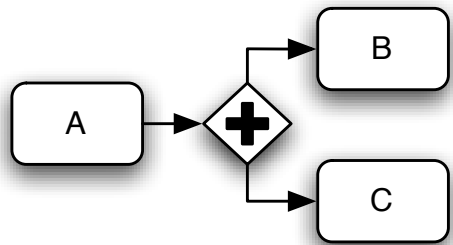
Sequence



$$A = \tau_A.\bar{b}.\mathbf{0}$$
$$B = b.\tau_B.B'$$

Basic Control Flow Patterns 2

Parallel Split

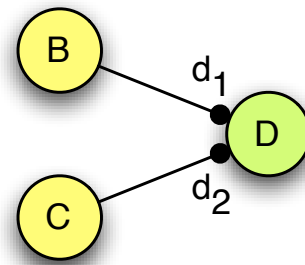
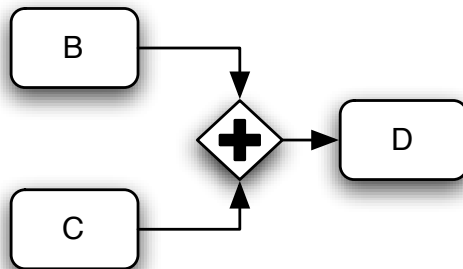


$$A = \tau_a.(\bar{b}.0|\bar{c}.0)$$

$$B = b.\tau_B.B'$$

$$C = c.\tau_C.C'$$

Synchronization



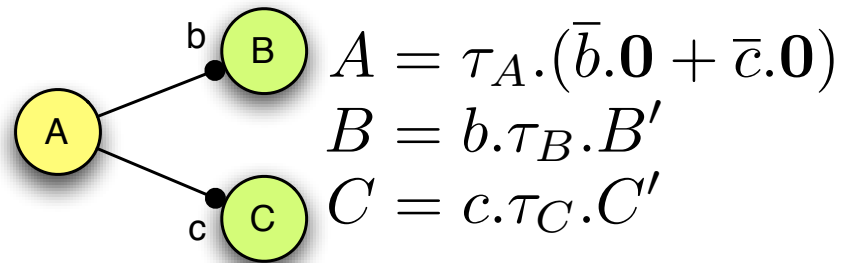
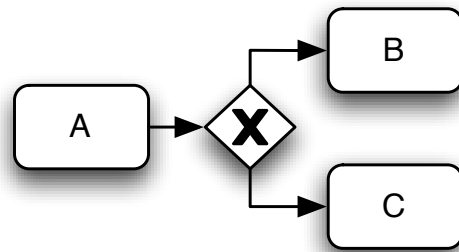
$$B = \tau_B.\bar{d}_1.0$$

$$C = \tau_C.\bar{d}_2.0$$

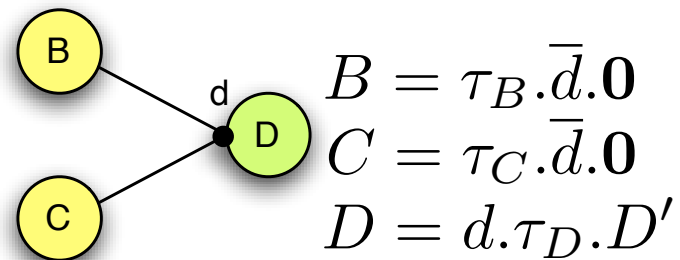
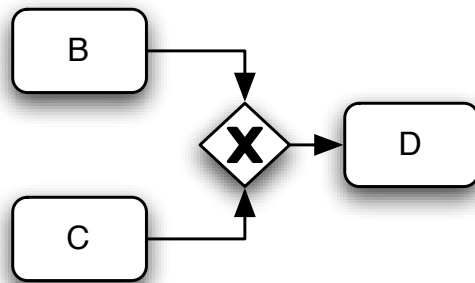
$$D = d_1.d_2.\tau_D.D'$$

