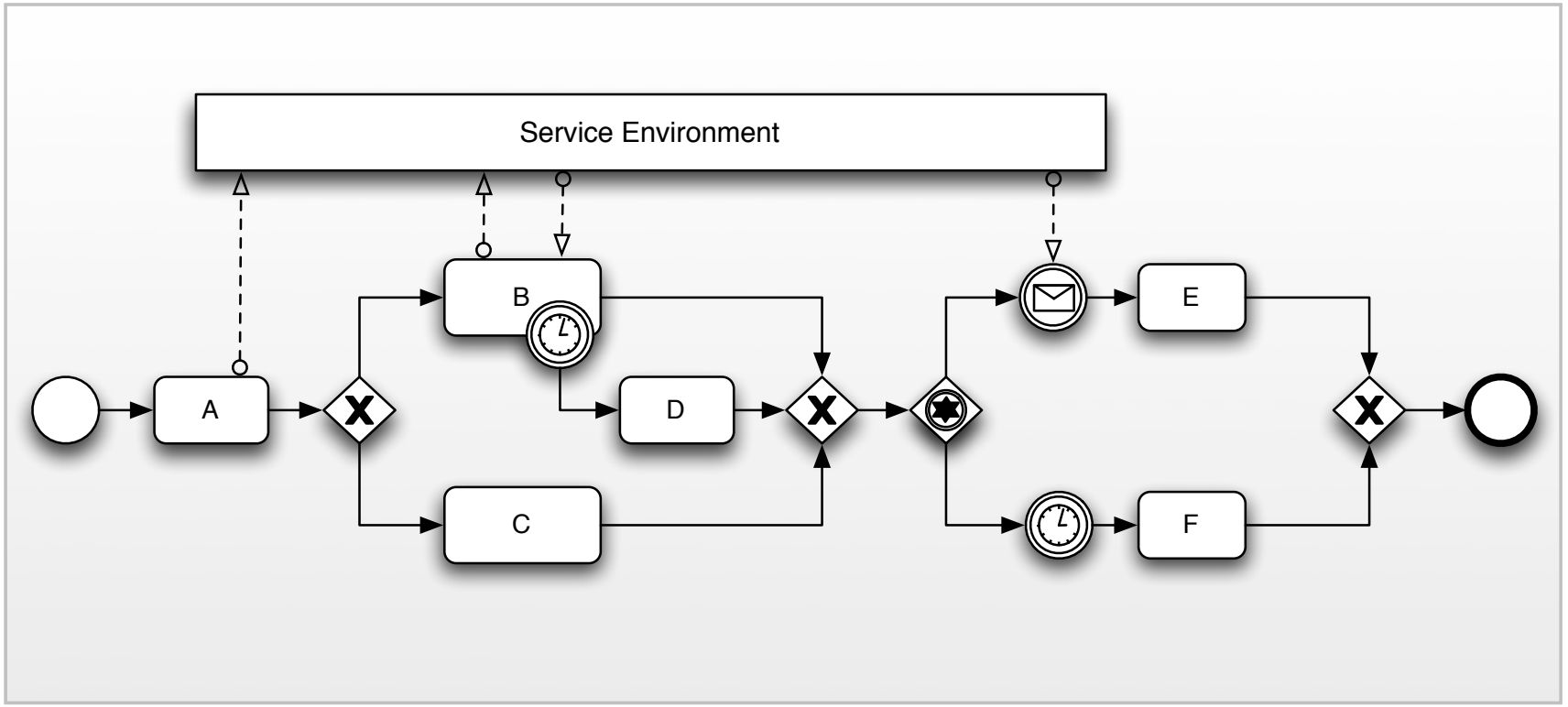
Towards a Formal Model for Agile Service Discovery and Integration

Frank Puhlmann, Hagen Overdick, Mathias Weske
Hasso-Plattner-Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany

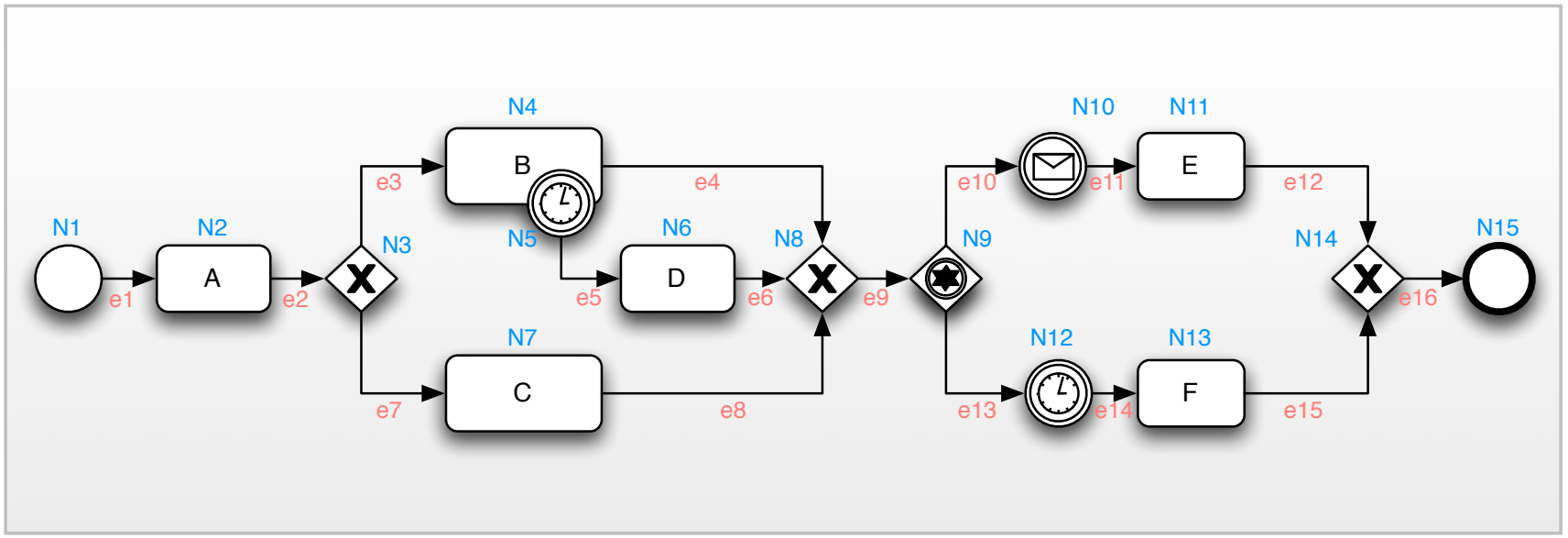
Goals

- To enables agile service discovery and invocation
- Formal Representation of
  - orchestrations as well as
  - choreographies
- Using a process algebra that provides dynamic process structures ⇒ Pi-Calculus
- Extend work on representing workflow pattern formalizations to the SOC domain



