

Service Composition and Synthesis

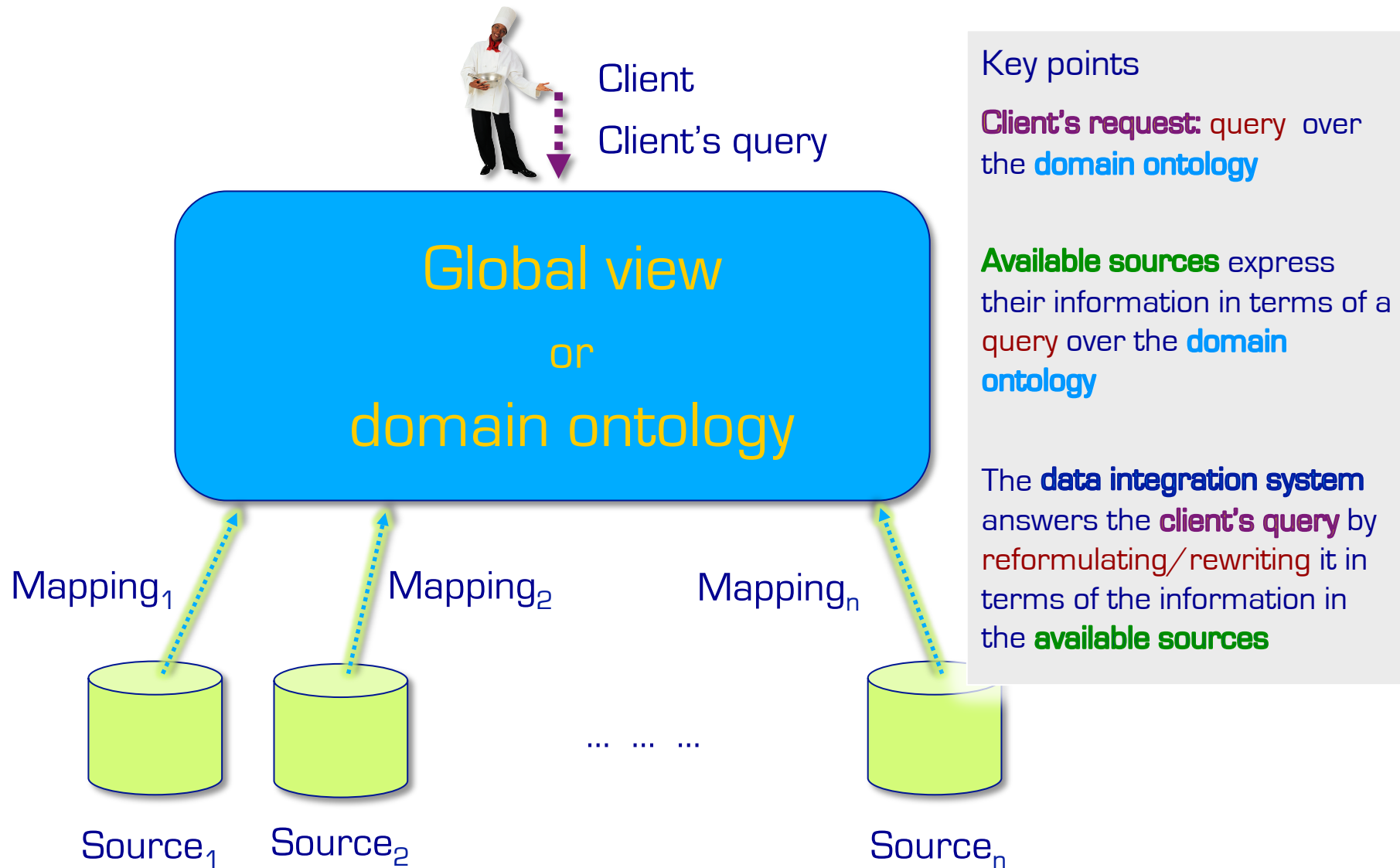
The Roman Model (including nondeterministic services)

Giuseppe De Giacomo
SAPIENZA Università di Roma, Italy

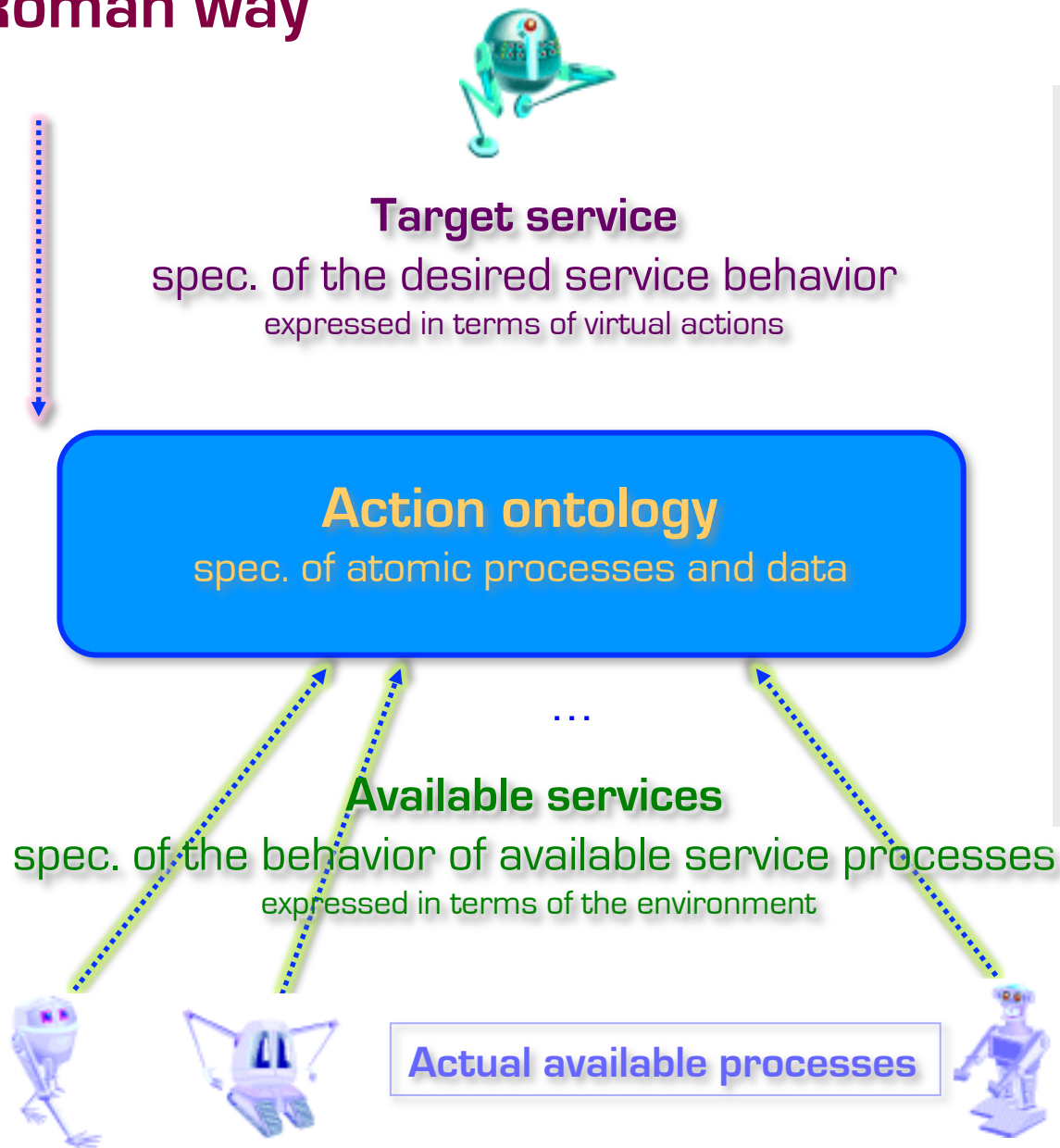
Introduction

- The promise of **Service Computing** is to use services fundamental elements for realizing distributed applications/solutions.
- **Services** are processes that export their **abstract specification**
- When no available service satisfies a desired specification, one might check whether (parts of) available services can be **composed** and **orchestrated** in order to realize the specification.
- **Working at an abstract level** enable us to exploit results from **automatic verification and synthesis** to verify and compose services.
- The problem of automatic composition becomes especially interesting in the presence of **stateful** (conversational) services.
- Among the various frameworks proposed in the literature, here we concentrate on the so called ``**Roman Model**'' *[name by Rick Hull]*.

