

# Efficient Analysis of BPEL 2.0 Processes using $\pi$ -Calculus

## Example Processes

Matthias Weidlich, Gero Decker  
Hasso-Plattner-Institute, University of Potsdam, Germany  
{matthias.weidlich, gero.decker}@hpi.uni-potsdam.de

### Abstract

*This document shows the exemplary application of our new BPEL 2.0 formalization based on the  $\pi$ -calculus. Therefore we introduce a scenario from the financial domain and show how these processes are represented using our formalization. In addition, we also formalize one of the example processes using existing approaches.*

## 1 Introduction

Regarding compatibility and consistency checking for BPEL processes we proposed a new approach in our paper *Efficient Analysis of BPEL 2.0 Processes using  $\pi$ -Calculus*.

This paper contains the processes of the example (which is again described in Section 2) introduced in the aforementioned paper. Section 3 shows the  $\pi$ -calculus based representation of these processes. As we also discussed complexity of other approaches, we also depict representations of the customer process using different approaches in Section 4.

## 2 Scenario Description

To introduce the topic of process verification we present an example from the financial domain, namely an investment spending scenario operated by a stockbroker. Figure 1 illustrates the example using the Business Process Modeling Notation (BPMN) [5]. At first the customer sends an investment request to a stockbroker. Depending on the investment request, the broker obtains further information about several stocks from rating agencies. Subsequently, a set of stocks is chosen and the broker retrieves the current stock prices (not necessarily from the stock exchange) and reviews his own stock warrants whether they can be used for the current investment. After the retrieval of stock prices, the broker sends the details about the investment to the customer. This activity is independent of the warrant analysis, as only

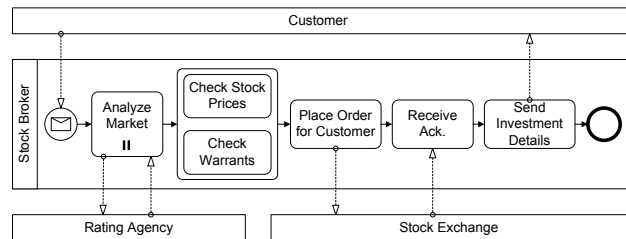


Figure 2. Incompatible stockbroker

the broker might benefit from the warrants. Later on, the broker compares his warrant prices with the current stock prices and places an order on behalf of the customer at the stock exchange. In the stock exchange process, the order is processed and an invoice is sent to the customer. After the customer met the account, the stock exchange sends an acknowledge message to the stockbroker indicating the successful proceeding of the order transaction.

Provided that the interconnected processes do not suffer from any structural incompatibility (e.g. different message formats), the introduced composition is compatible. Due to the absence of behavioral anomalies (e.g. deadlocks) all processes end properly. Nevertheless, we can imagine another process implementation for the stock broker as illustrated in Figure 2. In contrast to the first example, the stockbroker sends the investment details *after* he received the acknowledge message. As the customer requires these details before he is willing to pay for the stock order, which in turn must happen before the stock exchange sends the acknowledgement message, a deadlock occurs.

In other settings, abstract BPEL processes are given as behavioral specifications for a role. In such cases we need to check whether the fully executable BPEL is consistent with the abstract definition to ensure successful interaction. Regarding the introduced example, we can treat the introduced process for the customer as an abstract behavior description, while two specific process implementations are shown in Figure 3. In one process, the internal behavior has been changed, as the customer receives the investment

