Formal Model-Based Approaches for the Development of Composite Systems

MeFoSyLoMa Seminar
(originally, Habil. thesis defense, Nov. 24th, 2011)

Pascal Poizat
Université d’Evry Val d’Essonne;
LRI CNRS UMR 8623 et Université Paris-Sud 11

Evry, February 17th, 2012
Timeline

- **1997** LINA
- **2001** IBISC
- **2008** INRIA
- **2008** LRI

**1997 - 2001**
- formal methods integration

**2003 - 2006**
- software architectures

**2006**
- adaptation
- testing
- composition

**2008 - now**
- LRI

**See**
- IEEE TSE 33(3), 2007
- J.UCS 12(12), 2006
- IEEE TSE 34(4), 2008
- IEEE TSE under press
- TESTCOM'09, SAC'12
- ICSOC'08, ISoLa’10
- ICWS’10, ICSOC’10
What about MeFoSyLoMa?

1997
LINA

2001
IBISC

2003
software architectures

2006
adaptation

2008
INRIA

now
LRI

2006
composition

2007
formal methods integration

see
IEEE TSE 33(3), 2007

see
J.UCS 12(12), 2006

see
IEEE TSE 34(4), 2008
IEEE TSE under press
TESTCOM’09, SAC’12
ICSOC’08, ISoLa’10
ICWS’10, ICSOC’10
What about MeFoSyLoMa?

1997 LINA
2001 IBISC
2008 INRIA
LRI

formal methods integration
software architectures
adaptation
testing composition

see IEEE TSE 33(3), 2007
see J.UCS 12(12), 2006
see IEEE TSE 34(4), 2008
IEEE TSE under press
TESTCOM’09, SAC’12
ICSOC’08, ISoLa’10
ICWS’10, ICSOC’10

Applicative Domain
What about MeFoSyLoMa?

1997
LINA

2001
IBISC

2003
formal methods integration

2006
software architectures

INRIA

2008
LRI

adaptation
testing
composition

2006

What about MeFoSyLoMa?

Applicative Domain

MeFoSyLoMian Models/Tools

1997
see IEEE TSE 33(3), 2007

2001

2003

see J.UCS 12(12), 2006

2008

see IEEE TSE 34(4), 2008
IEEE TSE under press
TESTCOM’09, SAC’12
ICSOC’08, ISoLa’10
ICWS’10, ICSOC’10

now

What about MeFoSyLoMa?

Applicative Domain

MeFoSyLoMian Models/Tools

1997
LINA
formal methods integration
see IEEE TSE 33(3), 2007

2001
IBISC
software architectures
see J.UCS 12(12), 2006

2003
adaptation

2008
INRIA
MeFoSyLoMa

2006
INRIA
composition
testing

2008
LRI
adaptation

ICWS’10, ICSOC’10

IEEE TSE 34(4), 2008
IEEE TSE under press
TESTCOM’09, SAC’12
ICSOC’08, ISoLa’10

What about MeFoSyLoMa?
1997 LINA
2001 IBISC
INRIA 2008 LRI

What about MeFoSyLoMa?

Applicative Domain
MeFoSyLoMian Models/Tools

formal methods integration

2001 software architectures

MeFoSyLoMa

adaptation

2006
testing

composition

see IEEE TSE 33(3), 2007
see J.UCS 12(12), 2006
see IEEE TSE 34(4), 2008
IEEE TSE under press
TESTCOM’09, SAC’12
ICSOC’08, ISoLa’10
ICWS’10, ICSOC’10
the simplest things
get complex with time

Source: talk by Ph. Merle at GDR GPL 2011
what about software?

- increasing use of:
  - viewpoints
  - distribution
  - interaction

Source: WikiMedia Commons (by Kishorekumar 62)
Structuring

- **Modules**, Object-Orientation
  - +: well-defined provided interfaces, reusability
  - -: hidden required functionalities
Structuring

- **Modules, Object-Orientation**
  + well-defined provided interfaces, reusability
  - hidden required functionalities

- **Software Components, Services**
  + explicit required functionalities, dynamic binding
  - richer interfaces are **harder to use**
Software Architectures (SA)

**components**

- **TripService**
  - operation bookPlane
    - inputs from:string, to:string, departure:date, return:date
    - outputs eticket:string
  - operation book
    - inputs from:string, to:string, departure:date, return:date
    - outputs eticket:string
  - operation setup
    - inputs from:string, to:string, departure:date, return:date
    - outputs sessionid:integer

- **PlaneService**
  - planeClient
    - operation bookPlane
      - inputs from:string, to:string, departure:date, return:date
      - outputs eticket:string
    - operation setup
      - inputs from:string, to:string, departure:date, return:date
      - outputs sessionid:integer

- **HotelService**
  - hotelClient
    - operation bookHotel
      - inputs from:string, to:string, hotelstars:integer
      - outputs eticket:string
    - operation setup
      - inputs from:string, to:string, departure:date, return:date
      - outputs sessionid:integer

**interfaces**

- **TripService**
  - operation book
    - inputs from:string, to:string, departure:date, return:date
    - outputs eticket:string
  - operation setup
    - inputs from:string, to:string, departure:date, return:date
    - outputs sessionid:integer

- **PlaneService**
  - planeClient
    - operation bookPlane
      - inputs from:string, to:string, departure:date, return:date
      - outputs eticket:string
    - operation setup
      - inputs from:string, to:string, departure:date, return:date
      - outputs sessionid:integer

- **HotelService**
  - hotelClient
    - operation bookHotel
      - inputs from:string, to:string, hotelstars:integer
      - outputs eticket:string
    - operation setup
      - inputs from:string, to:string, departure:date, return:date
      - outputs sessionid:integer
Software Architectures (SA)

- **Interface** (design in-the-small)
  - `bookPlane`
    - **Inputs**: from: string, to: string, departure: date, return: date
    - **Outputs**: eticket: string

- **Component** (design in-the-large)
  - `TripService`
    - `book`
      - **Inputs**: from: string, to: string, departure: date, return: date, hotelstars: integer
      - **Outputs**: eticket: string, hotel: Hotel
    - `setup`
      - **Inputs**: from: string, to: string, departure: date, return: date
      - **Outputs**: sessionid: integer
    - `book`
      - **Inputs**: sessionid: integer
      - **Outputs**: information: Hotel
  - `PlaneService`
    - `bookPlane`
      - **Inputs**: from: string, to: string, departure: date, return: date
      - **Outputs**: eticket: string
    - `setup`
      - **Inputs**: from: string, to: string, departure: date, return: date
      - **Outputs**: sessionid: integer
    - `book`
      - **Inputs**: sessionid: integer
      - **Outputs**: information: Hotel
  - `HotelService`
    - `bookPlane`
      - **Inputs**: from: string, to: string, departure: date, return: date
    - `setup`
      - **Inputs**: from: string, to: string, departure: date, return: date
    - `book`
      - **Inputs**: sessionid: integer
      - **Outputs**: information: Hotel
  - `hotelClient`
  - `planeClient`