Data Integration



Service integration/composition: The Roman way



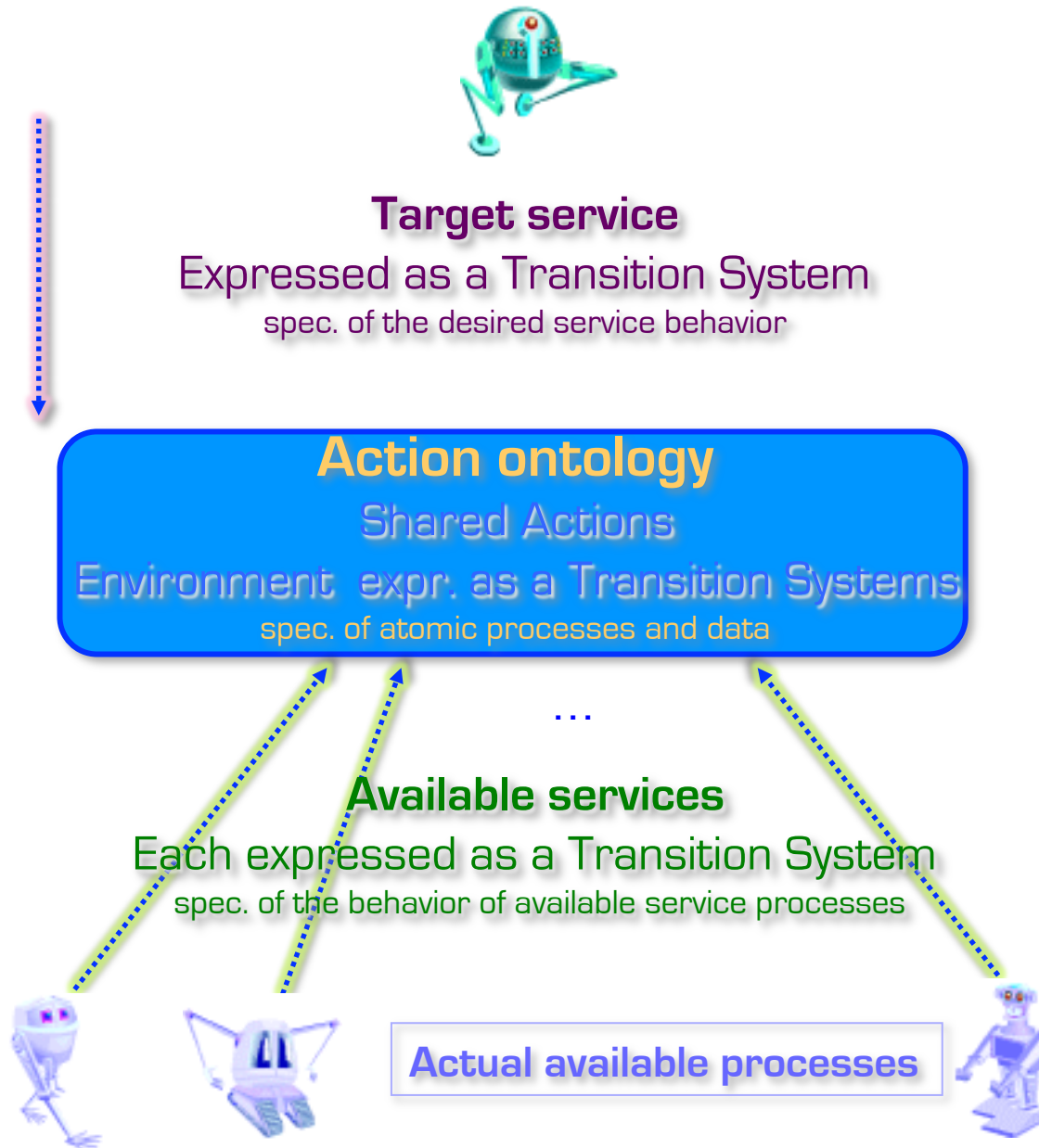
Key points

No available process for the target service

Must realize **target service** by **delegating** actual actions to **available services**

Available services are **stateful**, hence must **realize** the **target** using **fragments** of their **computations**

The Roman Model: *basics*



Key points

No available process for the target service

Must realize **target service** by **delegating** actual actions to **available services**

Available services are **stateful**, hence must **realize** the **target** using **fragments** of their **computations**

Roman Model's main ingredients

- The Roman Model exemplifies what can be achieved by composing conversational services and uncovers relationships with automated synthesis of reactive processes in Verification and AI Planning.
- Roman Model's main ingredients
 - **Each available service** is formally specified as a **transition system** that captures its possible conversations with a generic client.
 - Desired specification is a **target service**, described itself as a **transition system**.
 - the aim is to **automatically synthesize orchestrators** that realize the target service by delegating its actions to the available services, exploiting fragments of their execution.

Transition systems

- We represent services as **transition systems**:
- A TS is a tuple $\langle A, S, s_0, \delta, F \rangle$ where:
 - A is the set *shared* of actions
 - S is the set of states
 - $s_0 \in S$ is the set of initial states
 - $\delta \subseteq S \times A \times S$ is the transition relation
 - $F \subseteq S$ is the set of final states

Service composition

Problem of composition existence

- Given:
 - available services B_1, \dots, B_n
 - target service Tover the same environment (same set of atomic actions)
- Check whether T can be realized by **delegating** actions to B_1, \dots, B_n so as to **mimic** T over time *(forever!)*

Composition synthesis

*synthesis of the **orchestrator** that does the delegation*

Service composition as a game

There are at least two kinds of games. One could be called finite, the other infinite.

*A finite game is played for the purpose of winning ...
... an infinite game for the purpose of continuing the play.*

Finite and Infinite Games
J. P. Carse, *philosopher*

Service composition as a game:

Service composition vs Planning

Planning

Stateless service composition

- **Operators:** atomic actions
- **Goal:** desired state of affair
- **Game: *finite!***
 - compose operators sequentially so as to reach the goal
- **Playing strategy:** plan
(program having operators invocation as atomic instructions)

Roman model

Service composition

- **Operators:** available transition systems
- **Goal:** target transition system
- **Game: *infinite!***
 - compose available transition systems concurrently so as to play the target transition system
- **Playing strategy:** orchestrator
(process that delegate target actions to the available service)

Nondeterminism in Available Services

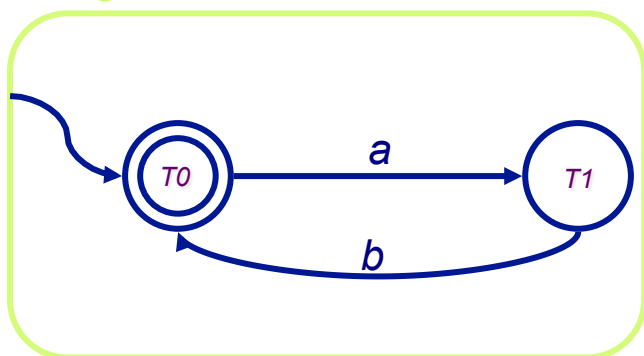
Devilish (don't know)!

- Nondeterministic available services
 - **Incomplete information** on the actual **behavior**
 - **Mismatch between behavior description** (which is in terms of the environment actions) and **actual behavior** of the agents/devices
- Deterministic target service
 - it's a spec of a desired service: [devilish] nondeterminism is banned

*In general, devilish nondeterminism difficult to cope with
eg. nondeterminism moves AI Planning from PSPACE (classical planning) to EXPTIME (contingent planning with full observability [Rintanen04])*

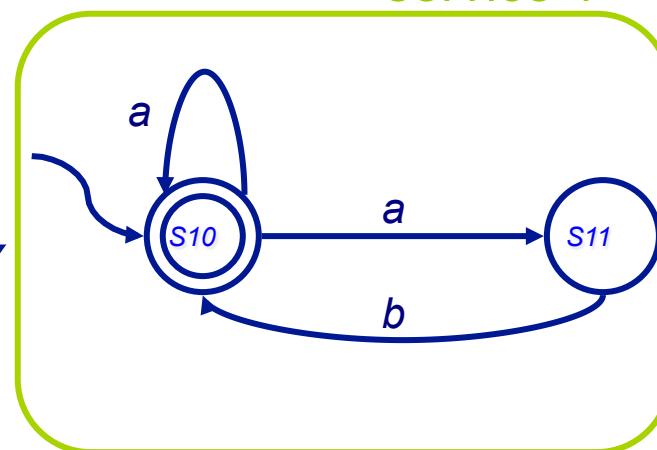
Simple example of service composition

target service

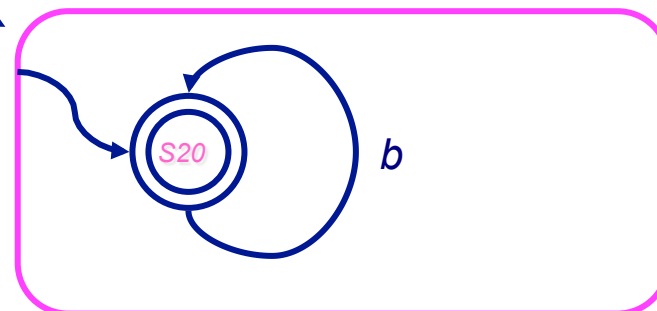


Devilish nondeterminism!

service 1



service 2

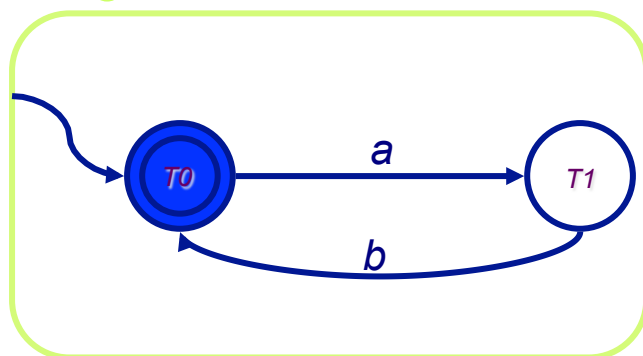


orchestrator

For simplicity we don't consider environment.