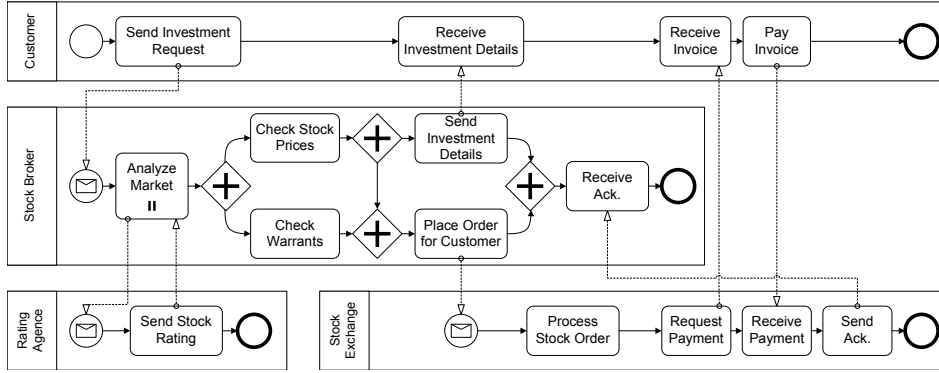


Figure 1. Stockbroker scenario

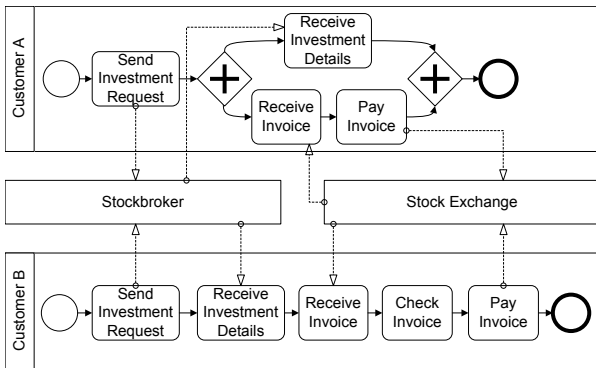


Figure 3. Different customer implementations

details and handles the invoice concurrently (Customer A). In the other process, we added an internal activity, in which the customer inspects the invoice (Customer B). For each of these process implementations, consistency to the specification has to be decided.

### 3 Processes of the Example in the $\pi$ -Calculus According to our Formalization

For illustration purposes, we used abbreviations of the names for partner links. Please note, that we use the syntax of the Advanced Bisimulation Checker [1] for the process definitions. As this tool does not support process replication, we used recursion for process instantiation instead. The Customer process is defined as follows:

```
CUST(irSir, irSid, prSpr, prSp) =
  t.'irSir<prSpr, prSp>.irSid.prSpr.'prSp.0
```

The processes realizing the rating agency are defined as follows (please note, that we had to introduce a mechanism to cancel the memory cell (using the name  $c$ ), as the whole

process for the rating agency can be invoke an unbound number of times):

```
RA(gsiSsir, mem_me, me) =
  gsiSsir(response). (
    RA(gsiSsir, mem_me, me) |
    RA2(response, mem_me, me)
  )
agent RA2(r, mem_me, me) =
  (^pid, c)'mem_me<pid, c>.'pid.'me<r>.
  t.'pid.me(r)'.c.'r.0
agent C_me(mem_me, true, me) =
  mem_me(id, c). ( M_me(id, true, me, c) |
  C_me(mem_me, true, me) )
agent M_me(pid, y, me, c) =
  pid. ('me<y>.M_me(pid, y, me, c) +
  me(y).M_me(pid, y, me, c)) + c.0
```

The processes for the stock exchange:

```
STOCKEX(orPo, orSa, mem_pr, pr) =
  (^pid) orPo(x, y)'.mem_pr<pid>.'pid.'pr<x, y>.
  t.'pid.pr(x, y)'.x.y.'orSa.0
C_pr(mem_pr, true, pr) =
  mem_pr(id). ( M_pr(id, true, true, pr) |
  C_pr(mem_pr, true, pr) )
M_pr(pid, x, y, pr) =
  pid. ('pr<x, y>.M_pr(pid, x, y, pr) +
  pr(x, y).M_pr(pid, x, y, pr))
```