Basic Control Flow Patterns 3

Exclusive Choice

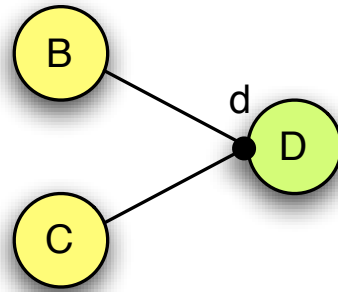


Simple Merge



Multiple Merge

- The Multiple Merge activates D each time one of the incoming branches are activated



$$B = \tau_B.\bar{d}.0$$

$$C = \tau_C.\bar{d}.0$$

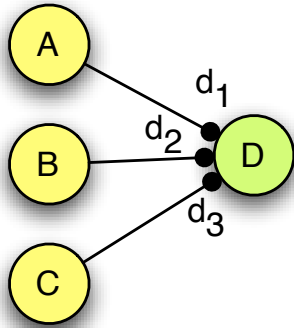
$$D = !d.\tau_D.D'$$

Replication in the Pi-Calculus

$$\text{Initial: } !d.\tau_d.D'$$

$$\text{After first invocation: } \tau_d.D' \mid !d.\tau_D.D'$$

Discriminator



$$A = \tau_A.\bar{d}_1.\mathbf{0} \quad B = \tau_B.\bar{d}_2.\mathbf{0} \quad C = \tau_C.\bar{d}_3.\mathbf{0}$$

$$D = (\mathbf{vh}, exec)(D_1|D_2)$$

$$D_1 = d_1.\bar{h}.\mathbf{0} \mid d_2.\bar{h}.\mathbf{0} \mid d_3.\bar{h}.\mathbf{0}$$

$$D_2 = h.\overline{exec}.h.h.D \mid exec.\tau_D.D'$$

Discriminator Template (I-out-of-M-join):

$$D = (\mathbf{vh}, exec)\left(\left(\prod_{i=1}^m d_i.\bar{h}.\mathbf{0}\right) \mid h.\overline{exec}.\{h\}_1^{m-1}.D \mid exec.\tau_D.D'\right)$$

N-out-of-M-join:

$$D = (\mathbf{vh}, exec)\left(\left(\prod_{i=1}^m d_i.\bar{h}.\mathbf{0}\right) \mid \{h\}_1^n.\overline{exec}.\{h\}_{n+1}^m.D \mid exec.\tau_D.D'\right)$$

Conclusion

- Our work shows that all Workflow Patterns are directly representable in the pi-calculus
- Advanced features of the pi-calculus are required for the more complex patterns
- The results additionally provide a formal semantics for the Workflow Patterns, making them concise and unambiguous
- Future research can be based upon these results, e.g.
 - Formal foundation for (graphical) workflow notations
 - Tools for executing pi-calculus workflows

Further Information

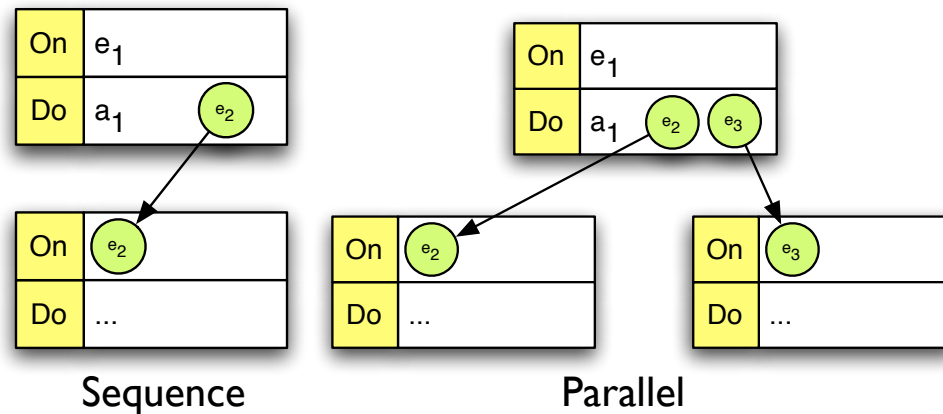
- The interactive Pi-Workflow Website at:
<http://pi-workflow.org>
- Based on SnipSnap, a Wiki and Weblog software
- Everyone is invited to register and comment the content of the site
- The weblog will contain news about the research progress

Thank you!

Questions?