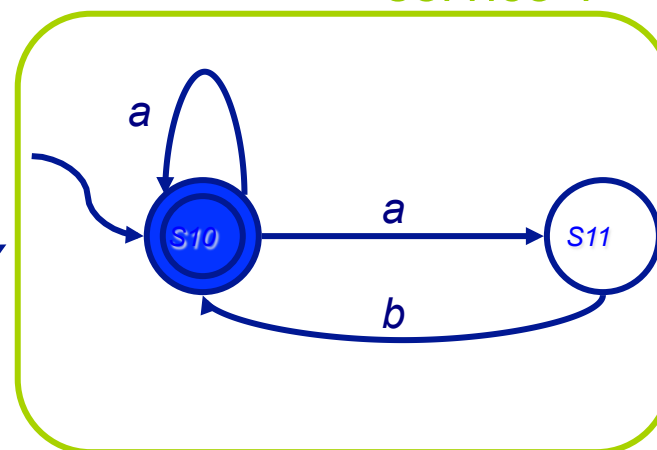
Simple example of service composition

target service

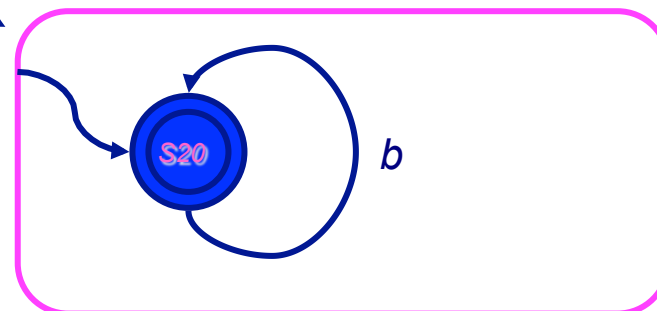


orchestrator

service 1

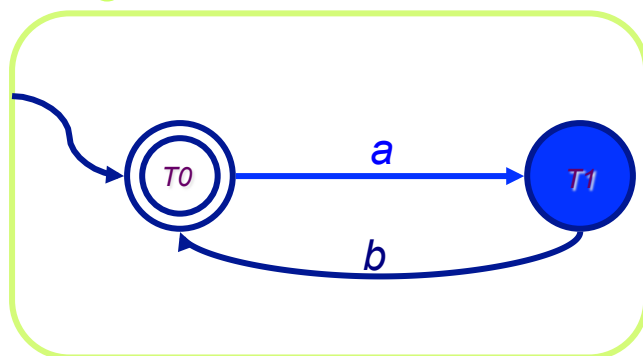


service 2



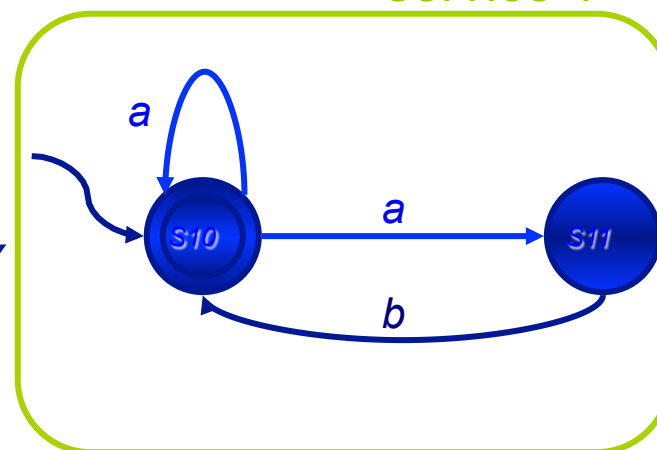
Simple example of service composition

target service

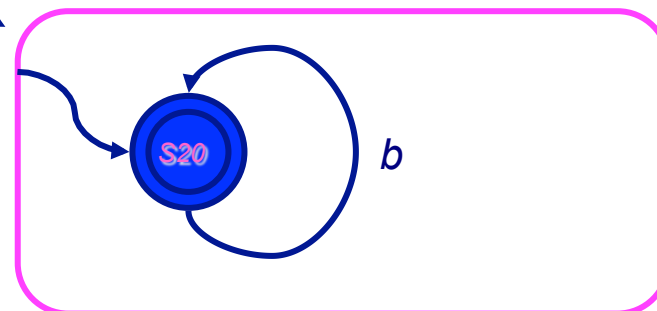


orchestrator

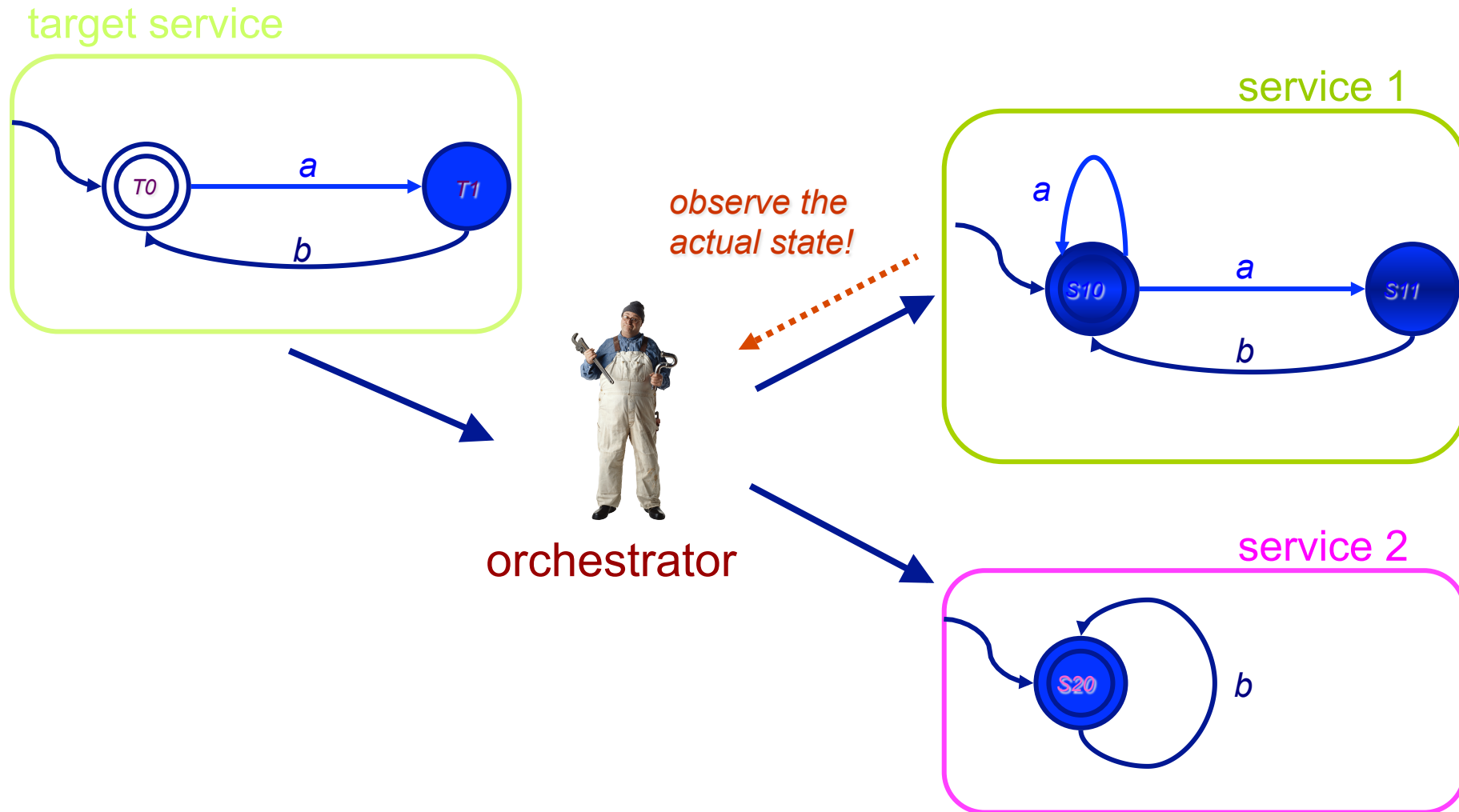
service 1



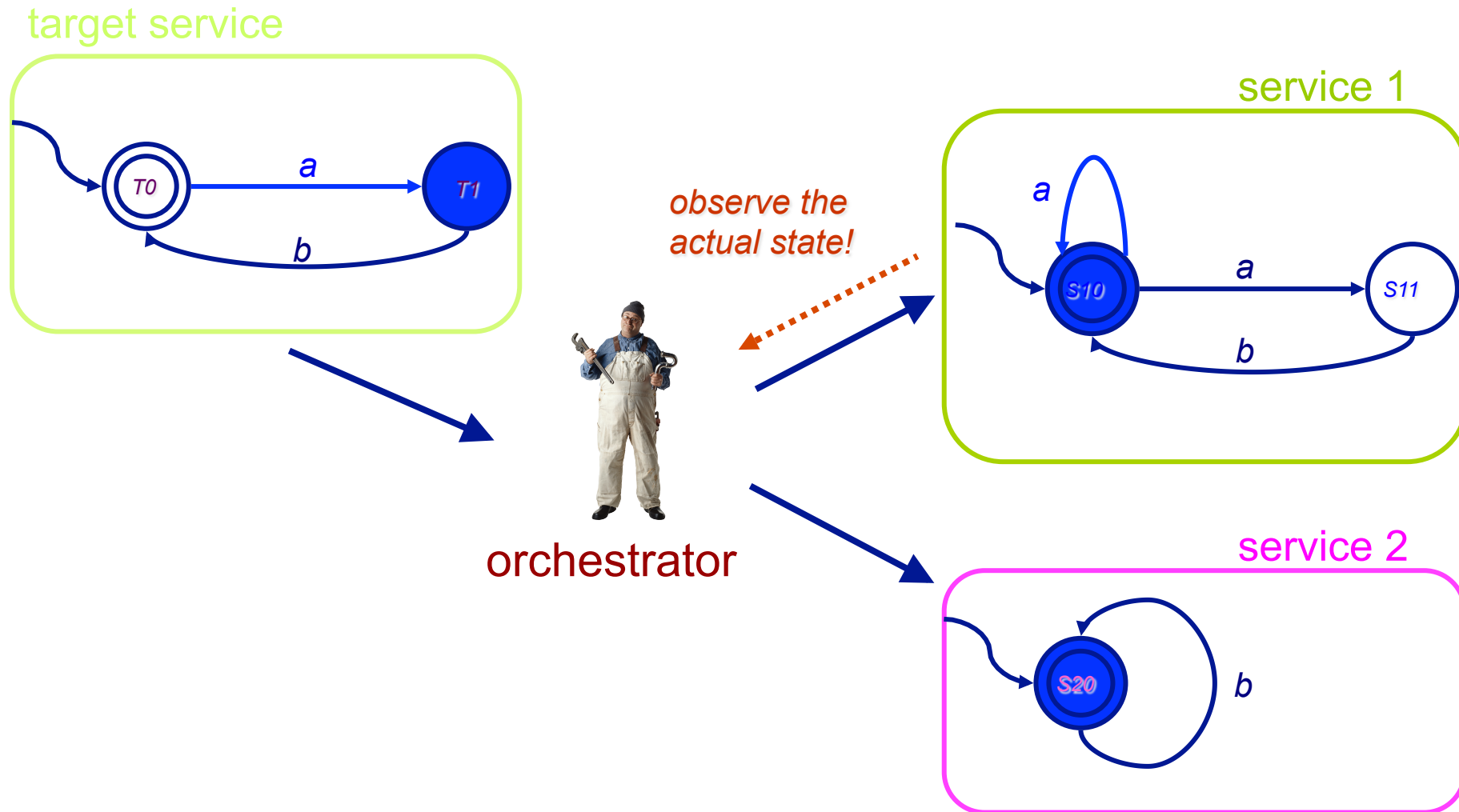
service 2



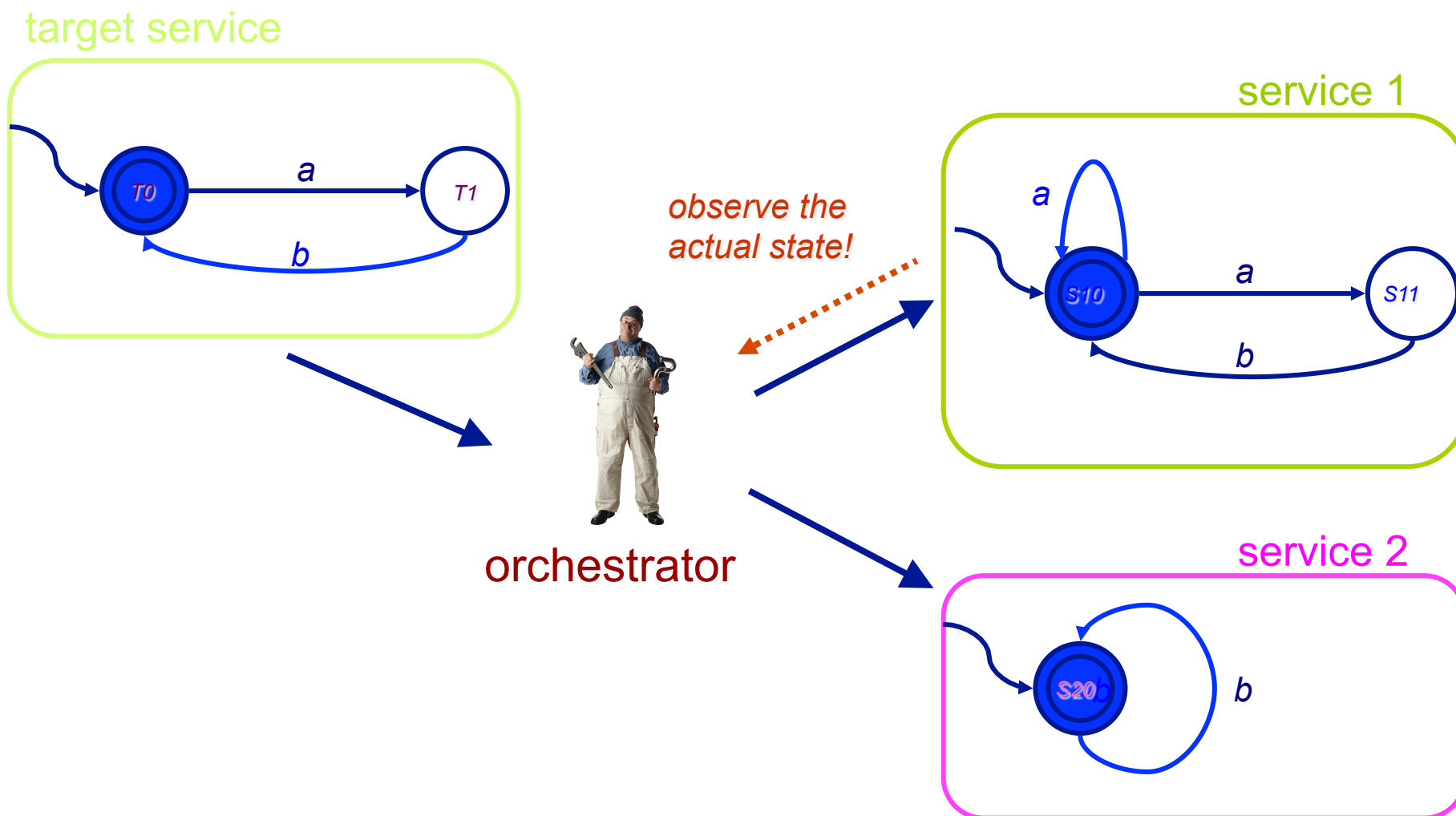
Simple example of service composition



Simple example of service composition

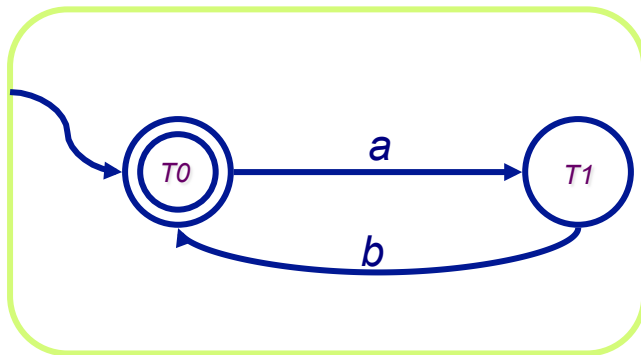


Simple example of service composition

