Adaptation of Service Protocols using Process Algebra and On-the-Fly Reduction Techniques

Radu Mateescu[1], **Pascal Poizat**[2], Gwen Salaün[3]

[1]INRIA / VASY Project-Team, France

[2]Univ. Évry, France and LRI, France
pascal.poizat@lri.fr
http://www.lri.fr/~poizat
this work has been done while at IBISC and INRIA / ARLES Project-Team, France

[3]Univ. Málaga, Spain

ICSOC'2008
December 1–5, 2008

Motivation (1/2)

- user requirements realized
  through the automatic orchestration of available services
- complex services have conversations
  these correspond to the service behavioral interfaces
- yet, services may present mismatch
  this prevents composition

solution ?

Motivation (2/2)

- software adaptation is a possible means to solve mismatch out
  mismatch corresponds to deadlock in service exchanges
  adaptors stand in-between services to avoid deadlock
  e.g., [Brogi and Popescu, ICSOC'06], [Motahari-Nezhad *et al.*, WWW'07],
  [Canal *et al.*, IEEE TSE 34(4), 2008], [Inverardi and Tivoli, SCP 71, 2008]

- adaptation features
  - tools (automation)
  - adaptation contracts (abstract requirements)
  - prune interactions leading to deadlocks (restrictive adaptation)
  - store and reorders messages (generative adaptation)

still, the adaptation process is complex (exponential wrt. service models)

Motivation (2/2)

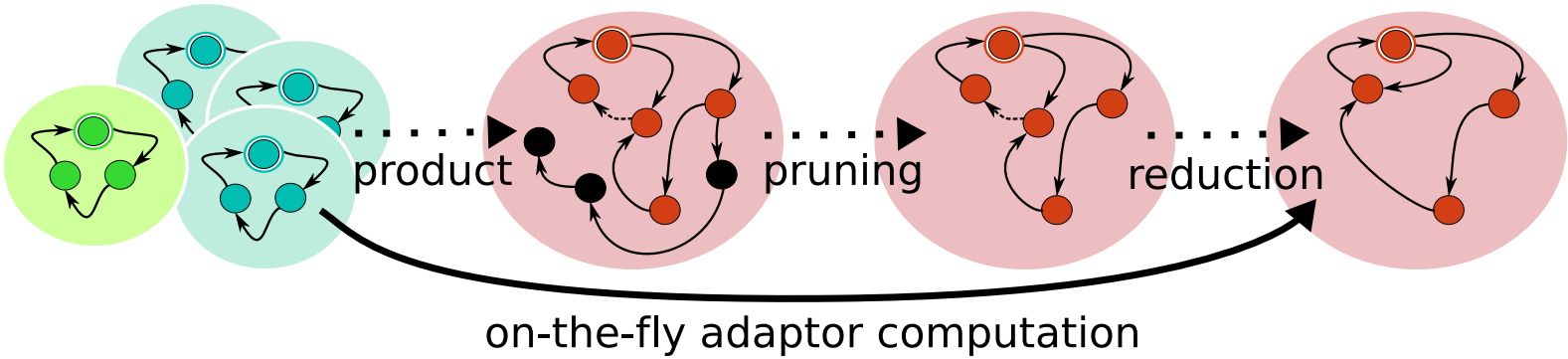
- software adaptation is a possible means to solve mismatch out
  mismatch corresponds to deadlock in service exchanges
  adaptors stand in-between services to avoid deadlock
  e.g., [Brogi and Popescu, ICSOC'06], [Motahari-Nezhad *et al.*, WWW'07],
  [Canal *et al.*, IEEE TSE 34(4), 2008], [Inverardi and Tivoli, SCP 71, 2008]

- adaptation features
  - tools (automation)
  - adaptation contracts (abstract requirements)
  - prune interactions leading to deadlocks (restrictive adaptation)
  - store and reorders messages (generative adaptation)

still, the adaptation process is complex (exponential wrt. service models)

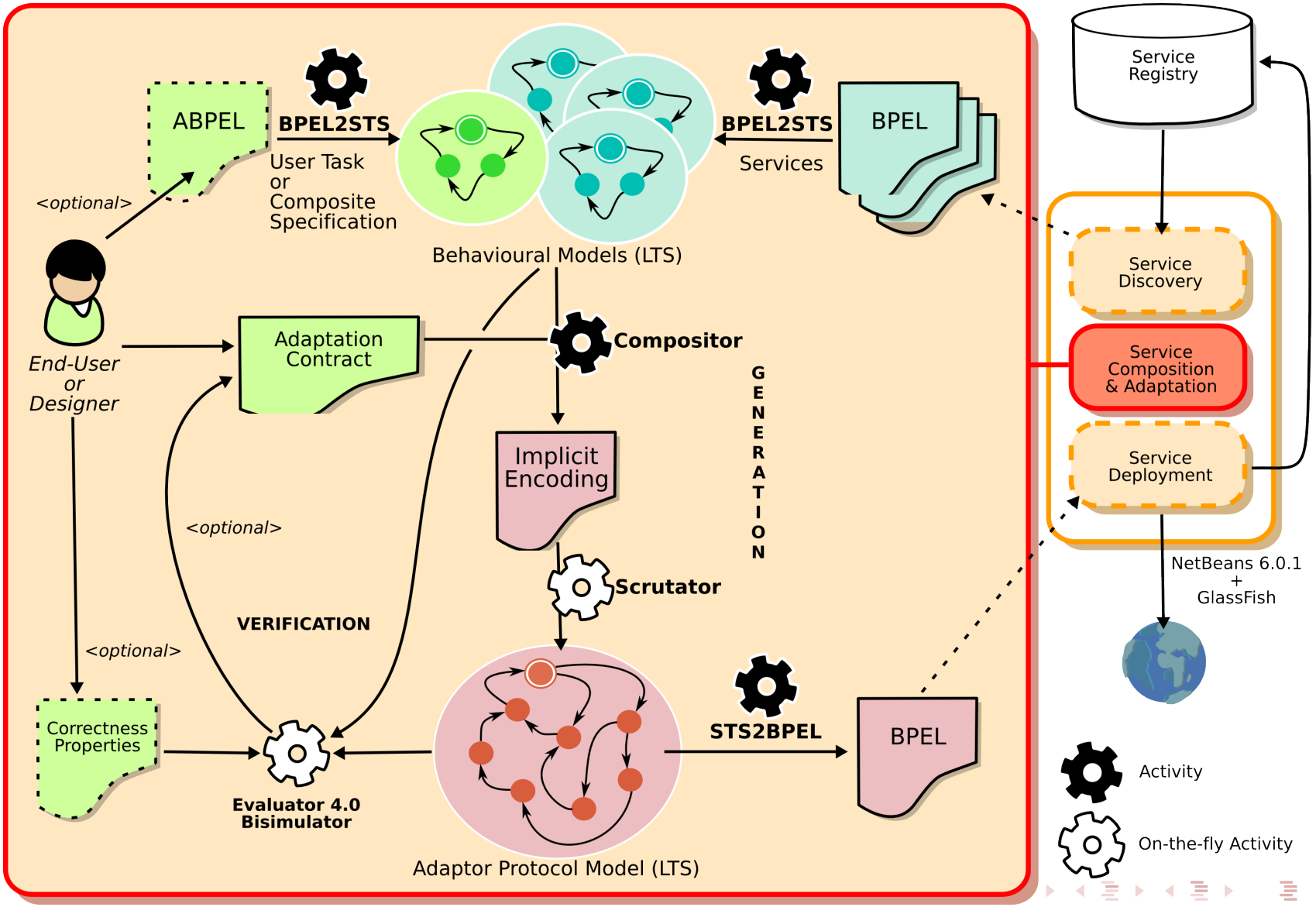
Objectives

- perform pruning and reduction efficiently



- verify adaptor-orchestrator correctness
  as required by handcrafted contracts and pruning
- implement models
  BPEL orchestrators