ECA Mapping

- Each workflow activity is mapped to a pi-calculus process with pre- and postconditions
- Event-based Workflow routing:



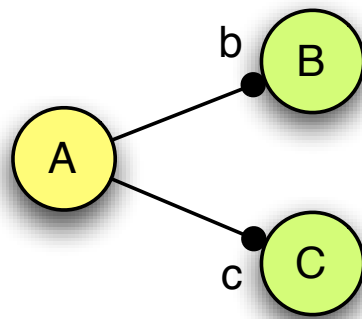
Precondition
=
Event and Condition

$$x.[a = b]\tau.\bar{y}.0$$

Postcondition
=
Action

Multiple Choice

- How does the Multi-Choice Pattern work?



$$A = (\mathbf{vexec})\tau_A.(A_1|A_2)$$

$$A_1 = \overline{exec}\langle b \rangle.\mathbf{0} +$$

$$\overline{exec}\langle c \rangle.\mathbf{0} +$$

$$\overline{exec}\langle b \rangle.\overline{exec}\langle c \rangle.\mathbf{0}$$

$$A_2 = !exec(x).\bar{x}.\mathbf{0}$$

$$B = b.\tau_B.B'$$

$$C = c.\tau_C.C'$$

OR-Joins

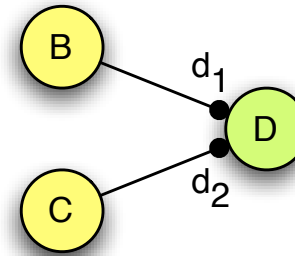
- How is the synchronizing merge handled?

- Pattern representation:

$$B = \tau_B.\overline{d_1}.\mathbf{0}$$

$$C = \tau_C.\overline{d_2}.\mathbf{0}$$

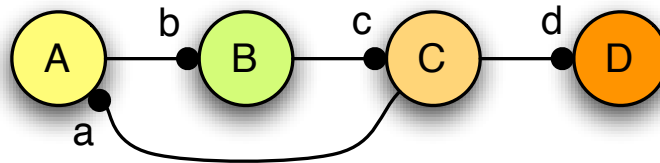
$$D = d_1.\tau_D.D' + d_2.\tau_D.D' + d_1.d_2.\tau_D.D'$$



- This pattern does not define how a runtime actually selects a summation; it just denotes all possibilities!
- A runtime could use different strategies, e.g.:
 - True/False token passing
 - Postphoned OR-join (YAWL like)

Arbitrary Cycles

- How do arbitrary cycles work?



$$A = !a.\tau_A.\bar{b}.\mathbf{0}$$

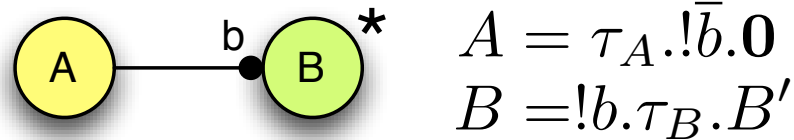
$$B = !b.\tau_B.\bar{c}.\mathbf{0}$$

$$C = !c.\tau_C.(\bar{a}.\mathbf{0} + \bar{d}.\mathbf{0})$$

$$D = d.\tau_D.D'$$

MI without Synchronization

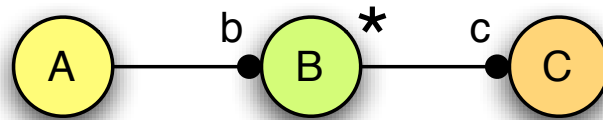
- Process A can spawn of any amount of multiple instances of a process B. No synchronization is required:



$$A = \tau_A.!\bar{b}.0$$
$$B = !b.\tau_B.B'$$

MI with a priori Design Time Knowledge

- Process A spawns of a design time known number of instances of B that have to be synchronized afterwards:



$$A = \tau_A.\bar{b}.\bar{b}.\bar{b}.\mathbf{0}$$

$$B = !b.\tau_B.\bar{c}.\mathbf{0}$$

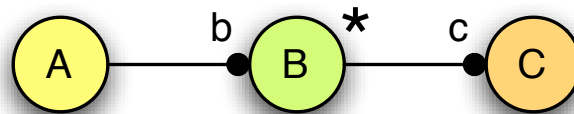
$$C = c.c.c.\tau_C.C'$$

- For n design time copies the pattern is:

$$A \mid B \mid C \equiv \tau_A.\{\bar{b}\}_1^n.\mathbf{0} \mid !b.\tau_B.\bar{c}.\mathbf{0} \mid \{c\}_1^n.\tau_C.C'$$