2 design activities

- «**design in-the-small**» design, implementation, and verification of sub-systems
- «**design in-the-large**» structuring of the system as a set of sub-systems
Software Architectures (SA)

• **signature** interface provides the access points (ports, operations)

• **behavioural** interface defines the usage protocol

  additionally, **semantic** interface fosters automation

  **non-functional** interface for QoS

**4 description levels**

**2 design activities**

- **«design in-the-small»** design, implementation, and verification of sub-systems

- **«design in-the-large»** structuring of the system as a set of sub-systems

**components**

**interfaces**

- **operation bookPlane**
  - inputs: from:string, to:string, departure:date, return:date
  - outputs: eticket:string

- **operation book**
  - inputs: sessionid:integer
  - outputs: information:Hotel

- **operation setup**
  - inputs: from:string, to:string, departure:date, return:date
  - outputs: sessionid:integer

- **operation cancel**
  - inputs: sessionid:integer
  - outputs: information:Hotel
Software Architectures (SA)

- **signature** interface
  - provides the access points (ports, operations)

- **behavioural** interface
  - defines the usage protocol

- additionally,
  - **semantic** interface fosters automation
  - **non-functional** interface for QoS

4 description levels

- **«design in-the-small»**
  - design, implementation, and verification of sub-systems

- **«design in-the-large»**
  - structuring of the system as a set of sub-systems

2 design activities
Software Architectures (SA)

- **Signature interface**: provides the access points (ports, operations)
  - **Operation** `book`:
    - **Inputs**: `from:string, to:string, departure:date, return:date, hotelstars:integer`
    - **Outputs**: `eticket:string`

- **Behavioural interface**
  - **Operation** `bookPlane`:
    - **Inputs**: `from:string, to:string, departure:date, return:date`
    - **Outputs**: `eticket:string`

- **Components**
  - **TripService**
    - **Operation** `book`:
      - **Inputs**: `from:string, to:string, departure:date, return:date`
      - **Outputs**: `eticket:string`
    - **Operation** `bookPlane`:
      - **Inputs**: `from:string, to:string, departure:date, return:date`
      - **Outputs**: `eticket:string`
  - **HotelService**
    - **Operation** `setup`:
      - **Inputs**: `from:string, to:string, departure:date, return:date`
      - **Outputs**: `sessionid:integer`
    - **Operation** `book`:
      - **Inputs**: `sessionid:integer`
      - **Outputs**: `information:Hotel`
    - **Operation** `cancel`:
      - **Inputs**: `sessionid:integer`

- **Interfaces**
  - **TripClient**
    - **Operation** `book`
  - **PlaneClient**
    - **Operation** `bookPlane`
  - **HotelClient**
    - **Operation** `setup`
    - **Operation** `book`
    - **Operation** `cancel`

- **Levels**
  - **4 description levels**

```
required by TripService
receive(tripClient, book, {from, to, dep, ret});
{hi} := invoke(hotel, setup, {from, to, dep, ret});
{infoh} := invoke(hotel, book, {hi});
{pi} := invoke(plane, bookPlane, {from, to, dep, ret});
if pi.equals("no plane") { invoke(hotel, cancel, {hi}); } reply(tripClient, book,...)
```

```
provided by HotelService
receive(hotelClient, setup, {from, to, departure, return});
... ; reply(hotelClient, setup, {sessionid});
pick { // «choose between»
onMessage(hotelClient, book, {sessionid}) -> ... ;
onMessage(hotelClient, cancel, {sessionid}) -> ... ; }
```
**Software Architectures (SA)**

**4 description levels**

- **signature interface** provides the access points (ports, operations)
- **behavioural interface**

**2 design activities**

**interfaces**

- **operation** bookPlane
  - **inputs**: from:string, to:string, departure:date, return:date
  - **outputs**: eticket:string

- **operation** book
  - **inputs**: from:string, to:string, departure:date, return:date
  - **outputs**: eticket:string

**components**

- **TripService**
  - **operation** setup
    - **inputs**: from:string, to:string, departure:date, return:date
    - **outputs**: sessionid:integer
  - **operation** book
    - **inputs**: sessionid:integer
    - **outputs**: information:Hotel
  - **operation** cancel
    - **inputs**: sessionid:integer

- **HotelService**
  - **operation** setup
    - **inputs**: from:string, to:string, departure:date, return:date
    - **outputs**: sessionid:integer
  - **operation** book
    - **inputs**: sessionid:integer
    - **outputs**: information:Hotel
  - **operation** cancel
    - **inputs**: sessionid:integer

**required by TripService**

- setup! → setup?
- book! → book?
- m! emission
- m? reception
- cancel!
- final state

**provided by HotelService**

- setup? → setup!
- book! → book?
- cancel!
- final state
SA vs. SOA

• Service-Oriented Architectures (SOA) are the modern instance of Software Architectures
SA vs. SOA

• Service-Oriented Architectures (SOA) are the modern instance of Software Architectures

• Simple correspondence with SOA major implementation: Web services

<table>
<thead>
<tr>
<th>Software Architectures</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic component</td>
<td>simple service</td>
</tr>
<tr>
<td>composite component (centralized / distributed)</td>
<td>composite service (orchestration / choreography)</td>
</tr>
<tr>
<td>signature interface</td>
<td>WSDL interface</td>
</tr>
<tr>
<td>behavioural interface</td>
<td>conversation, e.g., in ABPEL</td>
</tr>
<tr>
<td>semantic interface</td>
<td>semantic annotations within WSDL files</td>
</tr>
</tbody>
</table>
Development processes

1. compose
   - orchestration specification → orchestration model
2. implement
   - service models (interface) → orchestration implementation
3. test
   - service implementations

bottom-up centralized

4. test
   - peer models (interface) → orchestration implementations

top-down distributed

- A1 check realizability
- A2 generate peers
- B check conformance
Development processes

• both processes are based on **3 main activities:**
  - adaptation
  - composition
  - testing
• **fact:** components are developed separately by different third parties

• **consequence:** mismatch between provided and required interfaces

• mismatch **prevents:**
  - reuse
  - composition
  - replacement
Mismatch

- **mismatch categories:**

- name mismatch (1-1)
  - print vs. imprimer

- unspecified send/rec (0-1 / 1-0)
  - login ; request* ; logout vs. (login ; request ; logout)*

- generalized mismatch (n-m)
  - res := query(x,y) vs. id := q1(x); res := q2(id,y)

- reordering mismatch
  - set(file); res := do(action) vs. set(action); res := do(file)
Adaptation

• addresses mismatch by generating **adaptors**
Adaptation

• addresses mismatch by generating **adaptors**

• is not **evolution non-intrusiveness**

• is not **customizing no envisionned context**

• is not **middleware models/code generation rather than technical support**

• is not **control synthesis data & message buffering**
Adaptation Approaches

• **restrictive** adaptation techniques
  prune interactions leading to deadlocks
  [*Inverardi and Tivoli, 2003*]
  +: fully automatic, **n-ary adaptation**, system properties
  -: support fewer adaptation scenario