Example in BPMN

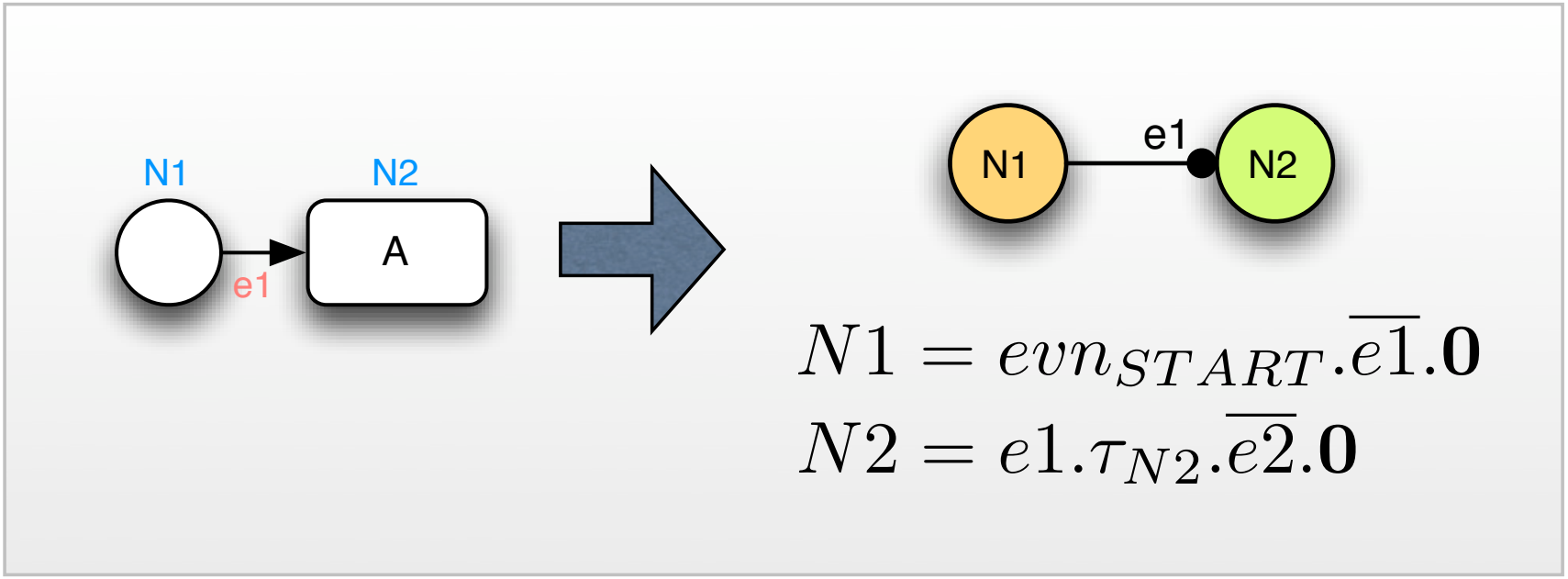
Orchestration



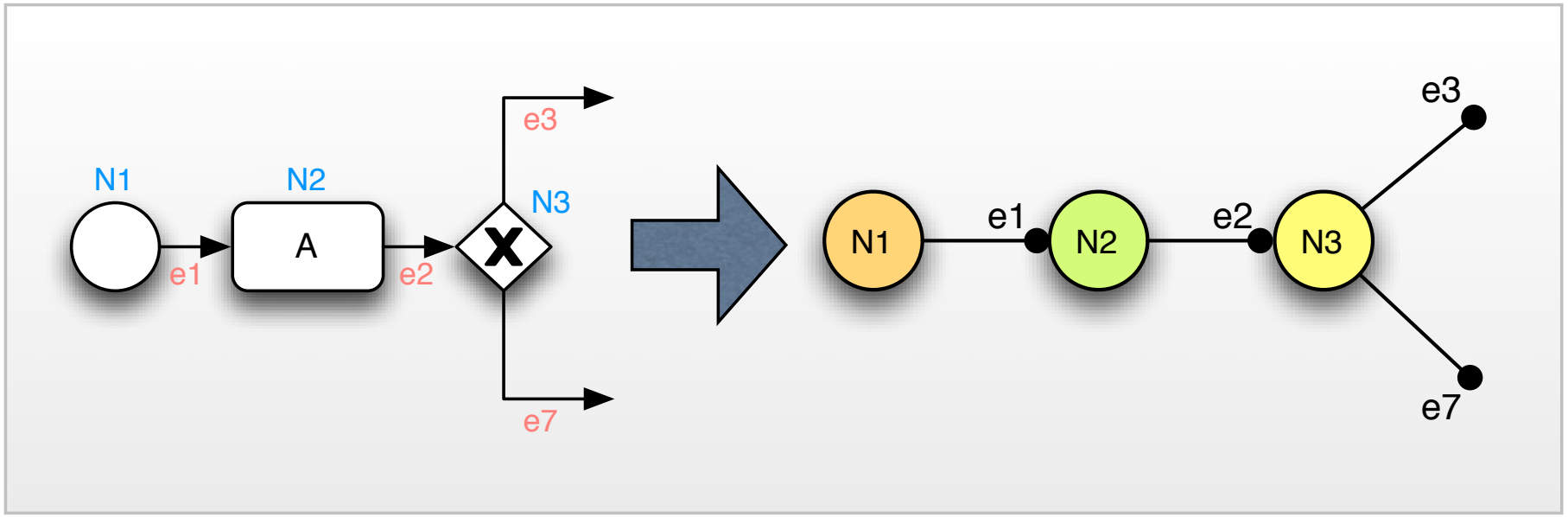
Mapping to the Pi-Calculus

Mapping strategy

- Match all (workflow) patterns to their corresponding Pi-Calculus representation