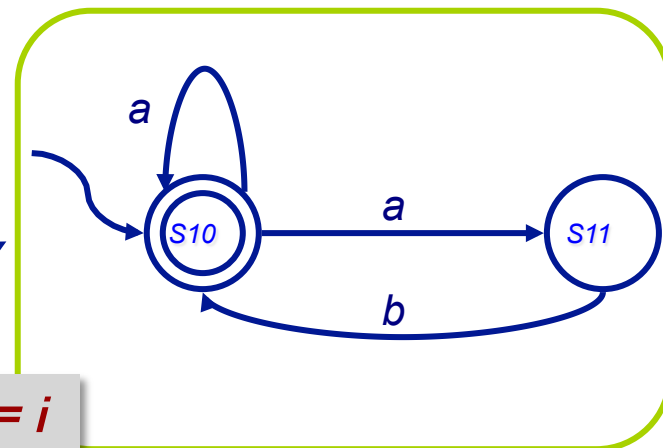


Simple example of service composition

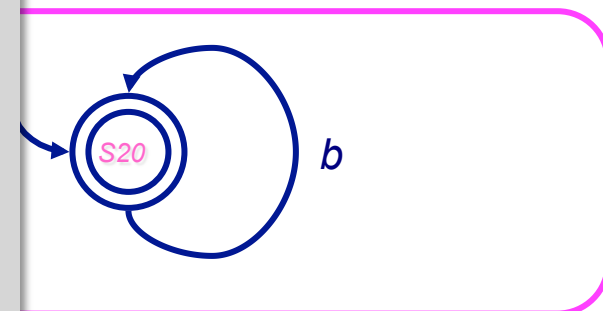
target service



service 1




service 2



- **Orchestrator program** is *any function* $P(h,a) = i$ that takes a **history** h and an **action** a to execute and **delegates** a to one of the available services i
- A **history** is a sequence that alternates states of the available services with actions performed:

$$[s_1^0, s_2^0, \dots, s_n^0] a_1 [s_1^1, s_2^1, \dots, s_n^1] \dots a_k [s_k^1, s_2^k, \dots, s_n^k]$$
- Observe that to take a decision P has **full access to the past**, but no access to the future

Synthesizing compositions

- Techniques for computing compositions:
- Reduction to PDL SAT
- Simulation-based 
- LTL synthesis as model checking of game structure

(all techniques are for finite state services)

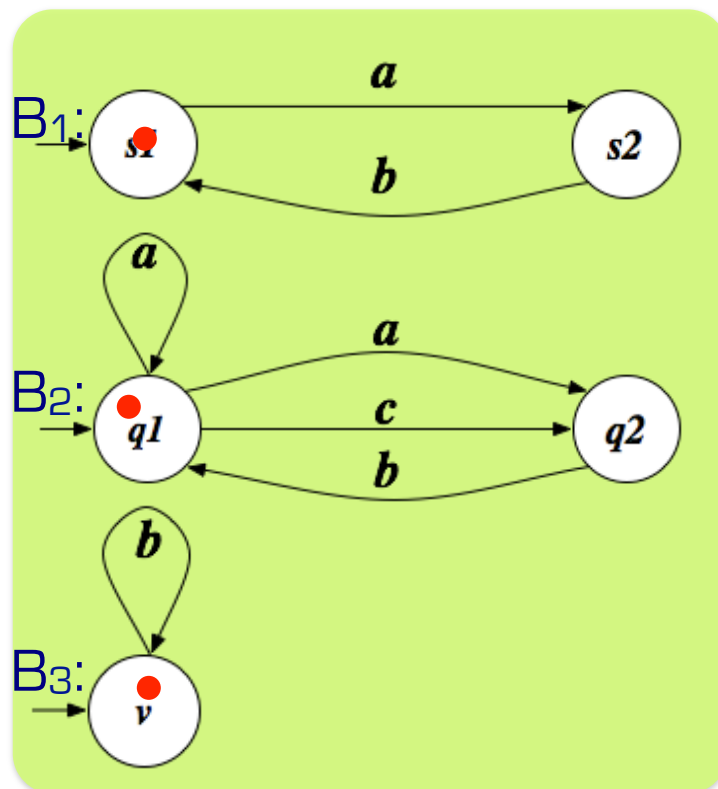
Simulation-based technique

Directly based on

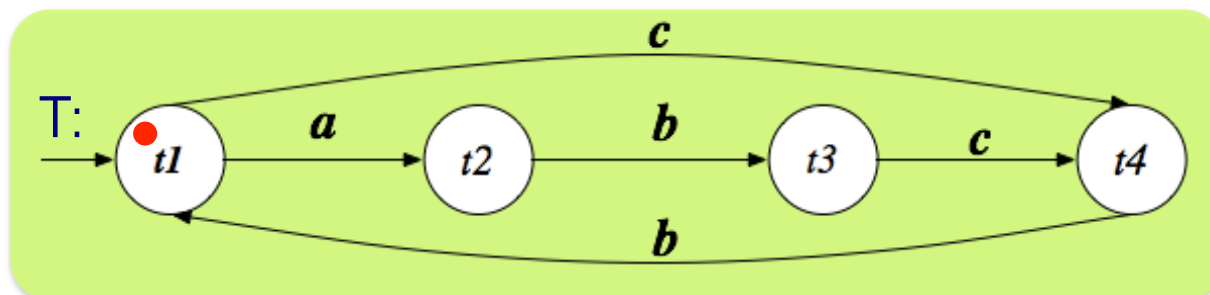
*... controlling the concurrent execution of available services B_1, \dots, B_n so as to **mimic** the target service T*

Thm: *Composition exists iff the asynchronous (Cartesian) product C of B_1, \dots, B_n can **(ND-)simulate** T*

Example of composition by simulation



Given from available and target service ...



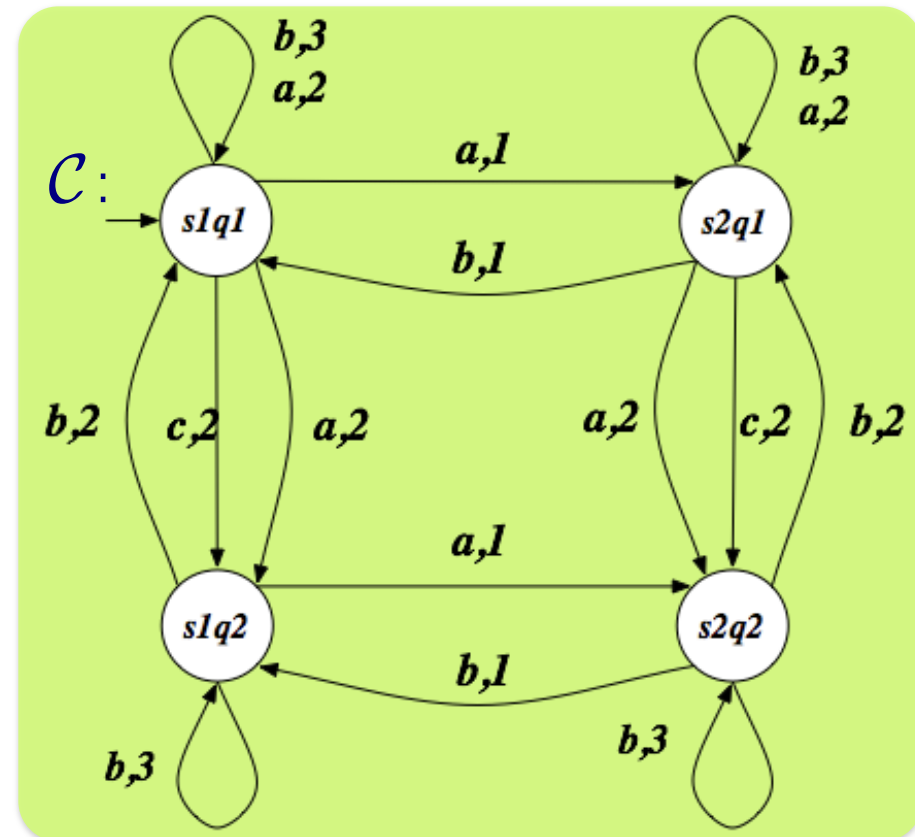
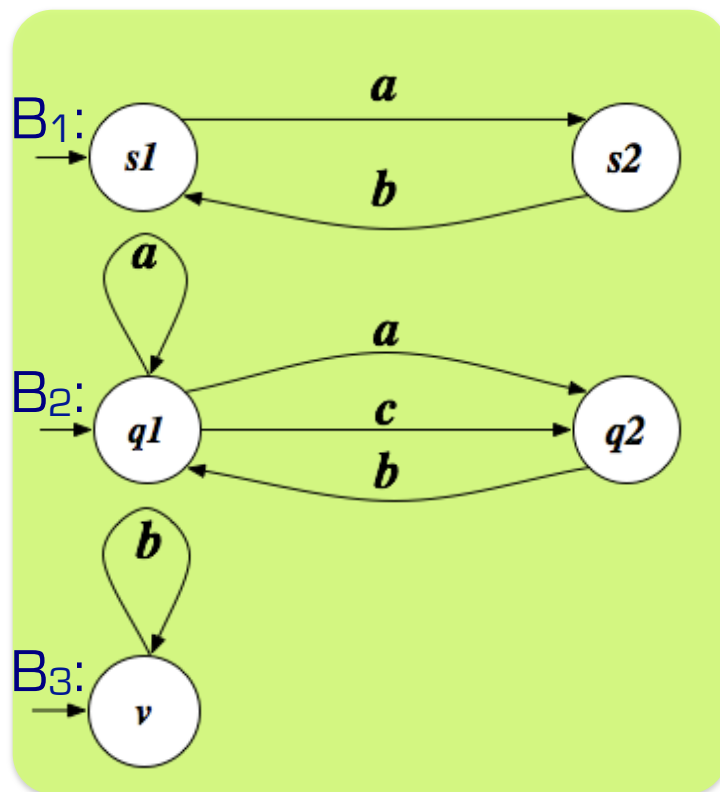
Computing composition via simulation