• **generative** adaptation techniques
  rename and reorder exchanged messages
  [*Yellin and Strom, 1997*, *Bracciali et al, 2005*, *Brogi et al, 2006*]
  +: **extended mismatch support**
  -: generally purely theoretical and binary approaches

• **ad-hoc** adaptation techniques
  [*Schmidt and Reussner, 2002*, *Benatallah et al, 2005*, *Dumas et al, 2006*]
  -: specific mismatch (patterns/algebras), often manual
Approach: Contracts

- **adaptation contracts** are used to specify what an adaptor can do

- possible correspondences
  - **vectors**
    - absorption: `<PDA:shutdown!; ROOM:_>`
    - renaming: `<PDA:pdf?; ROOM:text!>`
    - generation: `<PDA:_; ROOM:text_request?>`

- possible orderings, simple form of system properties
  - **vector-labelled transition systems**
    (v-LTS, LTS whose labels are vectors)
Approach: Constraints

- an adaptor acts **in-between** components

- it **respects the component behaviours**
  ➤ event mirroring (sending ↔ reception)

- it **gets messages when sent**
  ➤ messages are resources saved for later use

- it **sends messages when required**
  ➤ use of owned resources

Petri nets are well-suited for this
Approach: Technique (1/3)

- transformation of component LTS models into Petri nets
Approach: Technique (1/3)

- transformation of component LTS models into Petri nets

- place (store)
- transition (mirror)

- initial state ➠ token
- final state ➠ accept transition
Approach: Technique (2/3)

- transformation of the v-LTS contract into Petri nets

Select: vlist, vchoice, vend
SelectT: Select, vtget
Approach: Technique (2/3)

- transformation of the v-LTS contract into Petri nets

**Vector ➠ tau transition (transfer)**

**v-LTS ➠ ordering of tau transitions**

**Initial state ➠ token**

**Final state ➠ accept transition**

\[INIT\] 
\[venter\] 
\[SelectT, vlist, vchoice, vend\] 
\[Select: vlist, vchoice, vend\] 
\[SelectT: Select, vtget\] 

\[v-GUEST \text{1st}\] 
\[vgid\] 
\[SelectT\] 
\[vleave\] 
\[vtmode\] 
\[vreid\] 

\[v-GUEST \text{xth}\] 

**Diagram:**
- Petri nets representation of the v-LTS contract transformation to Petri nets.
Approach: Technique (3/3)

- **fusion** of the Petri nets on shared places
  messages are resources produced/consumed

- computation of the Petri net **marking graph**
  all possible interactions of adapted system

- **pruning** paths leading to deadlocks
  following [Inverardi and Tivoli, 2003]

- behavioural **reduction** to remove internal transitions
  introduced by adaptation basic steps (vectors)

ticket? → login! → credentials! → invoice? → logout!

this does not work

this does not work

eMuseum
eMuseum

this works
Contributions on Adaptation

restrictive and generative

n-ary

simple properties

application to WWF

FMOODS’06, WCOP’07

IEEE TSE 34(4), 2008

adaptor

9000 lines (python)

+ reuses TINA and CADP

also a Web service version:

http://adaptorweb.lcc.uma.es/
Contributions on Adaptation

restrictive and generative
n-ary
simple properties
application to WWF

FMOODS’06, WCOP’07
IEEE TSE 34(4), 2008

issues
pruning and reduction on complete state space
data is not directly supported
application to WWF is partly manual

adaptor
9000 lines (python)
+ reuses TINA and CADP
also a Web service version:
http://adaptorweb.lcc.uma.es/
Extensions

• behavioural interfaces and vectors are extended to support data (LTS $\Rightarrow$ STS)

• implicit model encoding: **STS network**

• **deadlock-freeness** $\mu X. \langle \text{FINAL} \rangle \text{true} \lor \langle \text{true} \rangle X$
  encoded as a boolean equation system (BES)

• use of the Caesar.Solve library to perform pruning by solving the BES on-the-fly on the states of the implicit model

• model **extraction** (BPEL $\Rightarrow$ STS)
  model **implementation** (STS $\Rightarrow$ BPEL)
  - filtering to remove unimplementable parts
  - state machine pattern
Extensions

- Behavioural interfaces and vectors are extended

**Network**

\[
\text{FINAL } \iff \text{true } \lor \langle \text{true} \rangle X
\]

<table>
<thead>
<tr>
<th>Application</th>
<th>Adaptor LTS</th>
<th>State space portion explored for reduced adaptor generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>raw</td>
<td>reduced</td>
</tr>
<tr>
<td></td>
<td>states</td>
<td>trans.</td>
</tr>
<tr>
<td>eMuseum</td>
<td>21418</td>
<td>48692</td>
</tr>
<tr>
<td>music-system</td>
<td>1720</td>
<td>4368</td>
</tr>
<tr>
<td>sql-server</td>
<td>1720</td>
<td>4264</td>
</tr>
<tr>
<td>multi-file query</td>
<td>1,542</td>
<td>3,709</td>
</tr>
<tr>
<td>mail-system</td>
<td>418</td>
<td>1059</td>
</tr>
<tr>
<td>pc-store</td>
<td>253</td>
<td>472</td>
</tr>
<tr>
<td>rate-service</td>
<td>241</td>
<td>483</td>
</tr>
<tr>
<td>video-on-demand</td>
<td>149</td>
<td>231</td>
</tr>
<tr>
<td>batchsql</td>
<td>137</td>
<td>239</td>
</tr>
<tr>
<td>restauro-booking</td>
<td>94</td>
<td>108</td>
</tr>
<tr>
<td>pc-store</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

- State machine pattern
Contributions on Adaptation

restrictive **and** generative

n-ary

simple **properties**

application to **WWF**

FMOODS’06, WCOP’07

IEEE TSE 34(4), 2008
Contributions on Adaptation

restrictive and generative
n-ary
simple properties
application to WWF

FMOODS’06, WCOP’07
IEEE TSE 34(4), 2008

on-the-fly approach
direct support for data
verification of adaptation properties
(general, application-specific, controllability)
application to WS

ICSOC’08
IEEE TSE under press

Acide (external)
adaptation contract
composition requirement (ABPEL)

BPEL2STS
model extraction
composition requirement model (STS)

compositor
encoding
encoding (LOTOS)
scrutator
pruning reduction

adaptation contract
composition requirement (ABPEL)

BPEL2STS
model extraction
service models

3000 lines (python)
scrutator
1300 lines (C) + reuses CADP
BPEL2STS / STS2BPEL
1500 lines (python) / 4600 lines (Java)

compositor
3000 lines (python)
scrutator
1300 lines (C) + reuses CADP
BPEL2STS / STS2BPEL
1500 lines (python) / 4600 lines (Java)
Contributions on Adaptation