Outline


ABPEL
BPEL2STS
User Task
or
Composite
Specification
BPEL2STS
Services
BPEL
Service
Registry
<optional>
Behavioural Models (LTS)
Service
Discovery
End-User
or
Designer
Adaptation
Contract
Compositor
Service
Composition
& Adaptation
Service
Deployment
Implicit
Encoding
GENERATION
<optional>
Scrutator
NetBeans 6.0.1
+
GlassFish
VERIFICATION
<optional>
Correctness
Properties
STS2BPEL
BPEL
Evaluator 4.0
Bisimulator
Adaptor Protocol Model (LTS)
Activity
On-the-fly Activity

Service Model

Service Conversation

A service conversation is a Labelled Transition System (LTS) *i.e.* a tuple (Events, States, Initial state, Final states, Transitions)

Event

An event for a service S_i has the form $S_i : M\ d\ P$ where either:

- $d = *$, $M = \tau$, P is empty (internal action)
- $d = ?$, M is a S_i input message name, and $P = V_1, \ldots, V_n$ (reception)
- $d = !$, M is a S_i output message name, and $P = V_1, \ldots, V_n$ (emission)

V_i are typed variables (names can be omitted)

$b : debitQuery?tid, string, double$ $mf : exitReply!double, string$

can be obtained from service descriptions (ABPEL, BPEL, WWF)

e.g., [Fu *et al.*, WWW'04], [Salaün *et al.*, ICWS'04], [Ferrara, ICSOC'04], [Foster, ICSOC'08]

Service Model

Service Conversation

A service conversation is a Labelled Transition System (LTS) *i.e.* a tuple (Events, States, Initial state, Final states, Transitions)

Event

An event for a service S_i has the form $S_i : M\ d\ P$ where either:

- $d = *$, $M = \tau$, P is empty (internal action)
- $d = ?$, M is a S_i input message name, and $P = V_1, \ldots, V_n$ (reception)
- $d = !$, M is a S_i output message name, and $P = V_1, \ldots, V_n$ (emission)

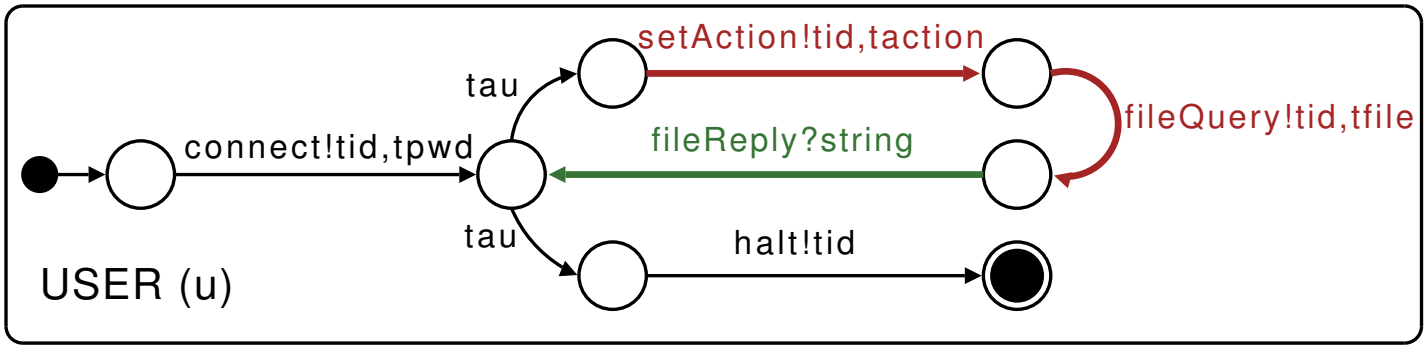
V_i are typed variables (names can be omitted)

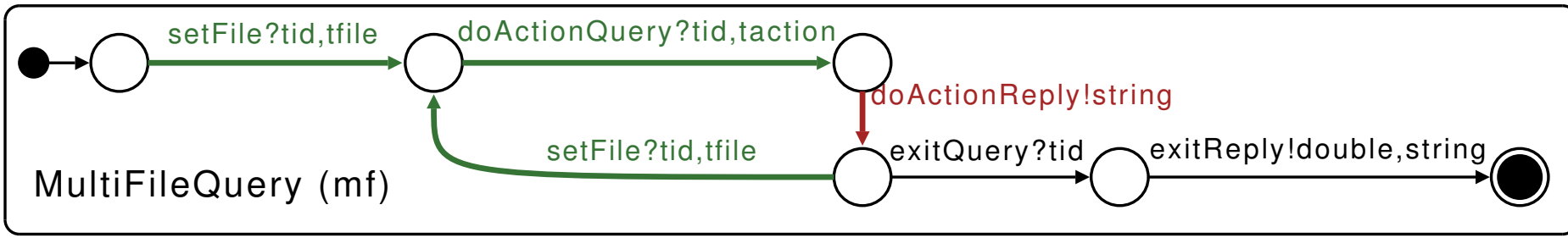
$b : debitQuery?tid, string, double$ $mf : exitReply!double, string$

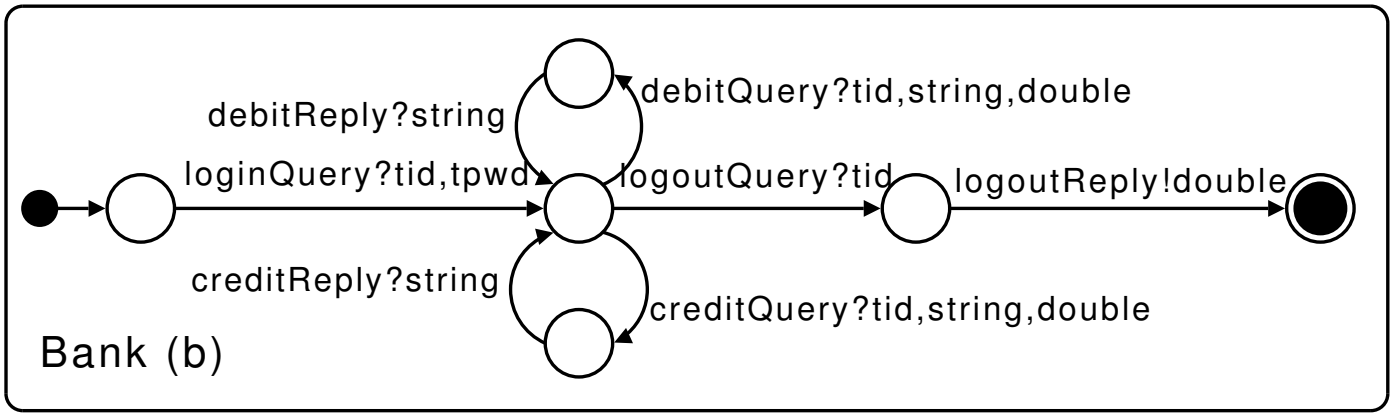
can be obtained from service descriptions (ABPEL, BPEL, WWF)

e.g., [Fu *et al.*, WWW'04], [Salaün *et al.*, ICWS'04], [Ferrara, ICSOC'04], [Foster, ICSOC'08]

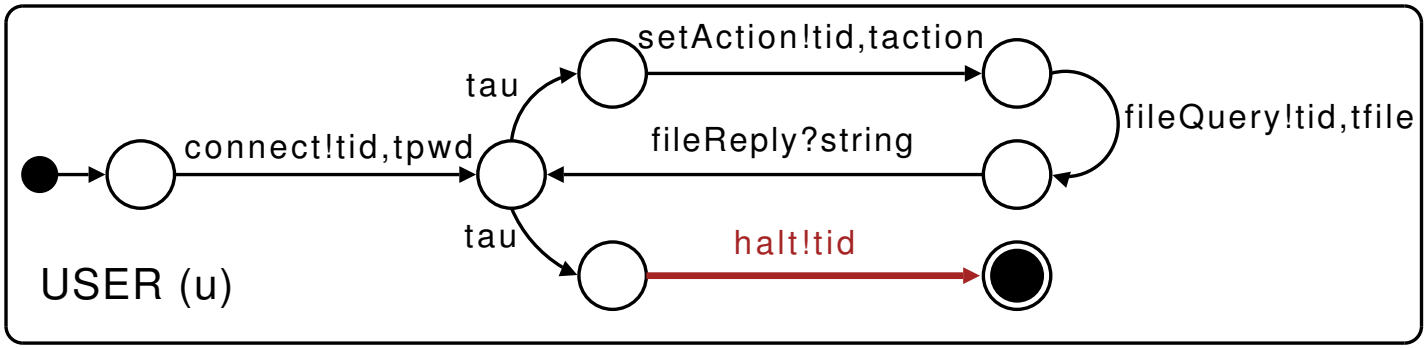
Example (Service Conversations)