Let B_1, \dots, B_n be the TSs of the available services.

The **available services TS** $\mathcal{C} = \langle A, S_{\mathcal{C}}, s_{\mathcal{C}}^0, \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle$ is the **asynchronous product** of B_1, \dots, B_n where:

- A is the set of actions
- $S_{\mathcal{C}} = S_1 \times \dots \times S_n$
- $s_{\mathcal{C}}^0 = (s_1^0, \dots, s_n^0)$
- $F_{\mathcal{C}} = F_1 \times \dots \times F_n$
- $\delta_{\mathcal{C}} \subseteq S_{\mathcal{C}} \times A \times S_{\mathcal{C}}$ is defined as follows:
- $(s_1 \times \dots \times s_n) \rightarrow_a (s'_1 \times \dots \times s'_n)$ iff
$$\exists i. s_i \rightarrow_a s'_i \in \delta_i \text{ and } \forall j \neq i. s'_j = s_j$$

Example of composition by simulation



... consider the **asynchronous product** of the available services ...

Simulation relation

Given a target service T and (the asynchronous product of) available services \mathcal{C} , an **(ND-)simulation** is a relation R between the states $t \in \mathcal{T}$ and (s_1, \dots, s_n) of \mathcal{C} such that:

- $(t, s_1, \dots, s_n) \in R$ implies that
 - if t is *final* then s_i is *final* ($i=1, \dots, n$)
 - for all $t \rightarrow_a t'$ in T , exists a $B_i \in \mathcal{C}$ s.t.
 - $\exists s_i \rightarrow_a s'_i$ in $B_i \wedge$
 - $\forall s_i \rightarrow_a s'_i$ in $B_i \Rightarrow (t', s_1, \dots, s'_i, \dots, s_n) \in R$
- If **exists a simulation** relation R (such that $(t^0, s_1^0, \dots, s_n^0) \in R$, then we say that or **T is simulated by \mathcal{C}** (or **\mathcal{C} simulates T**).
- **Simulated-by** is
 - (i) a simulation;
 - (ii) the largest simulation.

Simulated-by is a coinductive definition

Simulation relation (cont.)

Algorithm Compute (ND-)simulation

Input: target behavior T and (async. prod. of) available services \mathcal{C}

Output: the **simulated-by** relation (the largest simulation)

Body

$R = \emptyset$

$R' = S_T \times S_1 \times \dots \times S_n - \{(t, s_1, \dots, s_n) \mid t \in F_T \wedge s_i \notin F_i \text{ for some } i\}$

while $(R \neq R')$ {

$R := R'$

$R' := R' - \{(t, s_1, \dots, s_n) \mid \exists t \rightarrow_a t' \text{ in } T \wedge$

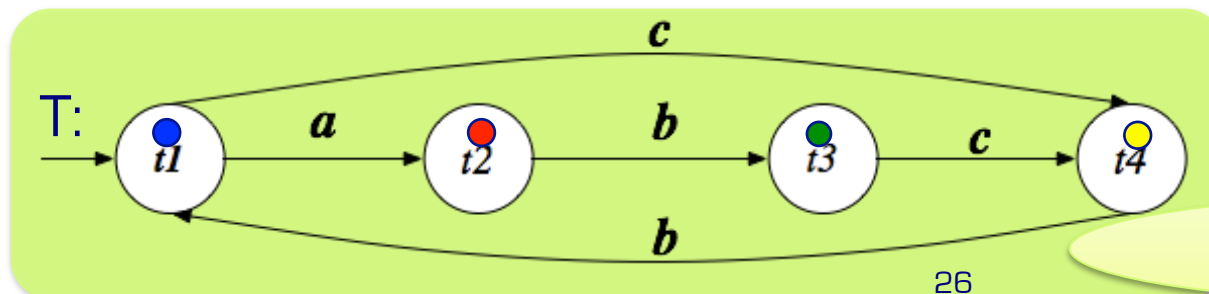
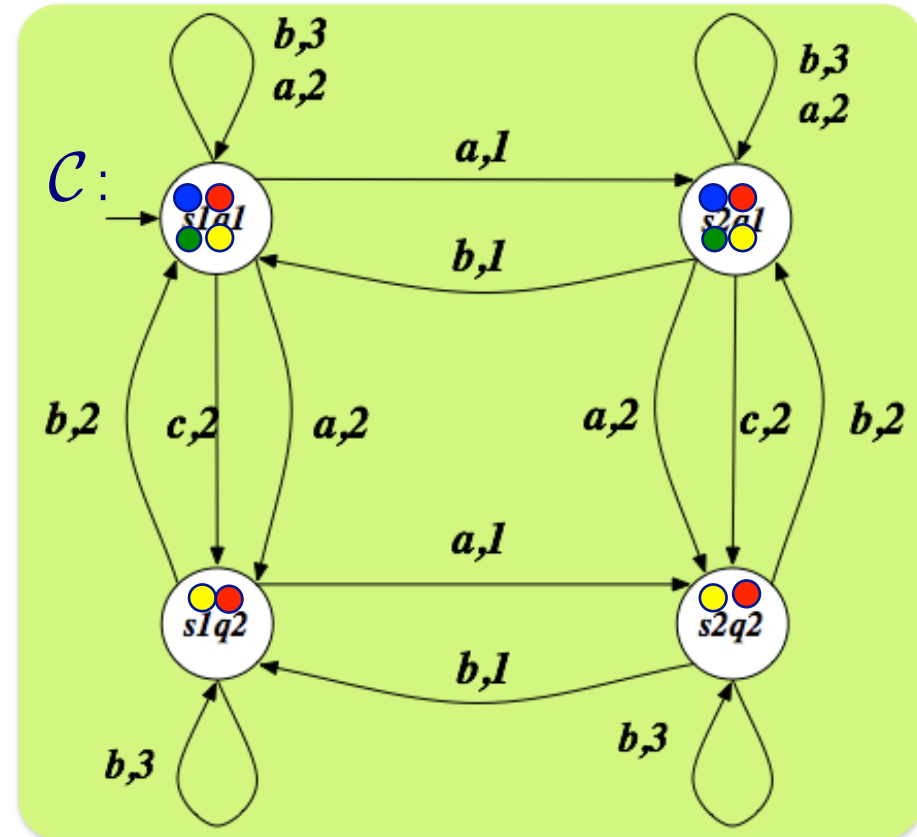
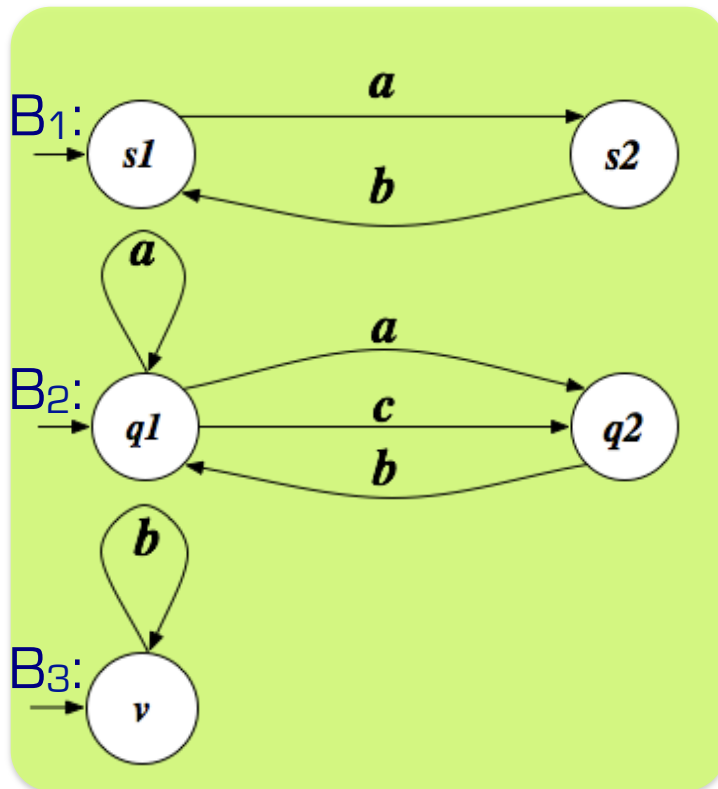
$\neg (\exists s_i \rightarrow_a s'_i \text{ in } B_i \wedge \forall s_i \rightarrow_a s'_i \text{ in } B_i \Rightarrow (t', s_1, \dots, s'_i, \dots, s_n) \in R')\}$