Restrictive and generative n-ary simple properties application to WWF
FMOODS’06, WCOP’07
IEEE TSE 34(4), 2008

On-the-fly approach direct support for data verification of adaptation properties (general, application-specific, controllability) application to WS
ICSOC’08
IEEE TSE under press

Open systems, local adaptors incremental adaptation
FACS’06, FMOODS’07
Contributions on Adaptation

restrictive and generative
n-ary
simple properties
application to WWF

FMOODS’06, WCOP’07
IEEE TSE 34(4), 2008

on-the-fly approach
direct support for data
verification of adaptation properties
(general, application-specific, controllability)
application to WS

ICSOC’08
IEEE TSE under press

open systems, local adaptors
incremental adaptation

FACS’06, FMOODS’07

distributed approach
fully-automatic (no contract)
application to semantic WS

FASE’08
Contributions on Adaptation

- **restrictive** and generative
  - n-ary
  - simple properties
  - application to **WWF**
  - FMOODS’06, WCOP’07
  - IEEE TSE 34(4), 2008

- **on-the-fly** approach
  - direct support for **data** verification of adaptation properties (general, application-specific, controllability)
  - application to **WS**
  - ICSOC’08
  - IEEE TSE under press

- **open** systems, **local** adaptors
  - **incremental** adaptation
  - FACS’06, FMOODS’07

- **behavioural reduction**
  - transactional equivalence
  - FORTE’07

- **distributed** approach
  - **fully-automatic** (no contract)
  - application to semantic **WS**
  - FASE’08
Agenda

1. Software architectures
2. Composition
3. Testing
4. Conclusions

You are here
Issue

• going **beyond adaptation**
  and mismatch bridging

• composing **automatically**
  services from **requirements**
  both with **conversations**
• going **beyond adaptation** and mismatch bridging

• composing **automatically** services from **requirements** both with **conversations**
Issue

- going **beyond adaptation** and mismatch bridging
- composing **automatically** services from **requirements** both with **conversations**

I need a tracking number

I want to buy something

I agree to give the product and personal info

Find me service(s) so that I can call select, purchase, ship, ...

I need a tracking number

I agree to give the product and personal info

I want to buy something
• going **beyond adaptation** and mismatch bridging

• composing **automatically** services from **requirements** both with **conversations**

---

**Issue**

- **Horizontal Mismatch**
  - **Input Data**
  - **Output Data**
  - **Vertical Mismatch**
  - **Capability Workflow**
  - **Operations**

**Diagram**

- Going beyond adaptation and mismatch bridging
- Composing automatically services from requirements both with conversations

**Examples**

- eBay: order(...), cancel(...), ship(...), bill(...), charge(...), gift(...), ack(...)
- PayPal: order(...), cancel(...), ship(...), bill(...), charge_cc(...), charge_pp(...), bill(...), finalise(...)
- pear_store: order(...), cancel(...), ship(...), bill(...), charge(...), gift(...), ack(...)
- Login(...), logout(...), get_credit(...), ask_bill(...), payment(...), payment_finalise(...)
Composition Approaches

• studied under different assumptions
  [Marconi and Pistore, 2009]

• conversations for services and for requirements
  support for input/output and for capabilities
  in few approaches:
  [Ben Mokhtar et al, 2007], [Bertoli et al, 2010]

• however, only horizontal mismatch supported in
  [Ben Mokhtar et al, 2007] (using semantics)
  and with simple assumption on service integration

• increasing use of planning for underspecification
  [Peer, 2005], [Chan et al, 2007]
Approach: Graph Planning

- polynomial construction
- all solutions of length n
- efficient tools available

Z: pre=\{a\}, effect*=\{a\}, effect+=\{b,d\}

U: pre=\{a\}, effect*=\{a\}, effect+=\{b,c\}

Y: pre=\{b\}, effect*=\{}, effect+=\{e\}

S: pre=\{c,d\}, effect*=\{}, effect+=\{e\}

initial state

exclusive actions

goal

layers
Approach: Graph Planning

- polynomial construction
- all solutions of length n
- efficient tools available

Initial state: a

Exclusive actions:
- U
- Z

Goal: solution: U; Y

Proposition: a

Invalid action: U

Precondition: b

Positive effect: e

Backtrack to solution: C

Layers:
- P0
- A1
- P1
- A2
- P2

Z: pre={a}, effect−={a}, effect+={b,c}

Y: pre={b}, effect−={}, effect+={e}

S: pre={c,d}, effect−={}, effect+={e}

U; pre={a}, effect−={a}, effect+={b,c}

U; pre={b}, effect−={}, effect+={e}

U; pre={c,d}, effect−={}, effect+={e}
Approach: Graph Planning

- polynomial construction
- all solutions of length $n$
- efficient tools available

**Solution:**
- $U; Y$

**What if $U$ fails?**
- Other solution: $Z; Y$

**Effective Construction:**
- $Z$: $pre=\{a\}$, $effect^{-}=\{a\}$.

**Exclusive Actions:**
- $d$

**Initial State:**
- $a$

**Goal:**
- $e$

**Layers:**
- $P_0$, $A_1$, $P_1$, $A_2$, $P_2$

**Propositions:**
- $S$: $pre=\{c,d\}$, $effect^{-}=\{\}$, $effect^{+}=\{e\}$
Approach: Graph Planning

- polynomial construction
- all solutions of length n
- efficient tools available

---

- exclusive actions
- goal
- initial state

- U
- Z
- Y
- S
- a
- b
- c
- d
- e

- solution: (U, Y)
- what if U fails? (Z, Y)
- what if Y fails? (no solution, but part of the graph is still valid)

- proposition
- pre={a}, effect={a}

- layers
- backtrack to solution
Approach: Technique

- **semantic typing** of service operations
  input data + output data + capabilities

- encoding **data adaptation**
  
  **casting** $d \prec d'$ enables cast: $d \rightarrow d'$

  **(de)composition** $d = \{d_i\}$ enables (de)comp: $\{d_i\} \leftrightarrow d$

- encoding **conversations**
  workflow to Petri net mapping [Kiepuszewski, 2003] adapted
  
  - to map workflows to graph planning actions
  - to enable/disable capabilities (requirement + services)
  - to enable/disable operations (services)

- encoding **operations**
  capability-enabled + inputs $\rightarrow$ capability-done + outputs
semantic data description

- shipping addr
- billing addr
- address
  - user addr
  - CC info
    - CC holder
    - CC number
  - user info
    - user name
  - PIM wallet
    - paypal info
      - paypal pwd
      - paypal login
    - amazon info
      - amazon login
      - amazon pwd
- order amount
- product price
- product specs
- pear product
  - pear product info
  - product
    - ePhone
    - ePad
I need a tracking number.
I want to buy something.
I agree to give the product and personal info, BUT FOR CC.