The following processes realize the stockbroker:

```
STOCKBROKER(irSir, irSid, gsiSsir, orPo, orSa,
  mem_pr2, pr2, cpl, h, skip, exec) =
  (^executed, executed2, executed3, executed4,
  go, go2, go3, pid, skipped)
  (irSir(x, y)'.mem_pr2<pid>.'pid.'pr2<x, y>.
  WHILE(executed, gsiSsir) |
  executed. (
    t.'executed3.0 |
    executed3.t.'cpl.'h.'go2.0 |
    go2.'irSid.'executed2.0 |
    t.t.0 |
    (skip.'skipped.0 +
    exec.'pid.pr2(x, y)'.orPo<x, y>.
    'executed4.0) |
    (skipped.'go3.0 + executed4.'go3.0) |
```

```

        go3.'executed2.0
    ) |
    executed2.executed2.orSa.0 )
TAR_COUNTER(h,a)= h.'a.0
TAR_DECISION(cpl,exec,a,skip)
= cpl.(a.'exec.0 | cpl.0) + a.'skip.0
WHILE(executed,gsiSsir)=
t.'executed.0 +
(^r)t.'gsiSsir<r>.r.WHILE(executed,gsiSsir)
C_pr2(mem_pr2,true,pr2)=
mem_pr2(id).(M_pr2(id,true,true,pr2) |
C_pr2(mem_pr2,true,pr2))
M_pr2(pid,x,y,pr2)=
pid.(^pr2<x,y>.M_pr2(pid,x,y,pr2) +
pr2(x,y).M_pr2(pid,x,y,pr2))

```

The second process realizing a stockbroker (as introduced in Section 2 and illustrated in figure 2) is defined as follows:

```

STOCKBROKER_2(irSir,irSid,gsiSsir,
orPo,orSa,mem_pr2,pr2)=
(^executed,executed2,pid)
(irSir(x,y).'mem_pr2<pid>.'pid.'pr2<x,y>).
WHILE(executed,gsiSsir) |
executed.(
t.'executed2.0 | t.'executed2.0) |
executed2.executed2.'pid.pr2(x,y).
^orPo<x,y>.orSa.'irSid.0
)

```

In the context of consistency checking, we introduced two different implementations (depicted in figure 3) for the customer service. They are formalized as follows:

```

CUST_A(irSir,irSid,prSpr,prSp) = (^executed) (
t.'irSir<prSpr,prSp>.(
irSid.'executed |
prSpr.'prSp.'executed
) |
executed.executed.0
)
CUST_B(irSir,irSid,prSpr,prSp)=
t.'irSir<prSpr,prSp>.
irSid.prSpr.t.'prSp.0

```

Further on, interaction soundness has been checked for the following systems:

```

ACUST(irSir,irSid,prSpr,prSp,o)=
^irSir<prSpr,prSp>.irSid.prSpr.'prSp.'o.0
ASYS_1(i,o) =
(^irSir,irSid,prSpr,prSp,orPo,orSa,gsiSsir,
true,mem_pr,pr,mem_pr2,pr2,mem_me,me)
i.(
ACUST(irSir,irSid,prSpr,prSp,o) |
RA(gsiSsir,mem_me,me) |
C_me(mem_me,true,me) |
STOCKEX(orPo,orSa,mem_pr,pr) |
C_pr(mem_pr,true,pr) |
(^h,a,exec,skip,cpl) (
STOCKBROKER(irSir,irSid,gsiSsir,
orPo,orSa,mem_pr2,pr2,cpl,h,skip,exec) |
TAR_COUNTER(h,a) |
TAR_DECISION(cpl,exec,a,skip)
)
)

```

```

) |
C_pr2(mem_pr2,true,pr2)
)
ASYS_2(i,o) =
(^irSir,irSid,prSpr,prSp,orPo,orSa,gsiSsir,
true,mem_pr,pr,mem_pr2,pr2,mem_me,me)
i.(
ACUST(irSir,irSid,prSpr,prSp,o) |
RA(gsiSsir,mem_me,me) |
C_me(mem_me,true,me) |
STOCKEX(orPo,orSa,mem_pr,pr) |
C_pr(mem_pr,true,pr) |
STOCKBROKER_2(irSir,irSid,gsiSsir,
orPo,orSa,mem_pr2,pr2) |
C_pr2(mem_pr2,true,pr2)
)
)