- Paper: Using the Pi-Calculus for Formalizing Workflow Patterns (Puhlmann, Weske)


N1
N2
A
e1
N1
e1
N2
$N1 = evn_{START}.\overline{e1}.\mathbf{0}$
$N2 = e1.\tau_{N2}.\overline{e2}.\mathbf{0}$

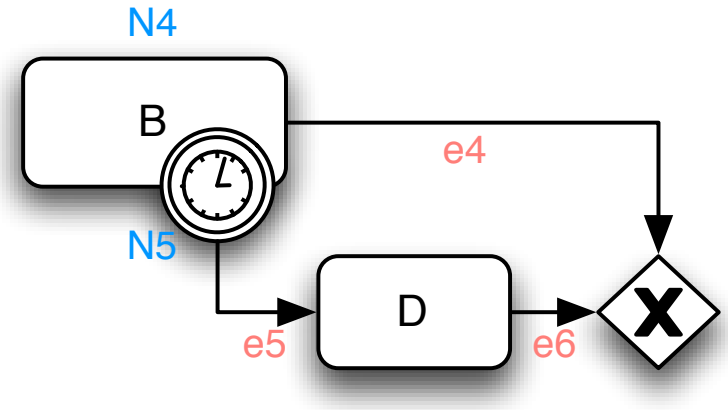
Sequence



Exclusive Choice

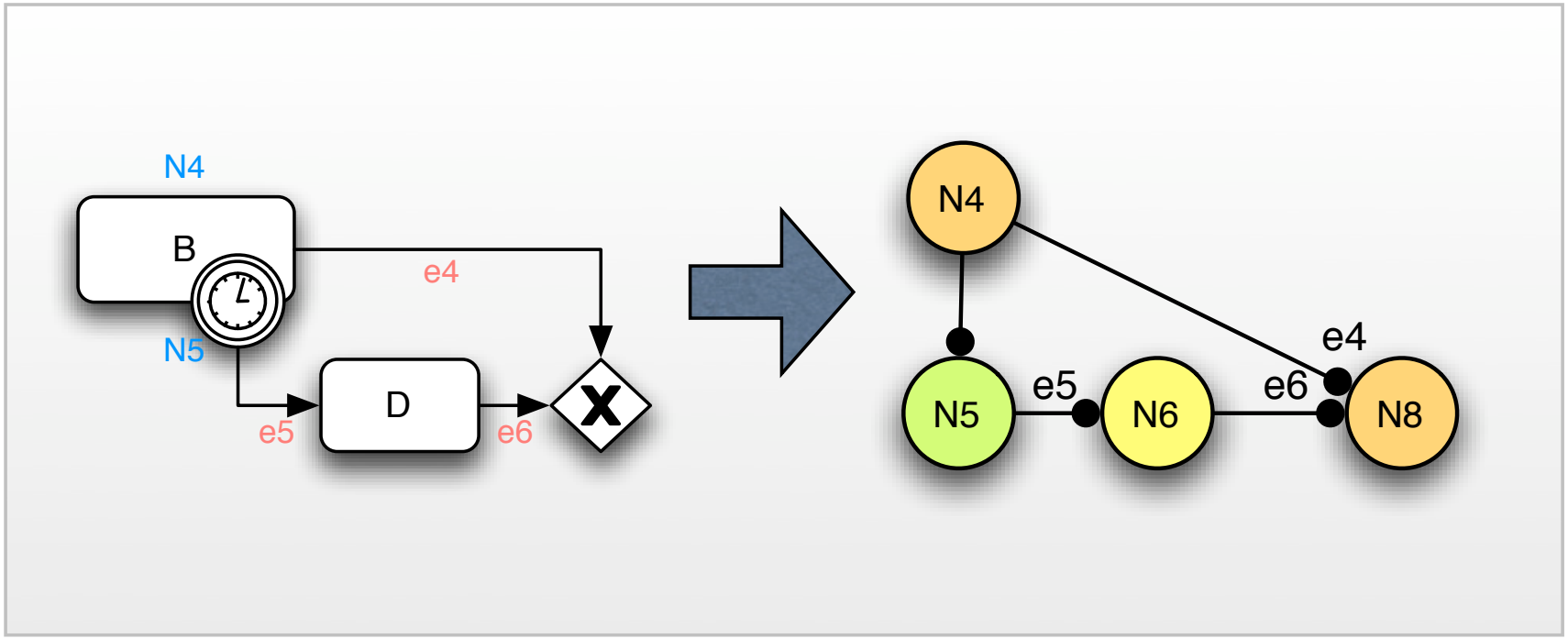
Hold on!