order_fianlization
order
payment
billing_setup
shipping_setup
product_selection
requirements

service_operations
service_conversations

order_info
product_info
billing_info
shipping_info
CC_info
I need a tracking number

I agree to give the product and personal info BUT FOR CC

I want to buy something

any solution?
I need a tracking number

I want to buy something

I agree to give the product and personal info BUT FOR CC

yes

requirements

product_selection

shipping_setup

billing_setup

payment

order_finalization
I want to buy something

I need a tracking number

I agree to give the product and personal info BUT FOR CC
receive(user,op,{ePad,user_info}) ;
flow {
[pear_product := cast(ePad)],
[user_name := user_info.user_name; user_address := user_info.user_address; pim_wallet := user_info.pim_wallet]
} ;
flow {
[product := cast(pear_product)],
[shipping_addr := cast(user_address)],
[paypal_info := pim_wallet.paypal_info; amazon_info := PIM_wallet.amazon_info]
} ;
flow {
[{e_sessionid} := invoke(w2,order,{product})],
[paypal_login := paypal_info.paypal_login; paypal_pwd := paypal_info.paypal_pwd]
} ;
{p_sessionid} := invoke(w3,login,{paypal_login,paypal_pwd}) ;
{order_amount} := invoke(w2,ship,{shipping_addr,e_sessionid}) ;
{paypal_trans_id} := invoke(w3,get_credit,{order_amount,p_sessionid}) ;
invoke(w2,charge_pp,{paypal_trans_id,e_sessionid}) ;
invoke(w3,ask_bill,{user_address,p_sessionid}) ;
invoke(w3,logout,{p_sessionid}) ;
{tracking_num} := invoke(w2,finalize,{e_sessionid}) ;
reply(user,op,{tracking_num});
Contributions on Composition

services with **conversations**
requirement with **conversation**
data flow and **control** flow
horizontal + vertical **adaptation**
application to **WS**

ICSOC’08, ISoLa’10

**Pycompose**

1200 lines (python)
+ reuses graphplan, PDDLGraphplan, or Blackbox

**WF2BPEL**

7900 lines (Java)
Contributions on Composition

services with conversations
requirement with conversation
data flow and control flow
horizontal + vertical adaptation
application to WS

ICSO'C08, ISoLa'10

repair vs. recomposition:
repaired solution quality =
computation time
application to WS

ICWS'10, ICSOC'10

Contributions on Composition

standard recomposition

PGA
5200 lines (Java)

repair

repair vs. recomposition:
repaired solution quality =
computation time
application to WS

ICWS'10, ICSOC'10
Agenda

software architectures

adaptation

composition

you are here

conclusions

testing
• verification (of design artifacts) and testing are **complementary**

\[
\text{verification requirements} \leftrightarrow \text{model}
\]

\[
\text{testing model} \leftrightarrow \text{implementation}
\]

• the need for testing **increases** in a development process based on **reuse** and with **dynamic binding**
Issue
BPEL expressiveness including:
- workflow features
- XPath assignments
- communication
- faults, timers, ...
BPEL expressiveness including:
- workflow features
- XPath assignments
- communication
- faults, timers, ...

structure (partnership)

structured data (lots of)
Issue

BPEL expressiveness including:
- workflow features
- XPath assignments
- communication
- faults, timers, ...

structure (partnership)
correlation data
structured data (lots of)

BPEL expressiveness including:
- workflow features
- XPath assignments
- communication
- faults, timers, ...
Testing Approaches

• involvement of the testing community
  [Bozkurt et al, 2010], [Russli et al, 2011]

• WSDL-testing does not support conversations
  white-box testing is not realistic wrt. reuse

• the treatment of data makes the difference
  WSDL-S ➞ EFSM + theorem prover [Sinha and Paradkar, 2006]
  BPEL ➞ CFG + symbolic execution + solver [Yan et al, 2006]
  BPEL ➞ Promela + model-checker [Zheng et al, 2007]
  UML ➞ STS + online approach [Frantzen et al, 2009]

• test passing in [Zheng et al, 2007] and [Frantzen et al, 2009]

• combining on-line approach + symbolic execution
  perspective of [Frantzen et al, 2009]
Approach: Technique (1/4)

- transformation ABPEL specification → STS

based on an earlier work by Mateescu and Rampacek (2008)

use of STS instead of dtLTS

reused as is: time, throw, sequence

extensions:
- data in constructs
- correlation and m. faults

<table>
<thead>
<tr>
<th>BPEL</th>
<th>STS</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>empty $\rightarrow$ 0</td>
</tr>
<tr>
<td>time</td>
<td>$p \xrightarrow{\text{time, rec(pl,o,v_{in}), send(pl,o,v_{out})}} p$</td>
</tr>
<tr>
<td>assign$^+$</td>
<td>$p_{1} := p_{2} \xrightarrow{\tau} p_{1} := p_{2} \rightarrow$ empty</td>
</tr>
<tr>
<td>throw</td>
<td>$\forall e \in Ex \text{ throw } e \rightarrow 0$</td>
</tr>
<tr>
<td>rec$^+$</td>
<td>$\text{rec(pl,o,v_{in})$ pl_{o?eq_{in}} / v_{in} := \text{eq_{in}} \rightarrow$ empty with $\exists i \in O(S_{pl}), in(o) = m$</td>
</tr>
<tr>
<td>send$^+$</td>
<td>$\text{send(pl,o,v_{out})$ pl_{o?eq_{out}} / v_{out} := \text{eq_{out}} \rightarrow$ empty with $\exists i \in O(S_{pl}), out(o) = m$</td>
</tr>
</tbody>
</table>
| receive$^*$               | $\text{receive(pl,o,v_{in}) = rec(pl,o,v_{in})$}
| reply$^*$                 | $\text{reply(pl,o,v_{out}) = send(pl,o,v_{out})$}
| invoke$^+$                | $\text{invoke(pl,o,v_{in}) = send(pl,o,v_{out})$}
| sequence$^*$              | $\forall a \in Ev \{\sqrt{a}\}$, $P\xrightarrow{p[a]} A, P \rightarrow Q$ |
| if$^*$                    | if $c$ then $P$ else $Q$ $\xrightarrow{[c]} \tau$ $P$ |
| while$^*$                 | if $c$ then $P$ else $Q$ $\xrightarrow{[c]} \tau$ $Q$ |
| scope$^*$                 | $\text{let } EH = \{(pl_{i},a_{i},v_{i})_{i \in I}\}$, $O_{I} = \{(pl_{i},a_{i},v_{i}) | (pl_{i},a_{i},v_{i}) \in O_{I}\}$, $E_{j} = \{e_{j} | e_{j} \in \text{in:}$ |
| event handler             | $\forall (pl_{i},a_{i},v_{i}) \in O_{I}$, $\text{with } \exists i_{j} \in O(S_{pl_{i}}), in(o_{i}) = m$ |
| passing                   | $\text{scope}(P,EH) \xrightarrow{\tau \wedge E_{j}}$ |
| handler                   | $\text{scope}(P,EH) \xrightarrow{\tau \wedge E_{j}}$ |
| supported fault termination | $\text{scope}(P,EH) \xrightarrow{\tau \wedge E_{j}}$ |
| message faults$^+$        | $\text{pick}(E) = \text{scope}(time,E)$ |
| flow$^+$                  | $\text{message faults}^+, \text{flow}^+, \text{until}^+$: see [14] |
| until$^+$                 | $\text{until}^+$: see [14] |
Approach: Technique (2/4)