```

## 4 Different Formalizations of the Customer Process

In the following, we list different formal representations of the introduced customer process. Our formalization leads to the process  $CUST$ :

$$CUST(IrSir, IrSid, PrSpr, PrSp) = \tau.\overline{IrSir}\langle PrSpr, PrSp \rangle.IrSid.PrSpr.\overline{PrSp}.0$$

Using the language  $web\pi_\infty$  as presented by Mazzara and Lucchi in [4], the customer process is defined as:

$$CUST(IrSir, IrSid, PrSpr, PrSp) = (\nu y_1)(\overline{y_1}\langle fn(CUST_1) \rangle | y_1(\tilde{u}_1).CUST_1(IrSir, IrSid, PrSpr, PrSp))$$

$$CUST_1(IrSir, IrSid, PrSpr, PrSp) = (\nu y_2)(\overline{IrSir}(\tilde{i}_1) | \overline{y_2}\langle fn(CUST_2) \rangle | y_2(\tilde{u}_2).CUST_2(IrSid, PrSpr, PrSp))$$

$$CUST_2(IrSid, PrSpr, PrSp) = (\nu y_3)(IrSid(\tilde{i}_2).\overline{y_3}\langle fn(CUST_3) \rangle | y_3(\tilde{u}_3).CUST_3(PrSpr, PrSp))$$

$$CUST_3(PrSpr, PrSp) = (\nu y_4)(PrSpr(\tilde{i}_3).\overline{y_4}\langle fn(CUST_4) \rangle | y_4(\tilde{u}_4).CUST_4(PrSp))$$

$$CUST_4(PrSp) = (\nu y_5)(\overline{PrSpr}(\tilde{i}_4).\overline{y_5} | y_5.0)$$

Further on, the approach presented by Fadlisyah ([2]) specifies the following  $\pi$ -calculus processes to formalize the customer process. Please note, the we abstracted from all fault, error and compensation handling. We only the show processes of the main context, one sequence and one

invoke activity, as the remaining activities can be derived in the same way. The main process is formalized as:

$$\begin{aligned}
 PRO(pp\tilde{r}oc, pl\tilde{i}nk) &= \\
 (\mathbf{v} \tilde{p}act_1)(PC(pp\tilde{r}oc, \tilde{p}act_1) \mid SEQ_1(\tilde{p}act_1, pl\tilde{i}nk)) \\
 PC(pp\tilde{r}oc, \tilde{p}act_1) &= \\
 \overline{activate}_{SEQ_1}.PC^{DoingAct}(pp\tilde{r}oc, \tilde{p}act_1) \\
 PC^{DoingAct}(pp\tilde{r}oc, \tilde{p}act_1) &= \\
 \overline{completed}_{SEQ_1}.PC^{CompletedAct}(pp\tilde{r}oc) \\
 PC^{CompletedAct}(pp\tilde{r}oc) &= \mathbf{0}
 \end{aligned}$$

One sequence is represented through the following processes:

$$\begin{aligned}
 SEQ_1(p\tilde{s}eq, pl\tilde{i}nk) &= \\
 (\mathbf{v} \tilde{p}act_1, \tilde{p}act_2)(SEQC(p\tilde{s}eq, \tilde{p}act_1, \tilde{p}act_2) \mid \\
 INV_1(\tilde{p}act_1, pl\tilde{i}nk) \mid SEQ_2(\tilde{p}act_1, pl\tilde{i}nk)) \\
 SEQC_1(p\tilde{s}eq, \tilde{p}act_1, \tilde{p}act_2) &= \\
 \overline{activate}_{SEQ_1}.SEQC_1^{FirstAct}(p\tilde{s}eq, \tilde{p}act_1, \tilde{p}act_2, activeType)
 \end{aligned}$$

$$\begin{aligned}
 SEQC_1^{FirstAct}(p\tilde{s}eq, \tilde{p}act_1, \tilde{p}act_2, activeType) &= \\
 \overline{activate}_{INV_1}.completed_{INV_1}. \\
 SEQC_1^{SecondAct}(p\tilde{s}eq, \tilde{p}act_2, activeType)
 \end{aligned}$$

