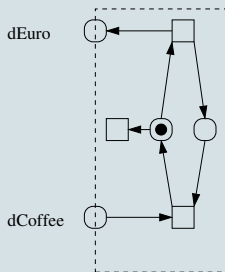
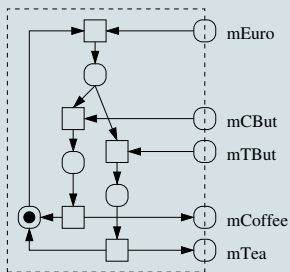


## Adapter Generation – some open issues

Arjan Mooij  
A.J.Mooij@tue.nl

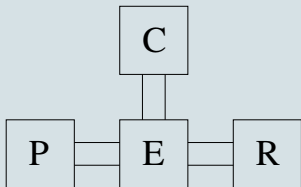
BEST colloquium, Eindhoven  
23rd March 2009

## Adapter specification



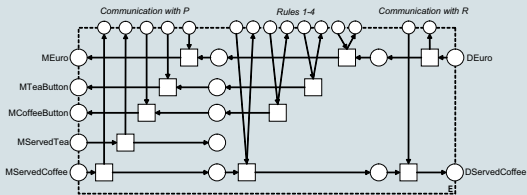
dEuro	$\mapsto$	mEuro
	$\mapsto$	mCBut
	$\mapsto$	mTBut
mCoffee	$\mapsto$	dCoffee

## Adapter generation



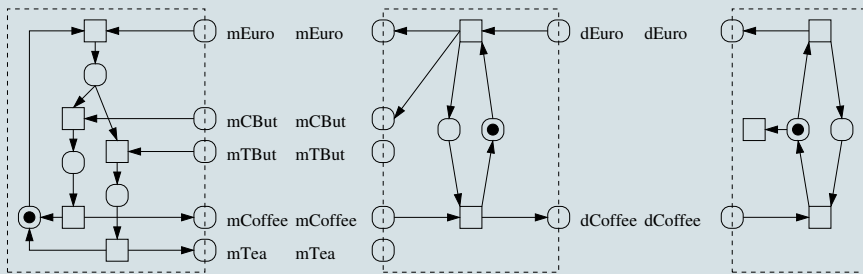
1. generate from the SEA rules an engine  $E$
2. compose given services  $P$  and  $R$  with engine  $E$
3. synthesize a controller (as a transition system) using Fiona
4. transform this controller to an open net  $C$  using Petrify
5. compose the engine  $E$  and the controller  $C$

## Structure of an engine



- ▶ internal place for every message type
- ▶ transitions that receive a message from the interface
- ▶ transitions that send a message to the interface
- ▶ transitions that perform a SEA rule
- ▶ interface places for the controller to enable transitions
- ▶ interface places to notify the controller of executed transitions

## Generated adapter



- ▶ how to obtain adapter specifications (open nets + SEA rules)?
- ▶ how to make the adapter specification more powerful?
- ▶ how to generate adapters more efficiently?
- ▶ how to generate adapters that are better and look simpler?

## Obtaining an adapter specification

How to obtain oWFN models of services:

- ▶ BPEL descriptions: Lohmann/Massuthe/Stahl/Weinberg
- ▶ process mining (for immediate use by Fiona): Günther

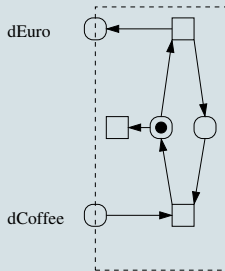
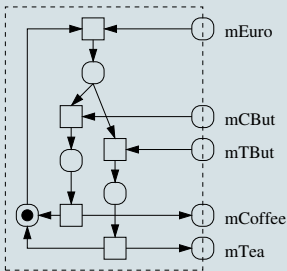
How to obtain the SEA rules (= domain knowledge):

- ▶ technologies of semantic web and ontologies
- ▶ from the incompatible services themselves?

Some ideas for using the incompatible services:

- ▶ trace-based mismatch patterns: Müllers/Dijkman/Mooij
- ▶ interactively diagnosing non-adaptability: Mooij/Lohmann

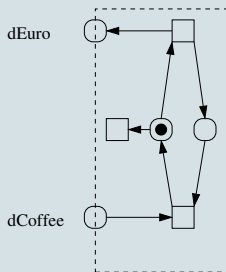
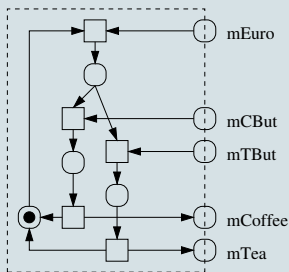
## Diagnosing non-adaptability