}

return R'

End

Example of composition by simulation



... compute **ND-simulation**

Using simulation for composition

- Given the largest simulation R of T by C , we can build every composition through the **orchestrator generator (OG)**.
- OG** = $\langle A, [1, \dots, n], S_r, s_r^0, \delta_r, \omega_r \rangle$ with
 - A : the **actions** shared by the behaviors
 - $[1, \dots, n]$: the **identifiers** of the available services in the community
 - $S_r = S_T \times S_1 \times \dots \times S_n$: the **states** of the orchestrator generator
 - $s_r^0 = (t^0, s_1^0, \dots, s_n^0)$: the **initial state** of the orchestrator generator
 - $\omega: S_r \times A_r \rightarrow 2^{[1, \dots, n]}$: the **output function**, defined as follows:

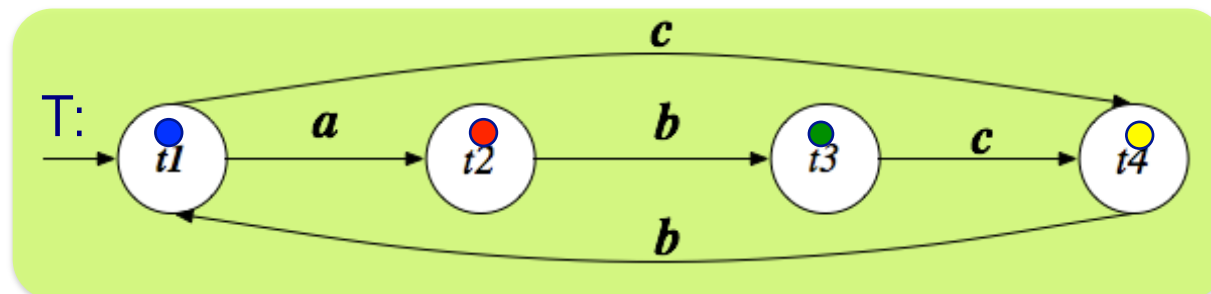
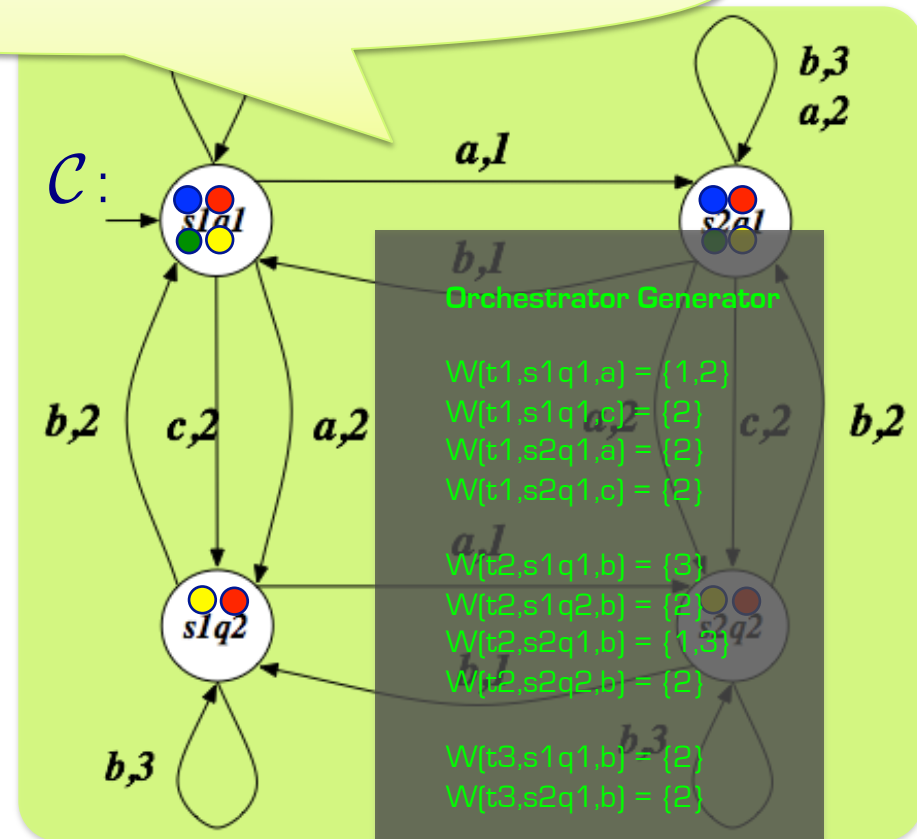
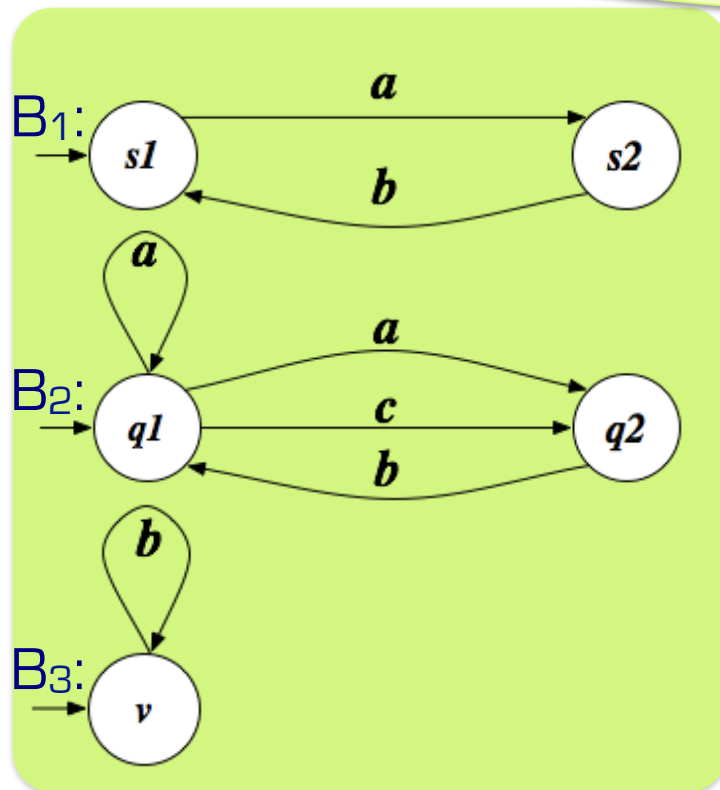
$$\omega(t, s_1, \dots, s_n, a) = \{ i \mid \exists t \rightarrow_a t' \text{ in } T \wedge \exists s_i \rightarrow_a s_i' \text{ in } B_i \wedge (t', s_1, \dots, s_i', \dots, s_n) \in R \}$$

- $\delta \subseteq S_r \times A \times [1, \dots, n] \rightarrow S_r$: the **state transition function**, defined as follows

$$(t, s_1, \dots, s_i, \dots, s_n) \rightarrow_{a,i} (t', s_1, \dots, s_i', \dots, s_n) \text{ iff } i \in \omega(t, s_1, \dots, s_i, \dots, s_n, a)$$

Example of composition by simulation

... compute the **orchestrator generator**



Results

- **Thm:** *choosing at each point any value in returned by the orchestrator generator gives us a composition.*
- **Thm:** *every composition can be obtained by choosing, at each point a suitable value among those returned by the orchestrator generator.*

Note: there infinitely many compositions but only one orchestrator generator that captures them all

- **Thm:** *computing the orchestrator generator is EXPTIME, and in fact exponential only in the number (and not the size) of the available behaviors.*

Composition in the Roman Model was shown to be EXPTIME-hard
[Muscholl&Walukiewicz07]

Just-in-time composition

- Once we have the orchestrator generator ...
- ... we can **avoid choosing any particular composition** a priori ...
- ... and **use directly ω** to choose the available behavior to which delegate the next action.
- We can be *lazy* and make such choice *just-in-time*, possibly adapting reactively to *runtime* feedback.