$$\begin{aligned}
 SEQC_1^{SecondAct}(p\tilde{s}eq, \tilde{p}act_1, \tilde{p}act_2, activeType) &= \\
 \overline{activate}_{SEQ_2}.completed_{SEQ_2}.StructAct^{Completed}(p\tilde{s}eq)
 \end{aligned}$$

$$StructAct^{Completed}(p\tilde{s}eq) = \overline{completed}.0$$

One invoke activity is formalized as follows:

$$\begin{aligned}
 INV_1(\tilde{p}act_1, pl\tilde{i}nk) &= \overline{activate}_{INV_1}.INV_1^{Do}(\tilde{p}act_1, pl\tilde{i}nk) \\
 INV_1^{Do}(\tilde{p}act_1, pl\tilde{i}nk) &= \overline{plink_i.n}.INV_1^{Complete}(\tilde{p}act_1) \\
 INV_1^{Complete}(\tilde{p}act_1) &= \overline{BasicAct}^{Complete}(\tilde{p}act_1) \\
 \overline{BasicAct}^{Complete}(\tilde{p}act_1) &= \overline{completed}.0
 \end{aligned}$$

Finally, Figure 4 shows the initial mapping of the customer process to a Petri net, as introduced by Stahl, while Figure 5 depicts the simplified representation and Figure 6 shows the reduced net. The Petri net have been generated with BPEL2oWFN[3], while the state space has been determined using the Integrated Net Analyser[6].

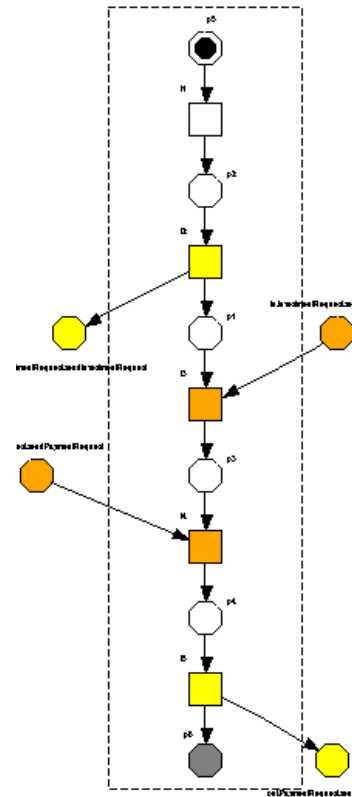


Figure 5. The customer process as a Petri net showing only the communication flow