MI with a priori Runtime Knowledge

- A process A can spawn a runtime known number of instances of B that are started after all copies have been created. The copies of B have to be synchronized before another process C is activated:



$$A = (\mathbf{v}run)\tau_A.A_1(c) \mid run.\overline{start}.0$$

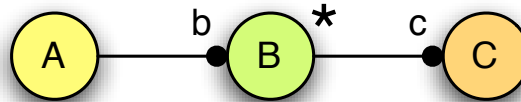
$$A_1(x) = (\mathbf{v}y)\bar{b}\langle y\rangle.y\langle x\rangle.A_1(y) + \overline{run}.\bar{x}.0$$

$$B = !b(y).y(x).start.\tau_B.y.\bar{x}.0$$

$$C = c.\tau_C.C'$$

MI with no priori knowledge

- Process A spawns multiple instances of another process B that have to be synchronized before the execution of C:



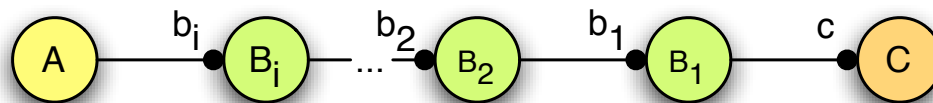
$$A = \tau_A.A_1(c)$$

$$A_1(x) = (\nu y)\bar{b}\langle y \rangle.y\langle x \rangle.A_1(y) + \bar{x}.0$$

$$B = !b(y).y(x).\tau_B.y.\bar{x}.0$$

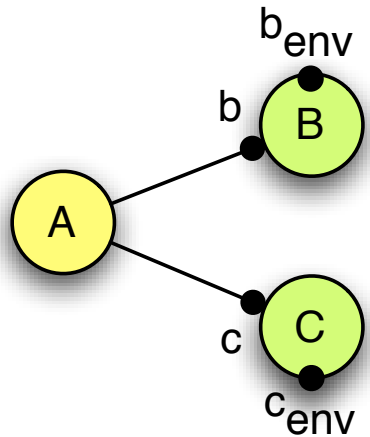
$$C = c.\tau_C.C'$$

The pattern works like a dynamic linked-list:



Deferred Choice

- How does the Deferred Choice Pattern work?



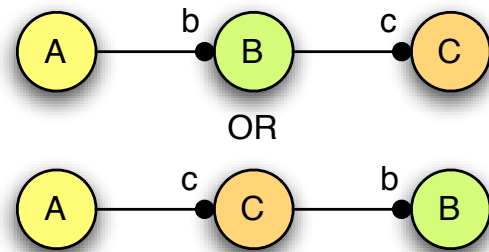
$$A = \tau_A.(\bar{b}.0 + \bar{c}.0)$$

$$B = b.(b_{env}.\overline{kill}.\tau_B.B' + kill.0)$$

$$C = c.(c_{env}.\overline{kill}.\tau_C.C' + kill.0)$$

Interleaved Parallel Routing

- How does the Interleaved Parallel Routing Pattern work?



$$A = \tau_A.\bar{x}.y.\bar{x}.y.A'$$

$$B = x.\tau_B.\bar{y}.0$$

$$C = x.\tau_C.\bar{y}.0$$

Milestone

- One possible representation of the Milestone Pattern:

$$A = \text{check}(x).([\text{x} = \top]\tau_{A1}.A' + [\text{x} = \perp]\tau_{A2}.A'')$$

$$B = M(\perp) \mid b.\bar{m} \langle \top \rangle .\tau_B.\bar{m} \langle \perp \rangle .B'$$

$$M(x) = m(x).M(x) + \overline{\text{check}} \langle x \rangle .M(x)$$

Cancel Patterns

- How does the Cancel Activity Pattern work?

$$A \mid \mathcal{E} \equiv a.\tau_A.A' + cancel.\mathbf{0} \mid !\tau_{\mathcal{E}}.\overline{cancel}.\mathbf{0}$$