- **unfolding** \( \text{STS} \rightarrow \text{SET} \)

- Symbolic Execution
Approach: Technique (2/4)

• **unfolding** $\text{STS} \rightarrow \text{SET}$

• Symbolic Execution

Program with concrete variables

```
var x, y, z: Nat
comp? x, y
if (x > y)
  { z := x + 1 }
else
  { z := y + 2 }
comp! z
```

Size without symbolic execution: [66,565; 132,612]
Approach: Technique (2/4)

- **unfolding** \( \text{STS} \rightarrow \text{SET} \)

- Symbolic Execution

```
var x, y, z: Nat
comp? x, y ;
if (x > y)
  { z := x + 1 }
else
  { z := y + 2 } ;
comp! z ;
```

1. \( \{ x:v0, y:v1, z:v2 \} \)
   true
   comp?v3,v4

2. \( \{ x:v3, y:v4, z:v2 \} \)
   true

3. \( \{ x:v3, y:v4, z:v2 \} \)
   \( v3 > v4 \)

4. \( \{ x:v3, y:v4, z:v2 \} \)
   \( \neg (v3 > v4) \)

5. \( \{ x:v3, y:v4, z:v5 \} \)
   \( v3 > v4 \land v5 = v3 + 1 \)

6. \( \{ x:v3, y:v4, z:v5 \} \)
   \( \neg (v3 > v4) \land v5 = v4 + 2 \)

7. \( \{ x:v3, y:v4, z:v5 \} \)
   \( v3 > v4 \land v5 = v3 + 1 \)

8. \( \{ x:v3, y:v4, z:v5 \} \)
   \( \neg (v3 > v4) \land v5 = v4 + 2 \)

9. \( \{ x:v3, y:v4, z:v5 \} \)
   \( v3 > v4 \land v5 = v3 + 1 \)

10. \( \{ x:v3, y:v4, z:v5 \} \)
    \( \neg (v3 > v4) \land v5 = v4 + 2 \)

Path condition

Symbolic variables

Program with concrete variables

Size without symbolic execution:
\([66,565; 132,612]\)

Size with symbolic execution:
\([10; 9]\)
Approach: Technique (2/4)

- **unfolding** \( \text{STS} \rightarrow \text{SET} \)

- Symbolic Execution

Program with concrete variables:

\[
\begin{align*}
1 & \{x:v0, y:v1, z:v2\} \\
& \text{true} \\
\text{comp}?v3,v4 \\
2 & \{x:v3, y:v4, z:v2\} \\
& \text{true} \\
\text{comp}?x,y ; \\
& \{ z:=x+1 \} \text{ if } (x>y) \text{ else } \{ z:=y+2 \} ; \\
& \text{comp!}z ; \\
3 & \{x:v3, y:v4, z:v2\} \\
& \text{v3>v4} \\
\text{comp!}v5 \\
4 & \{x:v3, y:v4, z:v2\} \\
& \text{not(v3>v4)}
\end{align*}
\]

Path condition:

\[
\begin{align*}
1 & \{x:v0, y:v1, z:v2\} \text{ true} \\
\text{comp}?v3,v4 \\
2 & \{x:v3, y:v4, z:v2\} \text{ true} \\
\text{var } x,y,z: \text{Nat} \text{ comp}?x,y ; \\
& \{ z:=x+1 \} \text{ if } (x>y) \text{ else } \{ z:=y+2 \} ; \\
& \text{comp!}z ; \\
3 & \{x:v3, y:v4, z:v2\} \text{ v3>v4} \\
\text{comp!}v5 \\
4 & \{x:v3, y:v4, z:v2\} \text{ not(v3>v4)}
\end{align*}
\]

Symbolic variables:

\[
\begin{align*}
5 & \{x:v3, y:v4, z:v5\} \\
& \text{v3>v4} \land v5=v3+1 \\
\text{comp!}v5 \\
6 & \{x:v3, y:v4, z:v5\} \\
& \text{not(v3>v4)} \land v5=v4+2 \\
\text{comp!}v5 \\
7 & \{x:v3, y:v4, z:v5\} \\
& \text{v3>v4} \land v5=v3+1 \\
\text{path condition} \\
8 & \{x:v3, y:v4, z:v5\} \\
& \text{not(v3>v4)} \land v5=v4+2
\end{align*}
\]

Size without symbolic execution:

\[\{66,565; 132,612\}\]

But there is no magic: solvers are required to use SET

Size with symbolic execution:

\[\{10; 9\}\]
• **unfolding** STS → SET

• Symbolic Execution

• SET size issue:
  • unfolding up to some length k
  • cutting **infeasible paths**
  • use of an **inclusion criterion** over SET nodes

• **online** algorithm to avoid false positives
• online testing

Algorithm 1: Online Testing Algorithm

Data: SET + a distinguished path p, path p = n_1 l_1 n_2 l_2 ... l_{k-1} n_k ;

begin
    π = π_k; i := 1; rtr := Pass;
    while i < k and rtr = Pass do
        switch l_i do
            case USER.e?x_s
                val := (SOLVE(π)[x_s]);
                try {send (e(val)); π := π ∧ x_s = val;}
                catch (e ∈ Ex) { rtr := Fail; }
            case USER.e!x_s
                start TAC;
                try {receive (e(val)); π = π ∧ (x_s = val);
                     if ¬SOLVE(π) then rtr := Fail; }
                catch (timeout_TAC) {rtr := Fail;}
                catch (receive e') { if e' ∈ may(η_i) then rtr := Inconclusive;
                                   else rtr := Fail;}
            case χ
                wait(1 unit of time);
            otherwise
                skip;
        i := i + 1;
    return rtr;
end
Approach: Technique (4/4)

- online testing

```plaintext
Algorithm 1: Online Testing Algorithm

Data: SET + a distinguished path p, path p = n_1 l_1 n_2 l_2 ... l_{k-1} n_k;

begin
\pi = \pi_k; i := 1; rtr := Pass;
while i < k and rtr = Pass do
switch l_i do
    case USER.e?x_s
        val := \text{SOLVE} (\pi[x_s]);
        try {send (\pi[\eta_i]); \pi = \pi \land x_s = val;}
        catch (timeout_TAC) {rtr := Fail;}
    case USER.e'
        start TAC;
        try {receive (e(val)); \pi = \pi \land (x_s = val);
            if \neg \text{SOLVE} (\pi) then rtr := Fail; }
        catch (timeout_TAC) {rtr := Fail;}
        catch (receive e') { if e' \in \text{may} (\eta_i) then rtr := Inconclusive;
            else rtr := Fail;}
    \pi := \pi \land \chi
    \pi := \pi \land \chi
    i := i + 1;
end

test passing (SoapUI)

<path cond. solving (Z3)
```
• online testing