Sketch of some initial ideas by Mooij/Lohmann:

- ▶ suppose no SEA rules are known
- ▶ one reachable deadlock: dEuro is available
- ▶ services waiting for: mEuro, or dCoffee

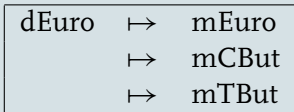
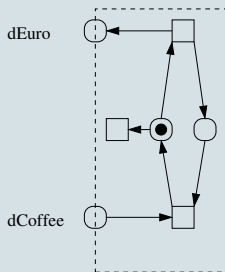
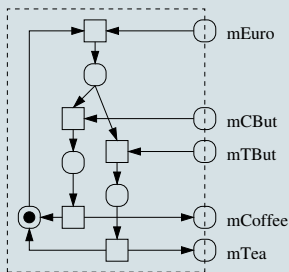
## Diagnosing non-adaptability...



$$\text{dEuro} \mapsto \text{mEuro}$$

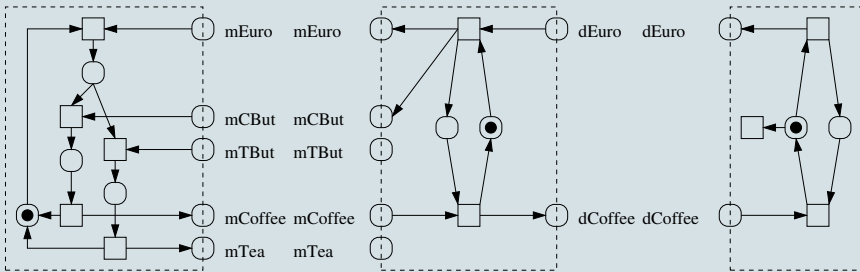
- ▶ one reachable deadlock: no messages available
- ▶ services waiting for: mCBut, mTBut, or dCoffee

## Diagnosing non-adaptability.....



- mCoffee or mTea is available; services waiting for mEuro, or dCoffee

## Diagnosing non-adaptability.....



dEuro	$\mapsto$	mEuro
	$\mapsto$	mCBut
	$\mapsto$	mTBut
mCoffee	$\mapsto$	dCoffee

## Extensions of the adapter specification/generation

Control over the messages send by the services: Gierds/Müller/Wolf

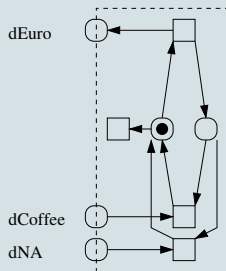
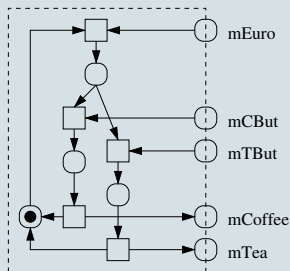
- ▶ which transmitted messages can be detected?
- ▶ which transmitted messages can be delayed?

Associate a cost function to the SEA rules:

- ▶ global optimization for acyclic services: Gierds
- ▶ local optimization at run-time: Mooij

So far, there are no results on global optimization in general

## An application of local optimization



Domain knowledge:

dEuro	$\mapsto$	mEuro
	$\mapsto$	mCBut
	$\mapsto$	mTBut
mCoffee	$\mapsto$	dCoffee
dEuro	$\mapsto$	dNA (LOW PRIORITY)

## Some SEA patterns

AND split/join:

- ▶ join: Man, Woman  $\rightarrow$  Couple      split: Couple  $\rightarrow$  Man, Woman

XOR split/join:

- ▶ join: Even  $\rightarrow$  Int and Odd  $\rightarrow$  Int
- ▶ split: Int  $\rightarrow$  Even and Int  $\rightarrow$  Odd      ???

Besides an unconditional XOR-split, we need a (total) conditional one:

1. the *controller* decides to produce Even or Odd
2. based on the value of Int, the *engine* produces either Even or Odd:

This can be modeled in terms of an engine:

$$\text{Int} \rightarrow \text{Even} \mid \text{Odd}$$

- ▶ abstract from conditions by assuming that their disjunction is total

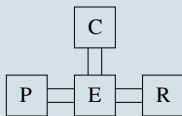
## Engine variations

(Un)boundedness of the internal places of the engine:

- ▶ unbounded (= conceptual)
- ▶ bounded using complementary places
- ▶ bounded using bad behavior in case of place overflow