- What about this construct?



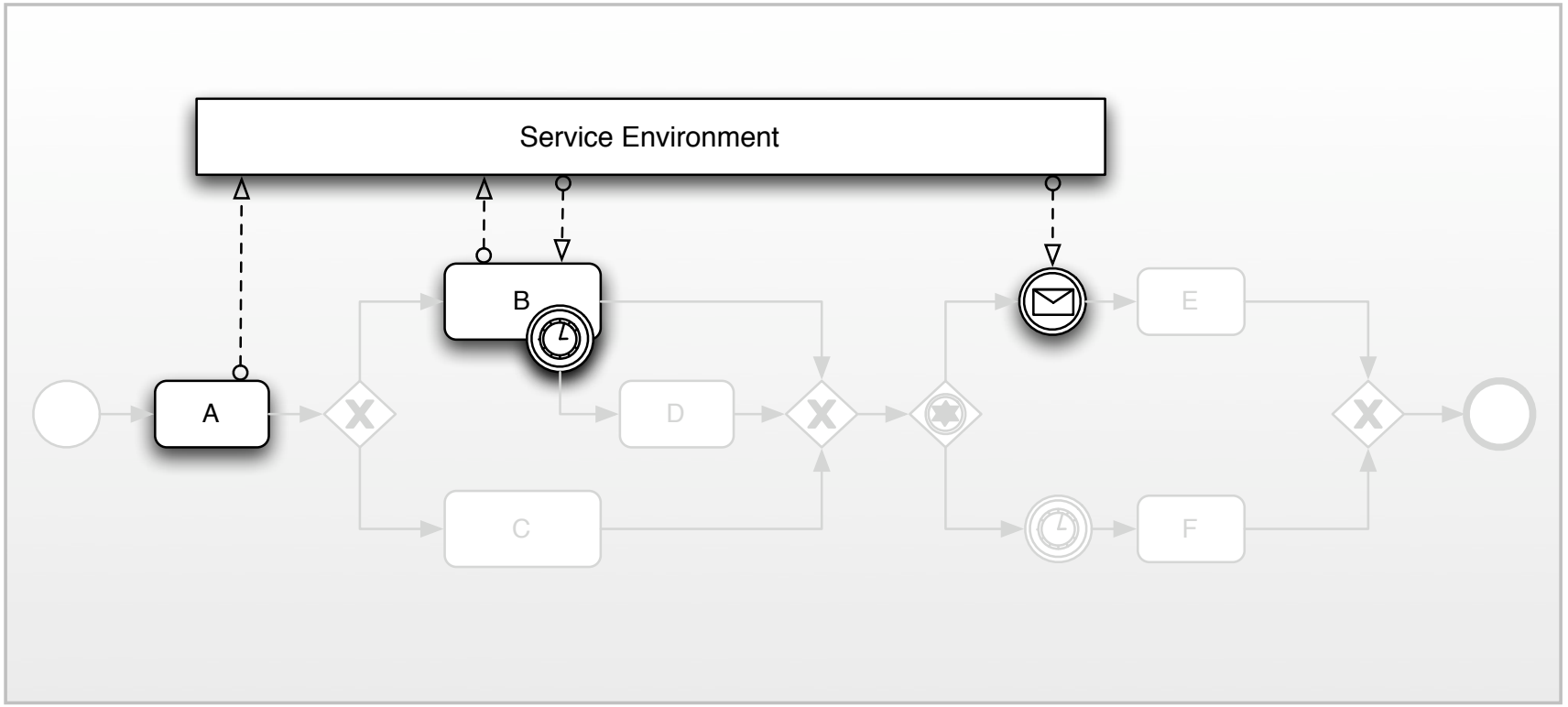
Event-based Rerouting

- New pattern!
- Represents the change of the control flow based on an event (e.g. a message) that occurs during the execution of an activity
- However, we can not stop the actual activity, only reroute the control flow immediately



Event-based Rerouting

Choreography



Choreography

Correlations

- Pi-calculus supports the concepts of dynamic channel creation via the new operator (**ν**)
- Each channel provides:
  - a unique identification and
  - a communication channel

➡ Sufficient for correlations!

Service Invocation

- Algorithm:
  - Create a new channel (a Pi-Calculus name)
  - Invoke the server with the channel and request as a parameter
  - Wait for a response on the link
- Advantage: The link can be used as an identifier as well as an unique response channel, so no additional formalism or handling is required

$$\mathbf{v}ch\ \overline{service}\langle ch, request\rangle.ch(response)$$

Formal Service Invocation

Asynchronous/ Synchronous Invocation

- The introduced concept is always asynchronous:
  - Request/Response can be split over different Pi-Calculus processes
  - If there are no operations between the request and response, we can call it “synchronous invocation”

$$
\begin{aligned}
N1 =& TASK(env_{START}, e1, env_{ABORT}, \tau_{N1}) \\