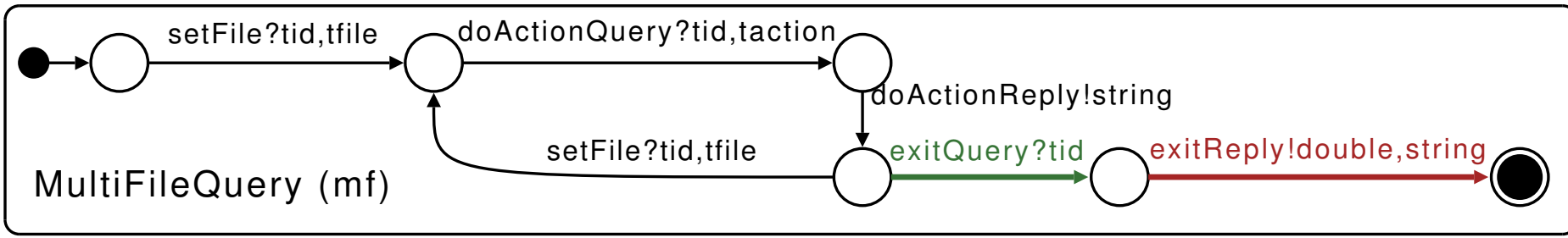


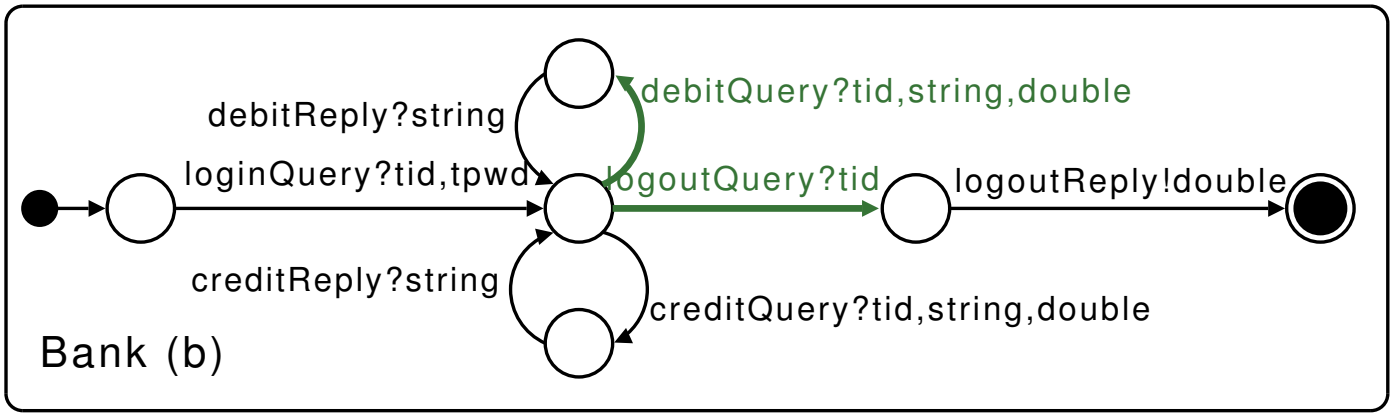




Example (Service Conversations)







Adaptation Contract

extension with value-passing of

Canal et al., *Model-based Adaptation of Behavioural Mismatching Components*,
IEEE TSE, 34(4):546–563, 2008

Vector

A vector represents correspondences between events
This includes message parts using a set of variables (placeholders)

vh=$\langle u : halt!ID; mf : exitQuery?ID; b : logoutQuery?ID \rangle$
vh2=$\langle mf : exitReply!PRICE, INFO; b : debitQuery?ID, INFO, PRICE \rangle$

Adaptation Contract

An adaptation contract is an LTS (V, S, I, F, T) where V is a set of vectors

contracts impose constraints on vector ordering (*à la* choreography)
can be computed using similarity measures or via generate-test tool support

Adaptation Contract

extension with value-passing of

Canal et al., *Model-based Adaptation of Behavioural Mismatching Components*,
IEEE TSE, 34(4):546–563, 2008

Vector

A vector represents correspondences between events
This includes message parts using a set of variables (placeholders)

vh=$\langle u : halt!ID; mf : exitQuery?ID; b : logoutQuery?ID \rangle$
vh2=$\langle mf : exitReply!PRICE, INFO; b : debitQuery?ID, INFO, PRICE \rangle$

Adaptation Contract

An adaptation contract is an LTS (V, S, I, F, T) where V is a set of vectors

contracts impose constraints on vector ordering (*à la* choreography)
can be computed using similarity measures or via generate-test tool support

Adaptation Contract

extension with value-passing of

Canal et al., *Model-based Adaptation of Behavioural Mismatching Components*,
IEEE TSE, 34(4):546–563, 2008

Vector

A vector represents correspondences between events
This includes message parts using a set of variables (placeholders)

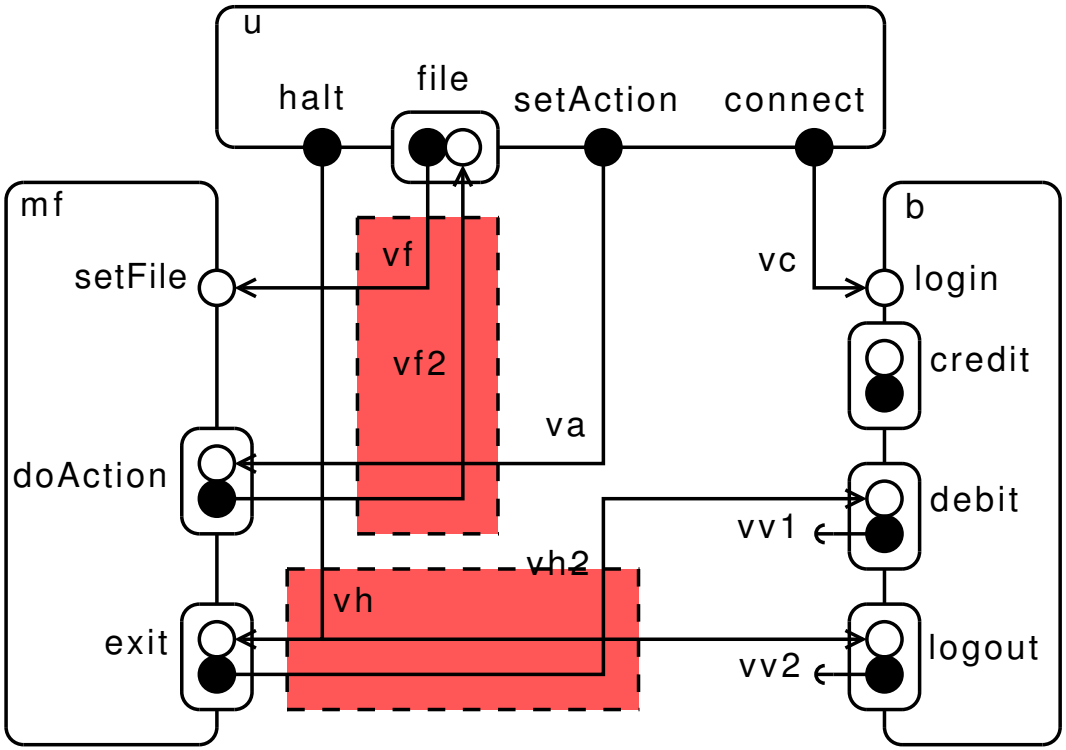
vh=$\langle u : halt!ID; mf : exitQuery?ID; b : logoutQuery?ID \rangle$
vh2=$\langle mf : exitReply!PRICE, INFO; b : debitQuery?ID, INFO, PRICE \rangle$