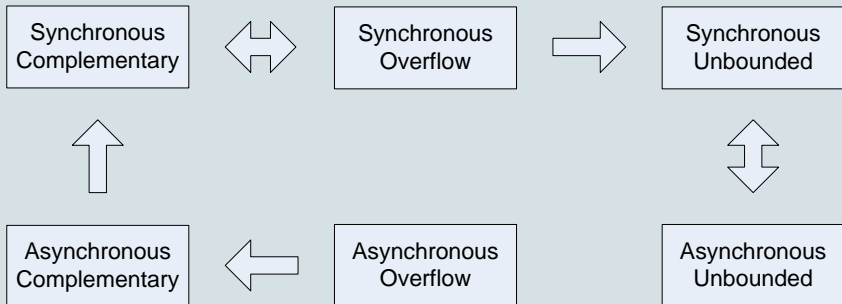
Communication between engine and controller:

- ▶ asynchronous (place fusion): Gierds/Mooij/Wolf
- ▶ synchronous (transition fusion): Mooij/Voorhoeve



## Engine variations...

Generality of the resulting six engines (for a fixed bound):



- ▶ History: async overflow  $\rightarrow$  async complementary  $\rightarrow$  sync overflow
- ▶ “Overflow” faster than “complementary”, and simpler adapters

## Simplifying the engine

Explore simplifications of the engine such that:

- ▶ smaller state-space to be explored by Fiona
- ▶ more opportunities for simplifying the final adapter

Initial ideas of Gierds/Müller:

- ▶ no interface places for conflict-free rule transitions
- ▶ remove acyclic internal traps

Initial ideas of Mooij:

- ▶ only an enabling input place for rule choices in the engine
- ▶ only a notifying output place for rule loops in the engine

So far, no results for synchronous engine-controller interface

## Compositional adapter generation

Initial ideas of Mooij:

- ▶ example: simple protocol, but with a few hundred message types
- ▶ battling the state space explosion problem

Partition  $P \oplus E \oplus R$  according to disjoint parallel compositions

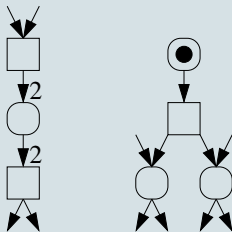
- ▶ equivalence for weak termination
- ▶ implication for deadlock-freedom

Future: explore the possibilities regarding:

- ▶ sequential composition
- ▶ repetition
- ▶ alternative composition

## Simplifying the final adapter

Unnecessarily complex fragments:



Unreachable behavior: Wolf, and Mooij/Voorhoeve

- ▶ exploit the true-nodes in the operating guideline
- ▶ remove adapter places that are redundant in the composed net
- ▶ can this idea improve the efficiency of the Petrify step?
- ▶ can this idea improve the readability of the result of Petrify?

## Absence of $\tau$ -labeled auto-loops in the adapter

Property of each Fiona-generated controller:

- ▶ absence of  $\tau$ -labeled auto-loops

This property is not maintained for adapter generation:

- ▶ e.g.: Koffie  $\rightarrow$  Kaffee , Kaffee  $\rightarrow$  Koffie
- ▶ or: SizeInches  $\rightarrow$  SizeMeters , SizeMeters  $\rightarrow$  SizeInches
- ▶ or: Man, Woman  $\rightarrow$  Couple , Couple  $\rightarrow$  Man, Woman
- ▶ once one rule can be applied, these rules can be applied forever
- ▶ cyclic dependencies & deadlock-freedom gives non-sense adapters

How to solve this?

- ▶ partially: controller generation with respect to weak termination?

## Proof techniques for reasoning about accordance

Techniques based on abstract notion of partner

- ▶ Reisig et al, and Mooij/Voorhoeve

Accordance equivalence for open nets:

- ▶ Murata rules
- ▶ projection inheritance: Aalst/Lohmann/Massuthe/Stahl/Wolf
- ▶ accordance transformations: Aalst/Lohmann/Massuthe/Stahl/Wolf

Accordance pre-order for open nets:

- ▶ accordance transformation: Aalst/Lohmann/Massuthe/Stahl/Wolf
- ▶ operating guidelines (if finite): Stahl/Massuthe/Bretschneider
- ▶ comparing adapters using engine converters: Mooij/Voorhoeve

We need more/better proof techniques for accordance!

## Summary

Support for the required inputs:

- ▶ process mining of services from logs
- ▶ extract SEA suggestions from services

More-sophisticated adapter generation:

- ▶ specification: optimization of costs, conditional XOR-split, ...
- ▶ engine: interfaces, internal structures, ...
- ▶ compositional construction of adapters/controllers
- ▶ simplifications of the final adapters

Fundamental open issues:

- ▶ Absence of  $\tau$ -labeled auto-loops in the final adapters
- ▶ Proof techniques for the accordance pre-order