Algorithm 1: Online Testing Algorithm

Data: SET + a distinguished path p, path p = n₁ l₁ n₂ l₂ ... lₖ₋₁ nₖ ;
begin
  π = πₖ ; i := 1 ; rtr := Pass ;
  while i < k and rtr = Pass do
    switch lᵢ do
      case USER.e?xᵦ
        val := SOLVE(π)[xᵦ];
        try {send (π, val) ; π := π ∧ xᵦ = val} ;
        catch (τ) ;
      case USER.start TAC ;
        try {receive (e(val)) ; π := π ∧ xᵦ = val ;
          if ~SOLVE(π) then rtr := Fail ;}
        catch (τ) ;
      catch (τ) ;
      case (func) ;
        wait(1 unit of time) ;
      otherwise
        skip ;
    end
    i := i + 1 ;
  return rtr ;
end

Approach: Technique (4/4)

- test passing (SoapUI)
- path cond. solving (Z3)
- output checking (Z3)
Contributions on Testing

L. Bentakouk PhD thesis

**black-box symbolic** approach
active **online** testing
application to **WS orchestration**

**TESTCOM/FATES’09**
**TAP’11**

---

**SWST**

- orchestration specification (WSDL+BPEL)
- model extraction
- orchestration model (STS)
- test model (STS)
- symbolic execution
- symbolic model symbolic sem. (SET)
- test model symbolic sem. (SET)
- symbolic execution
- extraction
- symbolic test cases (STC)
- constraint solving
- [ coverage criterion ]
- [ path length criterion ]
- online tester (oracle)
- test verdict
- test model symbolic sem. (SET)

---

**SWST**
2100 lines (python)
2200 lines (Java)
+ reuses Z3, SoapUI

---

**UML2CSP** (external)
**Z3** (external)
Contributions on Testing

L. Bentakouk PhD thesis

black-box symbolic approach
active online testing
application to WS orchestration

TESTCOM/FATES’09
TAP’11

H.N. Nguyen PhD thesis

black-box testing
passive offline testing
application to WS choreography
local and global conformance

SAC’12

Global conformance (synthesis included)

Testing Modules
6000 lines (Java)
Monitor
680 lines (Java)

Graph showing the global conformance with different fault scenarios.
Agenda

software architectures
adaptation
composition

testing
conclusions

you are here
Conclusions

«in-the-large» works

<table>
<thead>
<tr>
<th>issue</th>
<th>type</th>
<th>DL&lt;sup&gt;in&lt;/sup&gt;</th>
<th>DM&lt;sup&gt;in&lt;/sup&gt;</th>
<th>WM</th>
<th>DM&lt;sup&gt;out&lt;/sup&gt;</th>
<th>DL&lt;sup&gt;out&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptation</td>
<td>centralised</td>
<td>WWF</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>WWF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WSDL BPEL</td>
<td>Symbolic</td>
<td>Process Algebra</td>
<td>Symbolic</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>SAWSDL BPEL</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workflow</td>
<td>Event Structure</td>
<td>Petri Net</td>
<td>Event Structure</td>
<td></td>
</tr>
<tr>
<td>composition + repair</td>
<td>centralised</td>
<td>WSDL Workfow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OWL Workflow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td></td>
</tr>
<tr>
<td>testing</td>
<td>centralised</td>
<td>ABPEL</td>
<td>Symbolic</td>
<td>Symbolic</td>
<td>Symbolic</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Trans. Syst.</td>
<td>Exec. Tree</td>
<td>Test Cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>Chor SOAP logs</td>
<td>Trace</td>
<td>Trace</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>realisability</td>
<td>distributed</td>
<td>BPMN 2.0</td>
<td>Workflow</td>
<td>Process Algebra</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Supported by
Project PERSO (leader)
adaptation & composition
Project WebMoV
testing

FMOODS’06, FACS’06, FMOODS’07, FACS’07, FORTE’07, FASE’08, ICSOC’08
IEEE TSE 34(4), 2008
IEEE TSE under press

ICSOC’08, ISoLa’10
ICWS’10, ICSOC’10

TESTCOM’09, TAP’11
SAC’12

SAC’12
Conclusions

1. A general approach

<table>
<thead>
<tr>
<th>Issue</th>
<th>Type</th>
<th>$\text{DL}^{\text{in}}$</th>
<th>$\text{DM}^{\text{in}}$</th>
<th>$\text{WM}$</th>
<th>$\text{DM}^{\text{out}}$</th>
<th>$\text{DL}^{\text{out}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation</td>
<td>Centralised</td>
<td>WWF</td>
<td>Trans. Syst. Petri Net</td>
<td>Trans. Syst.</td>
<td>WWF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BPEL</td>
<td></td>
<td></td>
<td></td>
<td>BPEL</td>
</tr>
<tr>
<td>Composition + Repair</td>
<td>Centralised</td>
<td>WSDL</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OWL Workflow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABPEL</td>
<td>Symbolic Trans. Syst.</td>
<td>Symbolic Exec. Tree</td>
<td>Symbolic Test Cases</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td>Distributed</td>
<td>Chor SOAP logs</td>
<td>Trace</td>
<td>Trace</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Realisability</td>
<td>Distributed</td>
<td>BPMN 2.0</td>
<td>Workflow</td>
<td>Process Algebra</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
## Conclusions

### I. A general approach
- Domain Specific Languages / Models

#### 1. Domain Specific Languages / Models

<table>
<thead>
<tr>
<th>issue</th>
<th>type</th>
<th>DL\textsuperscript{in}</th>
<th>DM\textsuperscript{in}</th>
<th>WM</th>
<th>DM\textsuperscript{out}</th>
<th>DL\textsuperscript{out}</th>
</tr>
</thead>
<tbody>
<tr>
<td>distributed</td>
<td></td>
<td>SAWSDL</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>Trans. Syst.</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BPEL</td>
<td>Event Structure</td>
<td>Petri Net</td>
<td>Event Structure</td>
<td></td>
</tr>
<tr>
<td>composition +</td>
<td>centralised</td>
<td>WSDL</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td>BPEL</td>
</tr>
<tr>
<td>repair</td>
<td></td>
<td>OWL Workflow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td></td>
</tr>
<tr>
<td>testing</td>
<td>centralised</td>
<td>ABPEL</td>
<td>Symbolic Trans. Syst.</td>
<td>Symbolic Exec. Tree</td>
<td>Symbolic Test Cases</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chor</td>
<td>Trace</td>
<td>Trace</td>
<td>Trace</td>
<td></td>
</tr>
<tr>
<td>distributed</td>
<td></td>
<td>SOAP logs</td>
<td>Trace</td>
<td>Trace</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>realisability</td>
<td>distributed</td>
<td>BPMN 2.0</td>
<td>Workflow</td>
<td>Process Algebra</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**DOMAIN SPECIFIC**
- Input Domain Language DL\textsuperscript{in}
- Output Domain Language DL\textsuperscript{out}
- Input Domain Model DM\textsuperscript{in}
- Output Domain Model DM\textsuperscript{out}
- Input Working Model WM\textsuperscript{in}
- Output Working Model WM\textsuperscript{out}
- Input Working Language WL\textsuperscript{in}
- Output Working Language WL\textsuperscript{out}