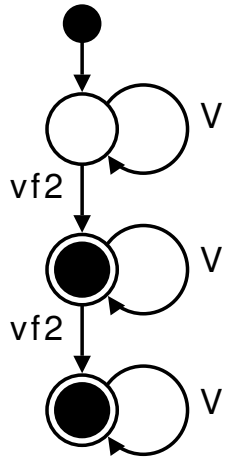
Adaptation Contract

An adaptation contract is an LTS (V, S, I, F, T) where V is a set of vectors

contracts impose constraints on vector ordering (*à la* choreography)
can be computed using similarity measures or via generate-test tool support

Example (Adaptation Contract)





V={vc,va,vf,vh,vh2,vv1,vv2}

vc = <u:connect!ID,PWD ; b:login?ID,PWD >
va = <u:setAction!ID,ACT ; mf:doAction?ID,ACT>
vf = <u:file!ID,FILE ; mf:setFile?ID,FILE>
vf2 = <u:file?RES , mf:doAction!RES>
vh = <u:halt!ID ; mf:exit?ID ; b:logout?ID >
vh2 = <mf:exit!PRICE,INFO ; b:debit?ID,INFO,PRICE>
vv1 = <b:debit!STATUS>
vv2 = <b:logoutReply!BALANCE>

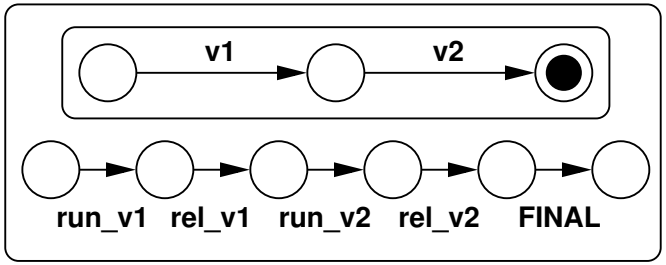
Towards an Implicit Adaptor Model

Idea

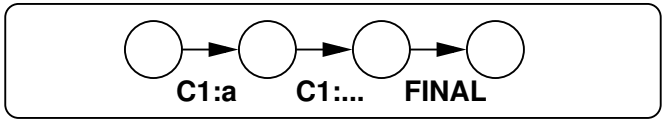
Adaptation constraints encoded as synchronized automaton:

- adaptor in-the-middle (orchestration)
- vector ordering imposed by contract
- in/out message ordering imposed by conversations
- receptions before emissions in vector application
- placeholders already received in a store

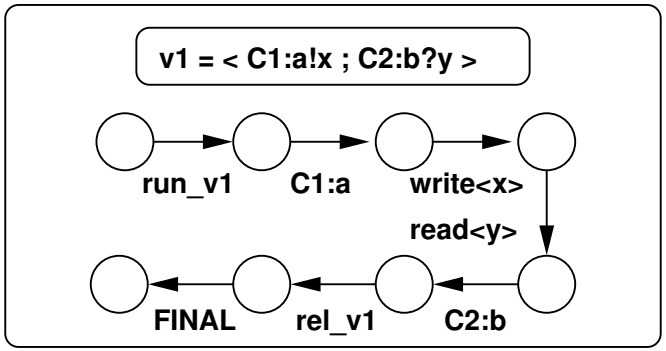
CONTRACT = 1 Automaton



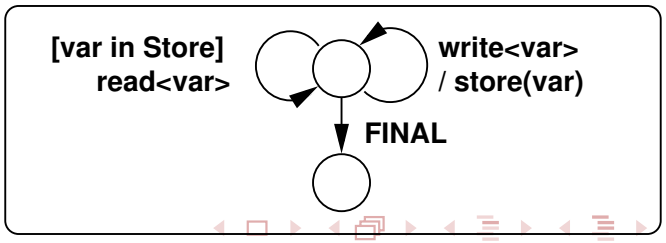
SERVICES = n Automata



VECTORS = m Automata

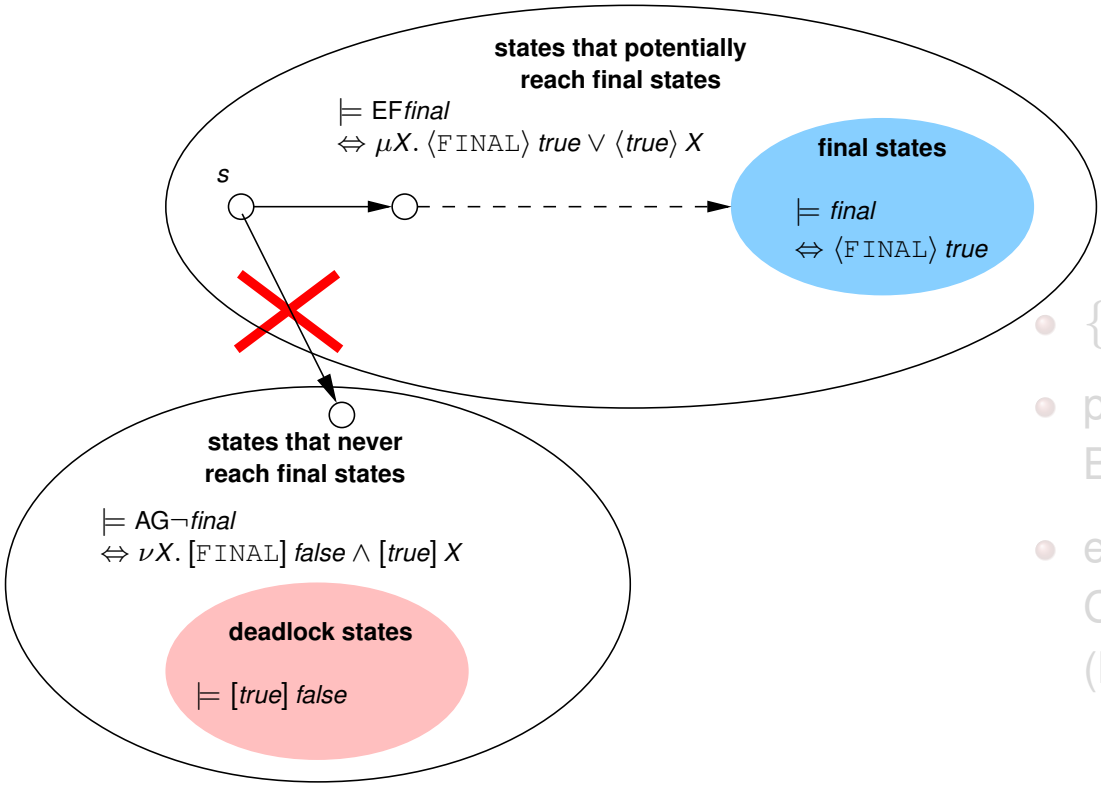


STORE = 1 Automaton



On-the-Fly Adaptor Generation

- forward exploration of the implicit adaptor model
- on-the-fly detection of states potentially reaching successful termination