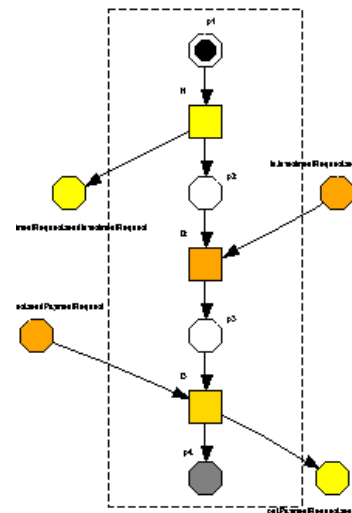
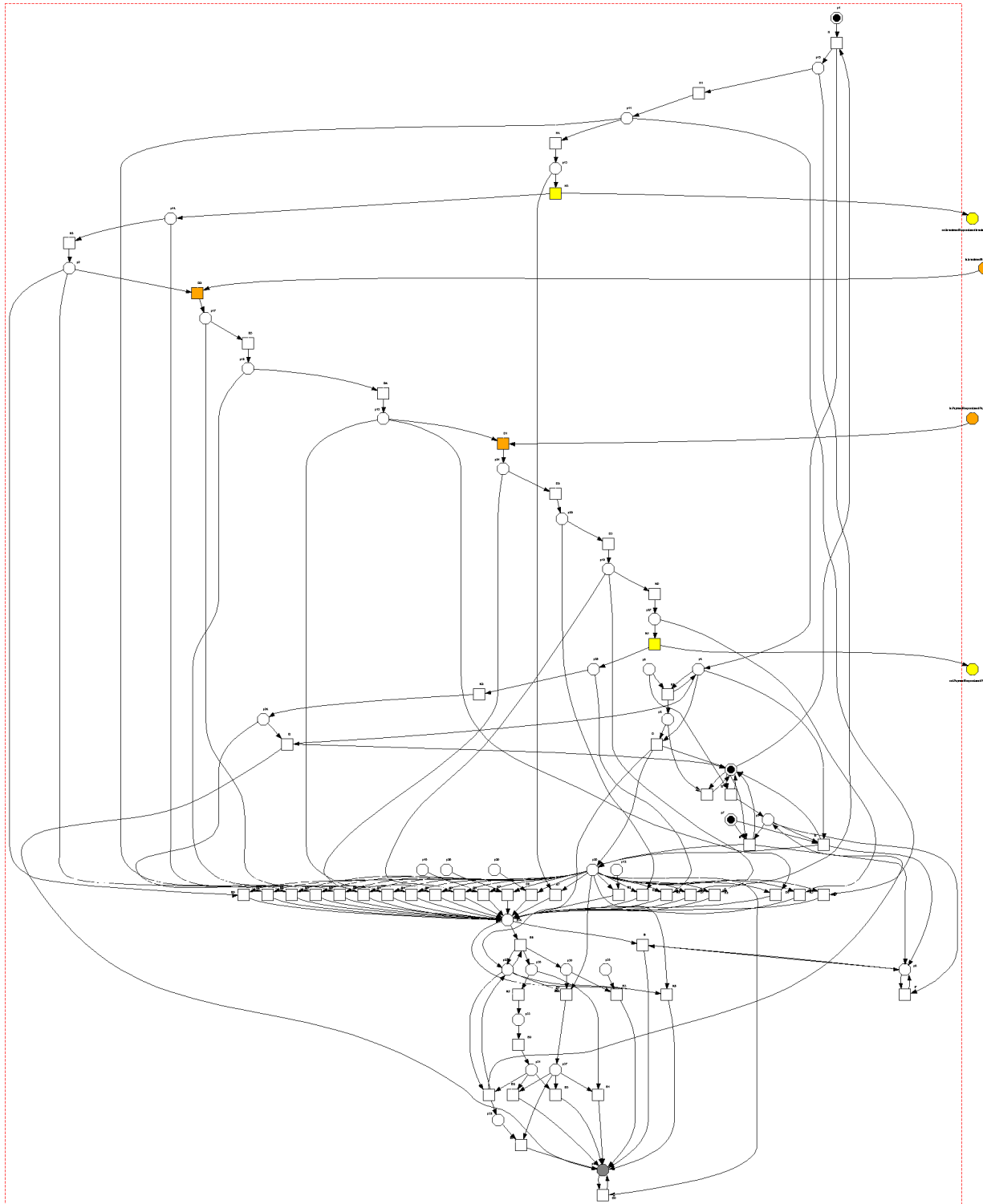


Figure 6. The customer process as a Petri net in its reduced representation



**Figure 4. The initial mapping of the customer process to a Petri net**

## References

- [1] S. Briais. Advanced Bisimulation Checker (ABC). <http://lamp.epfl.ch/~sbriais/abc/abc.html>, 2007.
- [2] M. Fadlisyah. Using the  $\pi$ -Calculus for Modeling and Verifying Processes on Web Services. Master's thesis, Insitute for Theoretical Computer Science, Dresden University of Technology, 2004.
- [3] N. Lohmann, C. Gierds, and M. Znamirowski. BPEL2oWFN. <http://www.gnu.org/software/bpel2owfn/>, 2007.
- [4] M. Mazzara and R. Lucchi. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, 2006. To appear.
- [5] OMG. Business Process Modeling Notation (BPMN) Specification Version 1.0, February 2006.
- [6] P. H. Starke and S. Roch. Integrated Net Analyser. <http://www2.informatik.hu-berlin.de/lehrstuehle/automaten/ina/>, April 1999.