**ENCODING**
- Working Tool (ad-hoc or reused)
## Conclusions

1. A general approach
   - Domain Specific Languages / Models
   - Issue Specific Languages / Models

### Encoding

<table>
<thead>
<tr>
<th>Issue</th>
<th>Type</th>
<th>Domain Language Input</th>
<th>Domain Model Input</th>
<th>Working Model Input</th>
<th>Working Language Input</th>
<th>Working Tool (ad-hoc or reused)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation</td>
<td>Centralised</td>
<td>WWF</td>
<td>Trans. Syst.</td>
<td>Process Algebra</td>
<td>Petri Net</td>
<td>WWF</td>
</tr>
<tr>
<td></td>
<td>Distributed</td>
<td>WSDL BPEL</td>
<td>Symbolic Trans. Syst.</td>
<td>Symbolic Trans. Syst.</td>
<td>BPEL</td>
<td></td>
</tr>
<tr>
<td>Composition + Repair</td>
<td>Centralised</td>
<td>WSDL OWL Workflow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td>BPEL</td>
</tr>
<tr>
<td>Testing</td>
<td>Centralised</td>
<td>ABPEL</td>
<td>Symbolic Trans. Syst.</td>
<td>Symbolic Exec. Tree</td>
<td>Symbolic Test Cases</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td>Distributed</td>
<td>Chor SOAP logs</td>
<td>Trace</td>
<td>Trace</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Realisability</td>
<td>Distributed</td>
<td>BPMN 2.0</td>
<td>Workflow</td>
<td>Process Algebra</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
## Conclusions

### I. A general approach

- Domain Specific Languages / Models
- Issue Specific Languages / Models

**Encodings** $\text{DM} \leftrightarrow \text{WM}$

<table>
<thead>
<tr>
<th>Domain and Issue Specific</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Language DL\text{in}</td>
<td></td>
<td>Domain Language DL\text{out}</td>
</tr>
<tr>
<td>Domain Model DM\text{in}</td>
<td></td>
<td>Domain Model DM\text{out}</td>
</tr>
<tr>
<td>Working Model WM\text{in}</td>
<td></td>
<td>Working Model WM\text{out}</td>
</tr>
<tr>
<td>Working Language WL\text{in}</td>
<td></td>
<td>Working Language WL\text{out}</td>
</tr>
</tbody>
</table>

### Development

1. A general approach
2. Domain Specific Languages / Models
3. Issue Specific Languages / Models

**Encodings** $\text{DM} \leftrightarrow \text{WM}$

<table>
<thead>
<tr>
<th>Issue</th>
<th>Type</th>
<th>DL\text{in}</th>
<th>DM\text{in}</th>
<th>WM</th>
<th>DM\text{out}</th>
<th>DL\text{out}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation</td>
<td>Centralised</td>
<td>awf2lts  (U. Malaga)</td>
<td>adaptor</td>
<td>Cadp, TINA</td>
<td>Its2awf  (U. Malaga)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Distributed</td>
<td>(ongoing)</td>
<td>LOLA</td>
<td>(ongoing)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Composition + Repair</td>
<td>Centralised</td>
<td>pycompose PGA</td>
<td>Graphplan PDDLGP Blackbox</td>
<td>WF2BPEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td>Centralised</td>
<td>BPEL2STS</td>
<td>SWST</td>
<td>SWST</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Distributed</td>
<td>Chor, Monitoring, and Synthesis Modules</td>
<td>Conformance Modules</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Realisability</td>
<td>Distributed</td>
<td>BPMN2Py Py2LNT</td>
<td>CADP</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Conclusions

II. no unique language
- no unique model
- we have to face existing DL\textsuperscript{in}/DL\textsuperscript{out}
- DM\textsuperscript{in}/WM/DM\textsuperscript{out} have specificities

### transition systems
+ simplicity, symbolic extensions
+ tool support
- //ism implementation

### process algebras
+ on-the-fly tool support
- non symbolic models

### event structures
+ workflow encoding
+ //ism implementation
- loops in behaviours
- tool support

### Petri nets
+ interleaving or true concurrency
+ interaction or resource viewpoint
+ workflow encoding
+ //ism implementation
+ tool support

<table>
<thead>
<tr>
<th>Issue</th>
<th>type</th>
<th>DL\textsuperscript{in}</th>
<th>DM\textsuperscript{in}</th>
<th>WM</th>
<th>DM\textsuperscript{out}</th>
<th>DL\textsuperscript{out}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>centralised</td>
<td>WWF</td>
<td>Trans. Syst.</td>
<td>Petri Net</td>
<td>Trans. Syst.</td>
<td>WWF</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>Workflow</td>
<td>Event Structure</td>
<td>Petri Net</td>
<td>Event Structure</td>
<td>Workflow Event Structure</td>
</tr>
<tr>
<td></td>
<td>centralised</td>
<td>WSDL/BSWLPOWL Workflow</td>
<td>Planning</td>
<td>Planning</td>
<td>Planning</td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td>centralised</td>
<td>ABPEL</td>
<td>Symbolic Trans. Syst.</td>
<td>Symbolic Exec. Tree</td>
<td>Symbolic Test Cases</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>Chor SOAP logs</td>
<td>Trace</td>
<td>Trace</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>distributed</td>
<td>BPMN 2.0</td>
<td>Workflow</td>
<td>Process Algebra</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Perspectives

H.N. Nguyen PhD thesis

eternal peer composition

online and distributed approach using test, diagnosis, and repair

model retrieval (concurrency + data)

R. Khéfifi PhD thesis

resource-centric composition

industrial application

new applicative domain (personal info.)

REST vs SOAP services

submitted project

verification of BPMN 2.0 choreographies

industrial application language expressiveness compositional verification

event structures

coloured Petri nets

solvers / provers

adaptation

rich semantics (pre/post)

true concurrency

tools and interconnection

WS versions (some already exist)
Formal Model-Based Approaches for the Development of Composite Systems

MeFoSyLoMa Seminar
(originally, Habil. thesis defense, Nov. 24th, 2011)

Pascal Poizat
Université d’Evry Val d’Essonne;
LRI CNRS UMR 8623 et Université Paris-Sud 11

Evry, February 17th, 2012