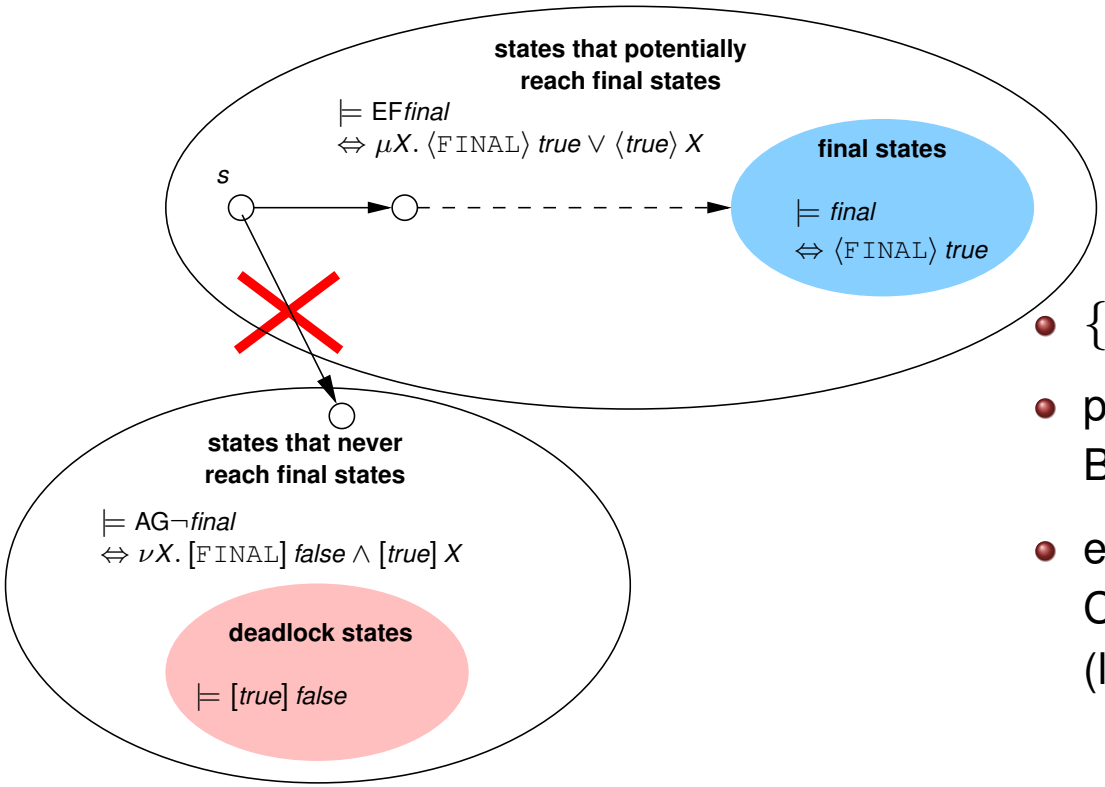


- $\{X_s =_\mu \bigvee_{s \xrightarrow{FINAL} s'} true \vee \bigvee_{s \to s''} X_{s''}\}$
- problem encoding in terms of a Boolean Equation System (BES)
- efficient BES resolution using the Caesar_Solve library (linear wrt. LTS size)

- reduction also performed on-the-fly at the same time wrt. τ-confluence, $\tau^*.a$, weak-trace equivalences

On-the-Fly Adaptor Generation

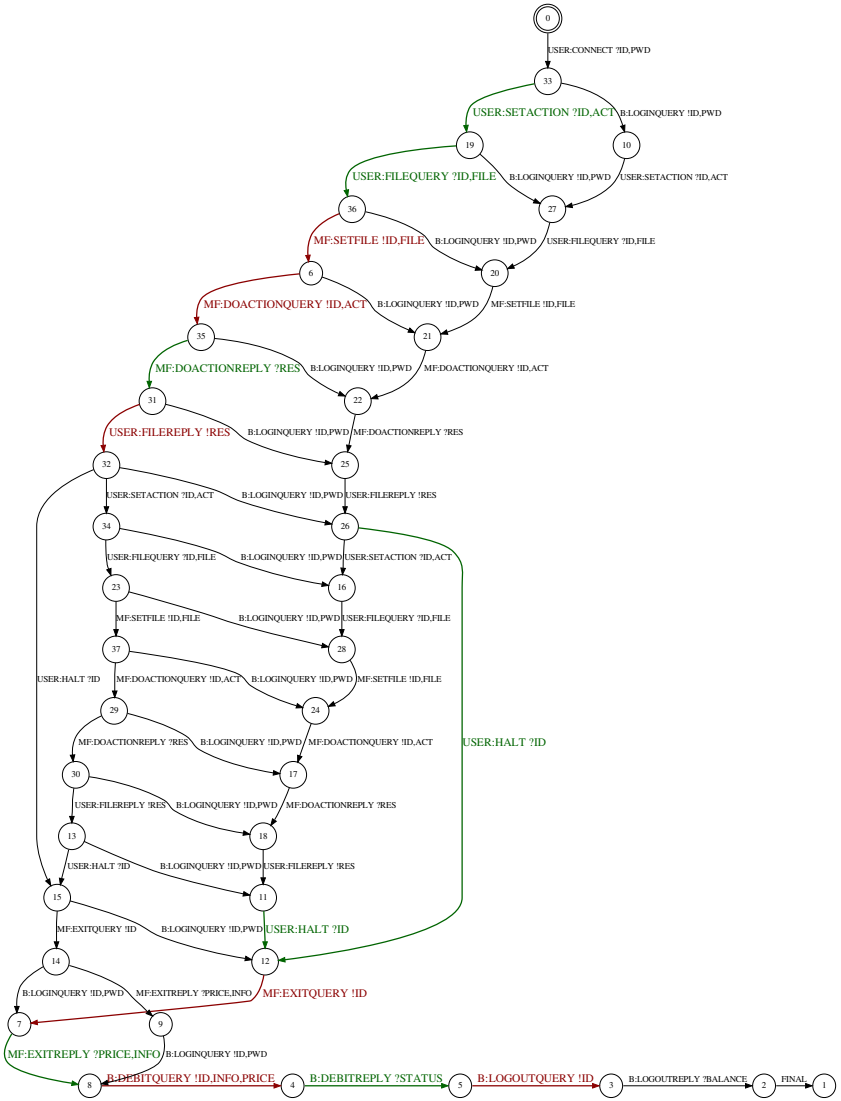
- forward exploration of the implicit adaptor model
- on-the-fly detection of states potentially reaching successful termination