N2 =& TASK(e1, e2, env_{ABORT}, \overline{w_{req1}}\langle w_{resp1}\rangle.\tau_{N2}) \\
N3 =& e2.\tau_{N3}.([c_{e3} = \top]\overline{e3}|[c_{e7} = \top]\overline{e7}) \\
N4 =& TASK(e3, e4, abort_{N5}, \overline{w_{req2}}\langle w_{resp2}\rangle.w_{resp2}.\tau_{N4}).\mathbf{0} \\
N5 =& env_{TIMEOUTN5}.\overline{abort_{N5}}\langle e5\rangle \\
N6 =& TASK(e5, e6, env_{ABORT}, \tau_{N6}) \\
N7 =& TASK(e7, e8, env_{ABORT}, \tau_{N7}) \\
N8 =& \mathbf{v}x(e4.\overline{x}.\mathbf{0}|e6.\overline{x}.\mathbf{0}|e8.\overline{x}.\mathbf{0}|x.\overline{e9}.\mathbf{0}) \\
CHOICE_{N9,N10,N12} =& e9.(w_{resp2}.\overline{e11}.\mathbf{0} + env_{TIMEOUTN12}.\overline{e14}.\mathbf{0}) \\
N11 =& TASK(e13, e14, env_{ABORT}, \tau_{N11}) \\
N13 =& TASK(e14, e15, env_{ABORT}, \tau_{N13}) \\
N14 =& \mathbf{v}x(e12.\overline{x}.\mathbf{0}|e15.\overline{x}.\mathbf{0}|x.\overline{e16}.\mathbf{0}) \\
N15 =& TASK(e16, env_{DONE}, env_{ABORT}, \tau_{N15})
\end{aligned}
$$

Formal Representation

Conclusion & Further Work

- We (partly and illustrating) showed how to use the Pi-Calculus in the service-oriented domain
- The Pi-Calculus has its pro's in supporting
  - Choreographies (esp. correlations)
  - while also allowing powerful service orchestrations incl. extensions to new patterns
- However, more research is required, e.g.
  - formal reasoning (service equivalence, soundness)
  - or precise mappings from graphical notations

Want more?

- http://pi-workflow.org