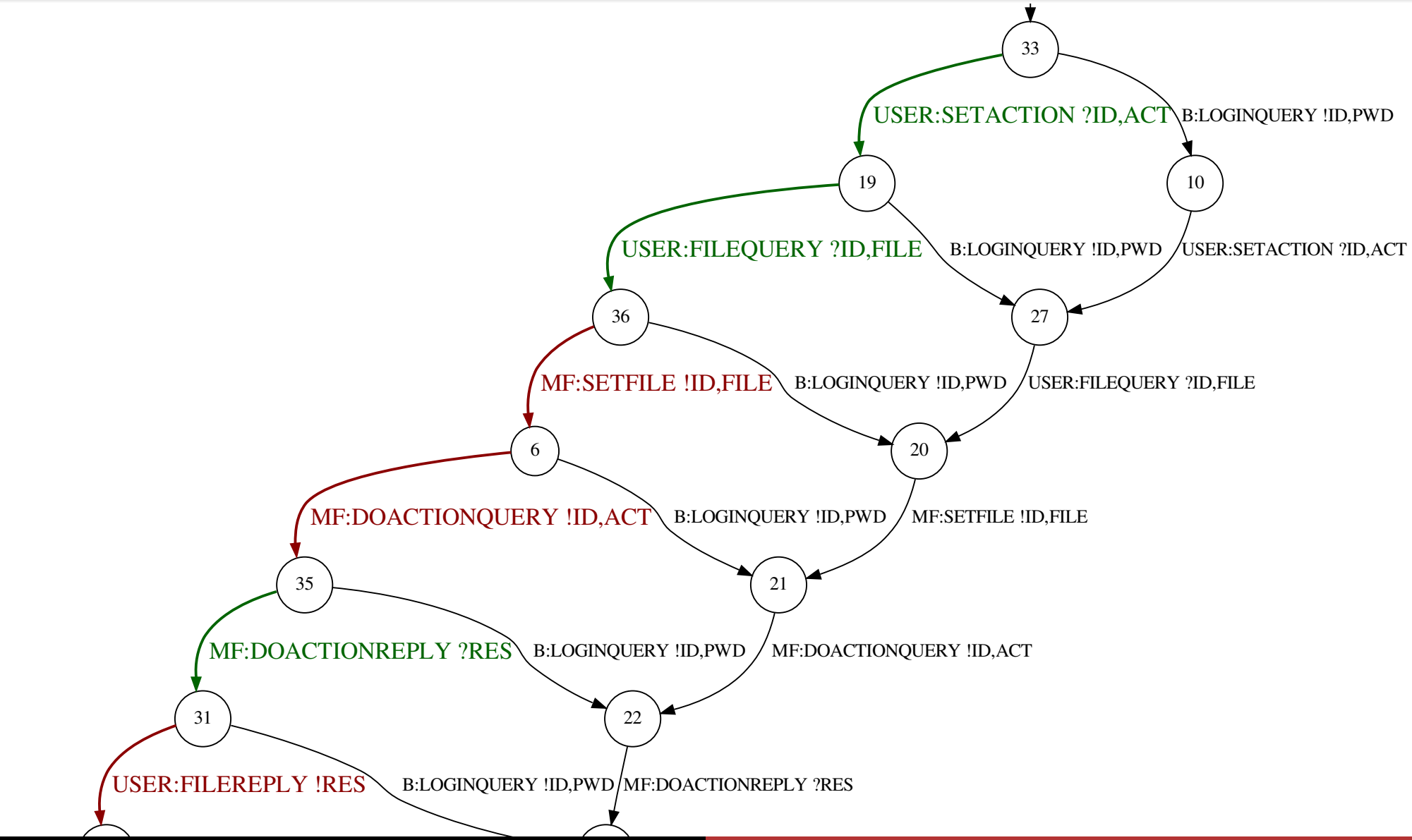
- $\{X_s =_\mu \bigvee_{s \xrightarrow{FINAL} s'} true \vee \bigvee_{s \rightarrow s''} X_{s''}\}$
- problem encoding in terms of a Boolean Equation System (BES)
- efficient BES resolution using the Caesar_Solve library (linear wrt. LTS size)

- reduction also performed on-the-fly at the same time wrt. τ-confluence, $\tau^*.a$, weak-trace equivalences

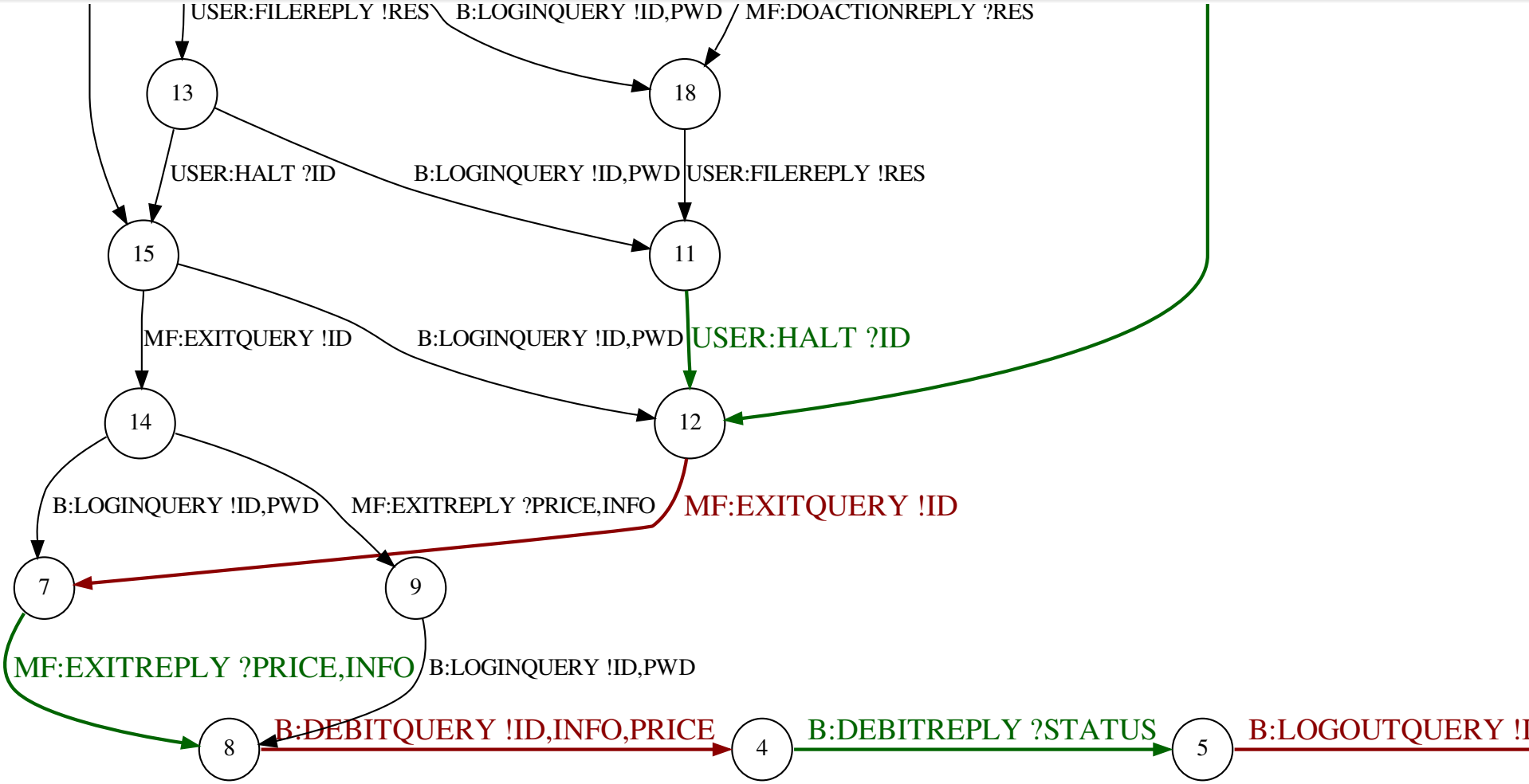
Example (Adaptor Model)



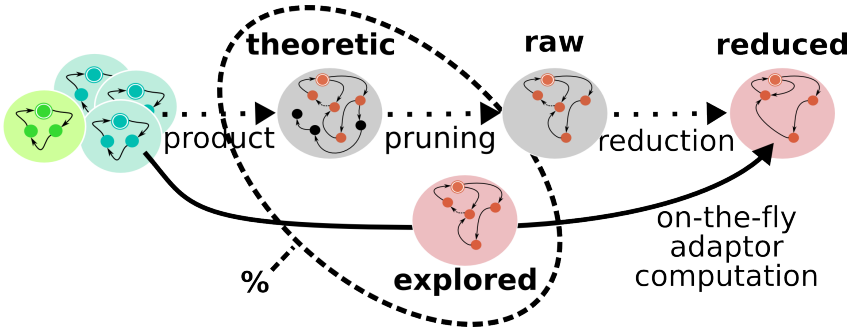
Example (Adaptor Model)



Example (Adaptor Model)



Experiments



Application	Adaptor LTS				State space portion explored for reduced adaptor generation			
	raw		reduced					
	states	trans.	states	trans.	states	%	trans.	%
eMuseum	21418	48692	978	2382	29026	72.8	17075	18.7
music-system	1720	4368	49	60	14805	85.9	32923	74.5
sql-server	1720	4264	22	26	2337	57.1	3427	32.9
multi-file query	1,542	3,709	61	79	6,269	99.95	11,623	69.76
mail-system	418	1059	418	1059	13630	99.7	23946	70.1
pc-store	253	472	16	16	782	88.2	1208	66.8
rate-service	241	483	28	32	400	52.6	675	37.2
video-on-demand	149	231	17	22	251	97.6	260	63.5
batchsql	137	239	31	43	429	67.1	276	21.6
restau-booking	94	108	33	37	264	99.6	280	83.1
pc-store	17	17	17	17	237	91.5	249	64.3

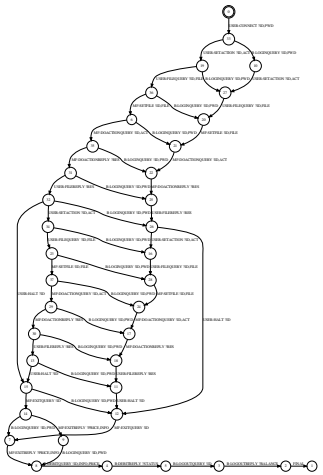
largest example computation: $\sim$1 minute (731 MHz / 512 MB PC Linux)

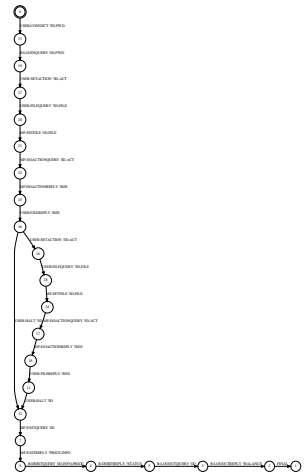
Principles

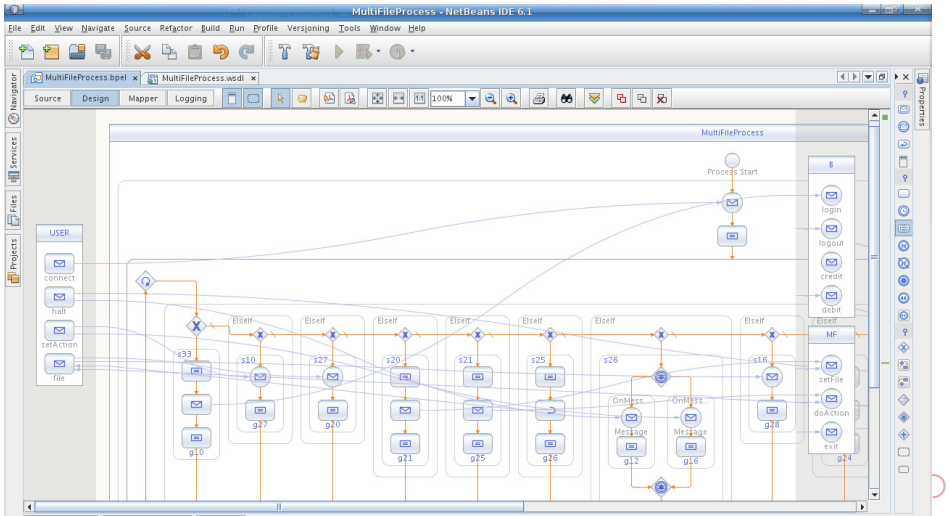
The previous steps followed a MDE (PIM/PSM) approach

Implementation is then achieved in two steps:

1. filtering the adaptor model to ensure its implementability
2. encoding the filtered model (state-machine pattern)







Filtering

Given an adaptor model (A, S, I, F, T) for a set of services S_i, three rules

R1 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s'$ then remove all transitions $s \longrightarrow_{a_i?\ldots} s'_i$

R2 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s_1$ and $s \longrightarrow_{a_2!\ldots} s_2$ with $a_1 \neq a_2$ or $s_1 \neq s_2$ then remove $s \longrightarrow_{a_2!\ldots} s_2$

R3 for every two-way operation $o[m_1, m_2]$ of some S_i, for every $s \longrightarrow_{!S_i:m_1} s'$, remove all transitions outgoing from s' but for $s' \longrightarrow_{?S_i:m_2} s''$

Cleaning the model up

C1 remove any state $s \in S$ such that there does not exist a path $I \longrightarrow^* s$
accordingly remove any transition outgoing from s

C2 remove any state $s \in S$ such that there does not exist a path $s \longrightarrow^* f$, with $f \in F$
accordingly remove any transition going to s

Filtering

Given an adaptor model (A, S, I, F, T) for a set of services S_i, three rules

R1 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s'$ then remove all transitions $s \longrightarrow_{a_i?\ldots} s'_i$

R2 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s_1$ and $s \longrightarrow_{a_2!\ldots} s_2$ with $a_1 \neq a_2$ or $s_1 \neq s_2$ then remove $s \longrightarrow_{a_2!\ldots} s_2$

R3 for every two-way operation $o[m_1, m_2]$ of some S_i, for every $s \longrightarrow_{!S_i:m_1} s'$, remove all transitions outgoing from s' but for $s' \longrightarrow_{?S_i:m_2} s''$

Cleaning the model up

C1 remove any state $s \in S$ such that there does not exist a path $I \longrightarrow^* s$
accordingly remove any transition outgoing from s

C2 remove any state $s \in S$ such that there does not exist a path $s \longrightarrow^* f$, with $f \in F$
accordingly remove any transition going to s

Filtering

Given an adaptor model (A, S, I, F, T) for a set of services S_i, three rules

- R1 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s'$ then remove all transitions $s \longrightarrow_{a_i?\ldots} s'_i$
- R2 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s_1$ and $s \longrightarrow_{a_2!\ldots} s_2$ with $a_1 \neq a_2$ or $s_1 \neq s_2$ then remove $s \longrightarrow_{a_2!\ldots} s_2$
- R3 for every two-way operation $o[m_1, m_2]$ of some S_i, for every $s \longrightarrow_{!S_i:m_1} s'$, remove all transitions outgoing from s' but for $s' \longrightarrow_{?S_i:m_2} s''$

Cleaning the model up

- C1 remove any state $s \in S$ such that there does not exist a path $I \longrightarrow^* s$
  accordingly remove any transition outgoing from s
- C2 remove any state $s \in S$ such that there does not exist a path $s \longrightarrow^* f$, with $f \in F$
  accordingly remove any transition going to s

Filtering

Given an adaptor model (A, S, I, F, T) for a set of services S_i, three rules

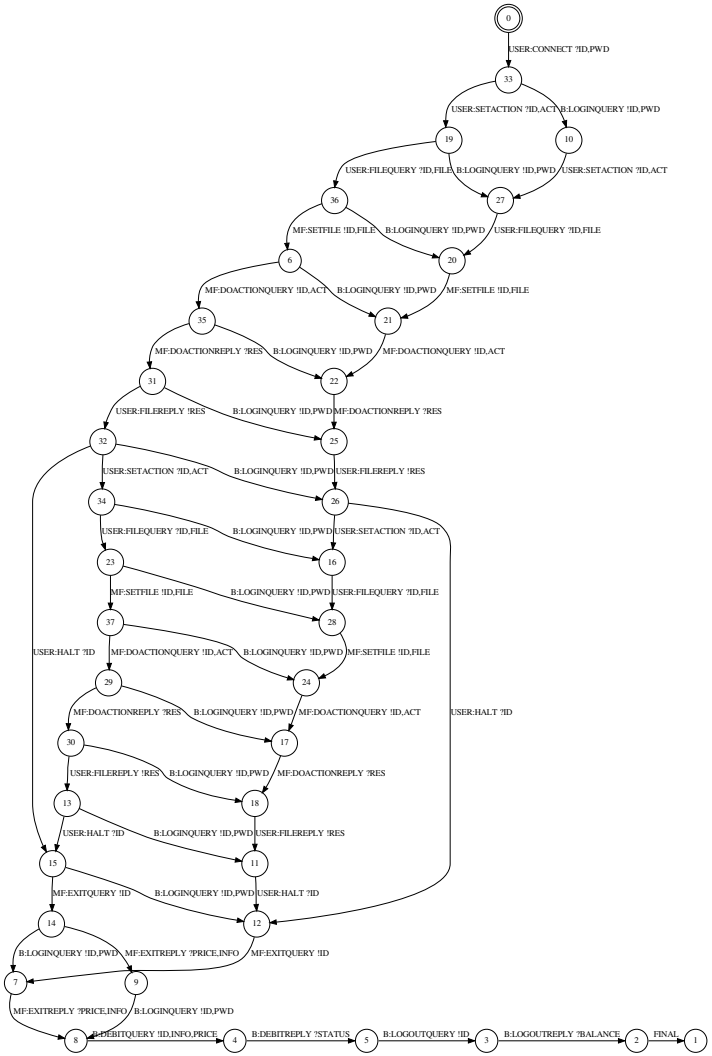
- R1 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s'$ then remove all transitions $s \longrightarrow_{a_i?\ldots} s'_i$
- R2 for every $s \in S$, if $s \longrightarrow_{a_1!\ldots} s_1$ and $s \longrightarrow_{a_2!\ldots} s_2$ with $a_1 \neq a_2$ or $s_1 \neq s_2$ then remove $s \longrightarrow_{a_2!\ldots} s_2$
- R3 for every two-way operation $o[m_1, m_2]$ of some S_i, for every $s \longrightarrow_{!S_i:m_1} s'$, remove all transitions outgoing from s' but for $s' \longrightarrow_{?S_i:m_2} s''$

Cleaning the model up

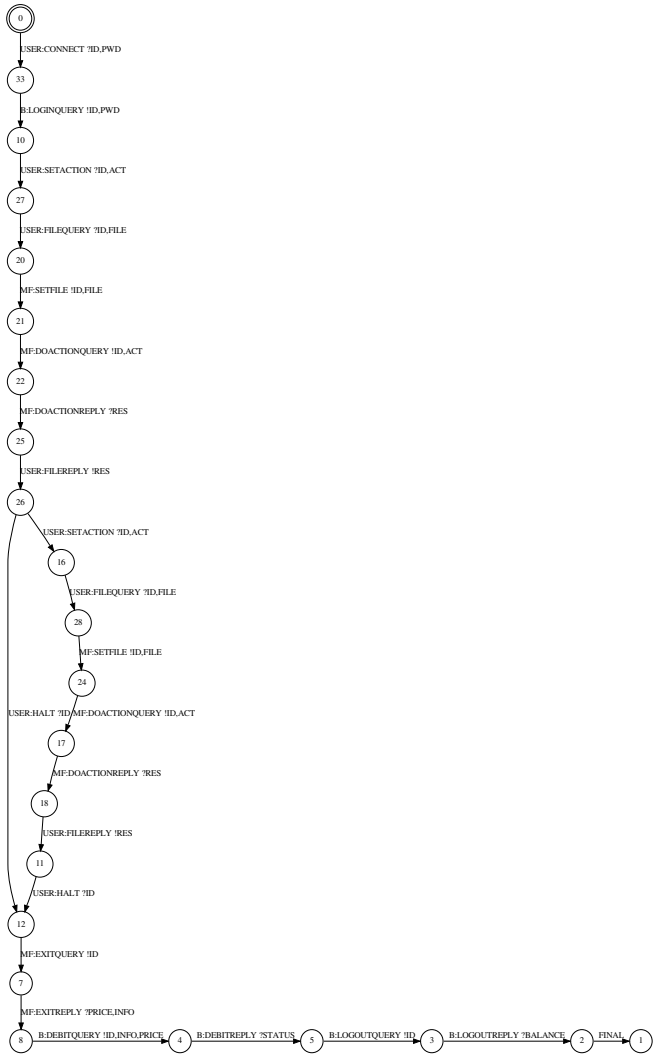
- C1 remove any state $s \in S$ such that there does not exist a path $I \longrightarrow^* s$ accordingly remove any transition outgoing from s
- C2 remove any state $s \in S$ such that there does not exist a path $s \longrightarrow^* f$, with $f \in F$
  accordingly remove any transition going to s

Example (Filtered Adaptor Model)

before filtering



after filtering



Encoding (1/2)

Given an adaptor model (A, S, I, F, T) for a set of services S_i,

partner links	one per service (S_i) + USER
variables	message parts (mp-var) vectors var. (v-var) STATE, FINAL
process	encoded using the state machine pattern • initially STATE=s, where $I \longrightarrow_{USER:m?} S$, and FINAL=false • while(not FINAL) loop • for final states (F): FINAL=true • for all states: communication

Encoding (2/2)

Given an adaptor model (A, S, I, F, T) for a set of services S_i,

invoke $s \longrightarrow_{S_i:m_1!x_1,\ldots,x_n} s'$, $s' \longrightarrow_{S_i:m_2?y_1,\ldots,y_m} s''$, $o[m_1, m_2] \in S_i$:
invoke(S_i,o,$< x_1, \ldots, x_n >$,$< y_1, \ldots, y_m >$)

receive $s \longrightarrow_{USER:m_1?x_1,\ldots,x_n} s'$, $o[m_1, m_2] \in USER$:
receive(USER,o,$< x_1, \ldots, x_n >$) *if no other reception*
pick + onMessage(USER,o,$< x_1, \ldots, x_n >$) *else*
+ onAlarm if $s \in F$

reply $s \longrightarrow_{USER:m_2!x_1,\ldots,x_n} s'$, $o[m_1, m_2] \in USER$:
reply(USER,o,$< x_1, \ldots, x_n >$)

assign. before invoke/reply: v-var $\longrightarrow$ mp-var
after receive: mp-var $\longrightarrow$ v-var

Concluding Remarks

Results

- design/deployment time service orchestration
  with restrictive+generative adaptation
- adaptation is computed efficiently (on-the-fly): scrutator tool
- distribution of adaptors is possible

  following, *e.g.*, [Autili *et al.*, JSS 81(12), 2008], [Salaün, SEFM'08]

Perspectives

efficiency	new reductions in scrutator, preorders
expressiveness	adding formal semantics to operations
user support	adaptation tool-box with help for contract specification

Concluding Remarks

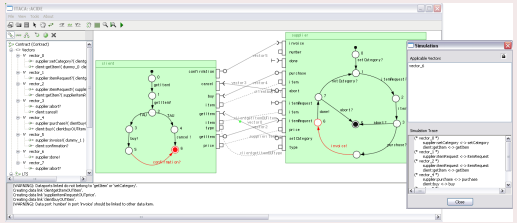
Results

- design/deployment time service orchestration
  with restrictive+generative adaptation
- adaptation is computed efficiently (on-the-fly): scrutator tool
- distribution of adaptors is possible

  following, *e.g.*, [Autili *et al.*, JSS 81(12), 2008], [Salaün, SEFM'08]

Perspectives

efficiency	new reductions in scrutator, preorders
expressiveness	adding formal semantics to operations
user support	adaptation tool-box with help for contract specification



thank you for your attention

Verification

The adapted orchestration O is deadlock-free and livelock-free by construction.
Still handcrafted contracts and pruning makes verification important.

Currently one can check:

- Application-independent properties
  (no human intervention)
  - *syntactic contract checking*
  - *occurence of action/variables (in O)*
- Application-specific properties
  (requires one gives the properties)
  - *safety properties*: something bad never happens (in O)
  - *liveness properties*: something good eventually happens (in O)

all are achieved on-the-fly using CADP

Verification

The adapted orchestration O is deadlock-free and livelock-free by construction.
Still handcrafted contracts and pruning makes verification important.

Currently one can check:

- Application-independent properties
  (no human intervention)
  - *syntactic contract checking*
  - *occurence of action/variables (in O)*
- Application-specific properties
  (requires one gives the properties)
  - *safety properties*: something bad never happens (in O)
  - *liveness properties*: something good eventually happens (in O)

all are achieved on-the-fly using CADP

Verification

The adapted orchestration O is deadlock-free and livelock-free by construction.
Still handcrafted contracts and pruning makes verification important.

Currently one can check:

- Application-independent properties
  (no human intervention)
  - *syntactic contract checking*
  - *occurence of action/variables (in O)*
- Application-specific properties
  (requires one gives the properties)
  - *safety properties*: something bad never happens (in O)
  - *liveness properties*: something good eventually happens (in O)

all are achieved on-the-fly using CADP