Just-in-time compositions can be used to reactively act upon failures [KR08]!

Failures

- Available services may become unexpectedly unavailable for various reasons. We consider four kinds of behavioral failures:
 - A service **temporarily freezes**; it will eventually resume in the same state it was in;
 - A service unexpectedly and arbitrarily (i.e., without respecting its transition relation) **changes its current state**;
 - A **service dies**; that is, it becomes permanently unavailable;
 - A dead service unexpectedly comes **alive again** (this is an opportunity more than a failure).

Parsimonious failure recovery (1)

Algorithm Computing ND-simulation - parameterized version

- Input:** - target service $T = \langle A, S_T, t^0, \delta_T, F_T \rangle$
 - available services $\mathcal{S}_i = \langle A, S_i, s_i^0, \delta_i, F_i \rangle, i = 1, \dots, n$
 - relation R_{raw} including the simulated-by relation
 - relation R_{sure} included the simulated-by relation

Output: the **simulated-by** relation (the largest simulation)

Body

$Q = \emptyset$

$Q' = R_{\text{raw}} - R_{\text{sure}}$ // Note $R' = Q' \cup R_{\text{sure}}$

while $[Q \neq Q']$ {

$Q := Q'$

$Q' := Q' - \{[t, s_1, \dots, s_n] \mid \exists t \rightarrow_a t' \text{ in } T \wedge \neg \exists k = 1, \dots, n \text{ s.t.}$

$(\exists s_k \rightarrow_a s'_k \wedge \forall s_k \rightarrow_a s'_k \supset [t', s_1, \dots, s'_k, \dots, s_n] \in Q' \cup R_{\text{sure}})\}$

}

return $Q' \cup R_{\text{sure}}$

End

Parsimonious failure recovery (2)

- Let $[1, \dots, n] = \text{WUF}$ be the available services.
- Let \mathbf{R}_{WUF} be the **simulated-by** relation of target by services WUF.
- Then the following holds:
 - $\mathbf{R}_W \subseteq \pi_W(\mathbf{R}_{\text{WUF}})$
 - $\pi_W(\mathbf{R}_{\text{WUF}})$ is the **projection on W** of a relation: easy to compute
 - *Note: $\pi_W(\mathbf{R}_{\text{WUF}})$ is not a simulation of target by services W*
 - $\mathbf{R}_W \times F \subseteq \mathbf{R}_{\text{WUF}}$
 - $\mathbf{R}_W \times F$ is the **cartesian product** of 2 relations (F is trivial): easy to compute
 - *Note: $\mathbf{R}_W \times F$ is a simulation of target by services WUF*

Parsimonious failure recovery (3)

- If **services F die**
compute simulated-by R_W with $R_{\text{raw}} = \pi_W(R_{W \cup F})$!
- If **dead services F** come back
compute simulated-by $R_W \cup F$ with $R_{\text{sure}} = R_W \times F$!

Remember:

- $R_W \subseteq \pi_W(R_{W \cup F})$
- $R_W \times F \subseteq R_{W \cup F}$ and $R_W \times F$ is a simulation of target by services $W \cup F$

Tools for computing composition based on simulation

- Computing simulation is a well-studied problem (related to computing **bisimulation** a key notion in process algebra). Tools, like the Edinburgh Concurrency Workbench and its clones, can be adapted to compute composition via simulation.
- Also **LTL-based synthesis** tools, like TLV, can be used for (indirectly) computing composition via simulation [Patrizi PhD09]

We are currently focusing on the second approach.

Adding data to the Roman Model

Adding data is crucial in certain contexts:

- **Data** - rich description of the **static information** of interest.
- **Behaviors** - rich description of the **dynamics** of the process

But makes the approach extremely challenging:

- We get to work with infinite transition systems
- Simulation can still be used for capturing composition
- But it cannot be computed explicitly anymore.

We present two orthogonal approaches to deal with them.

The Roman Model: *American tweak*

with Rick Hull + Jianwen Su



Target service

Expressed as a Guarded TS with
parameters

spec. of the desired service behavior

Action ontology

Data-aware Environment or DB/Artifact +
atomic action that affect stored data

spec. of atomic processes and data

...

Available services

Each expressed as a Guarded TS with parameters
spec. of the behavior of available service processes



Actual available processes



Key points

**No available process for
the target service**

Must realize **target
service** by **delegating**
actual actions to
available services

Available services are
stateful, hence must
realize the **target**
using **fragments** of their
computations

Data-Aware Service Composition

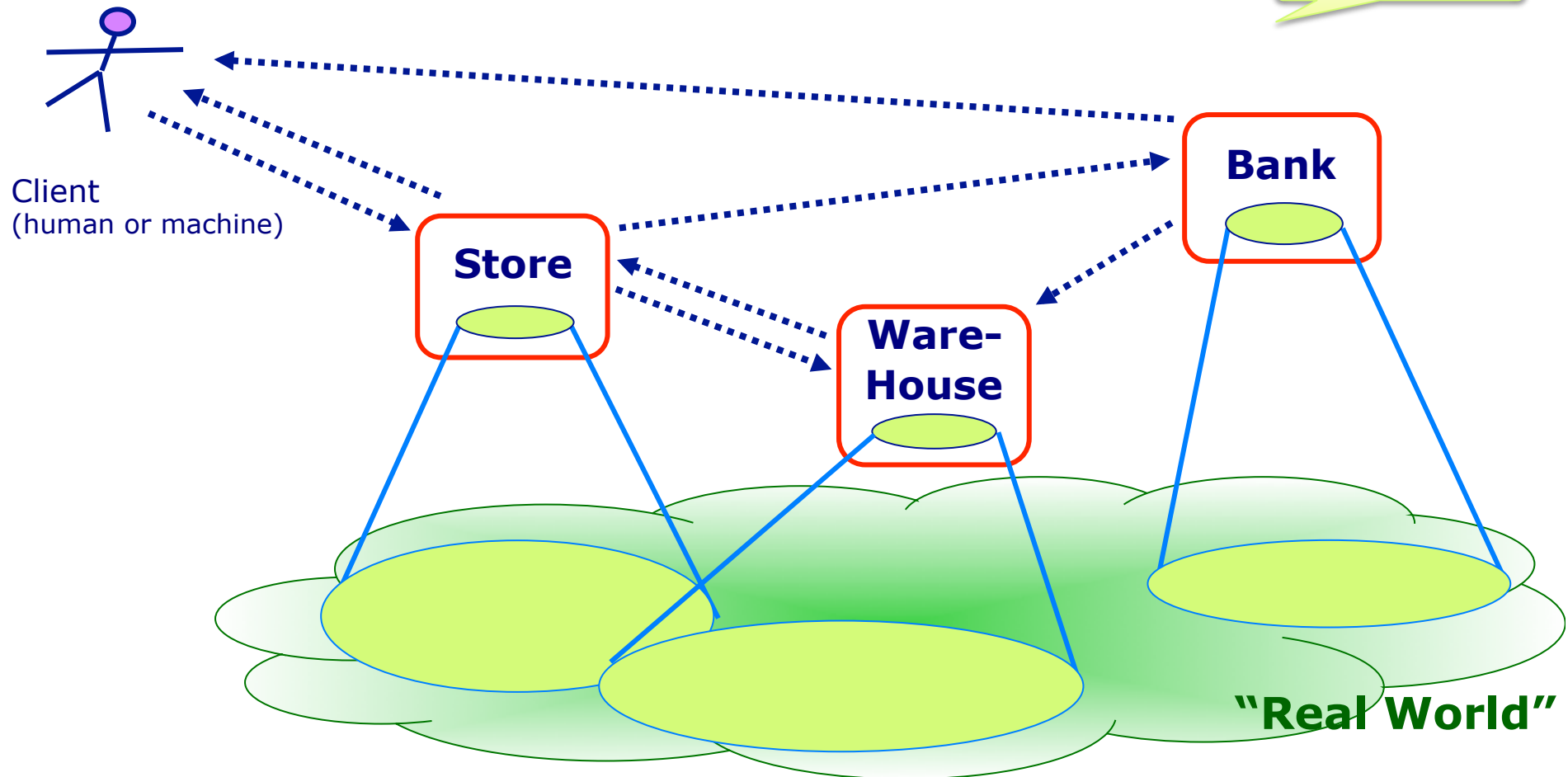
Data-Aware Service Composition

Fabio Patrizi & Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma, Rome, ITALY

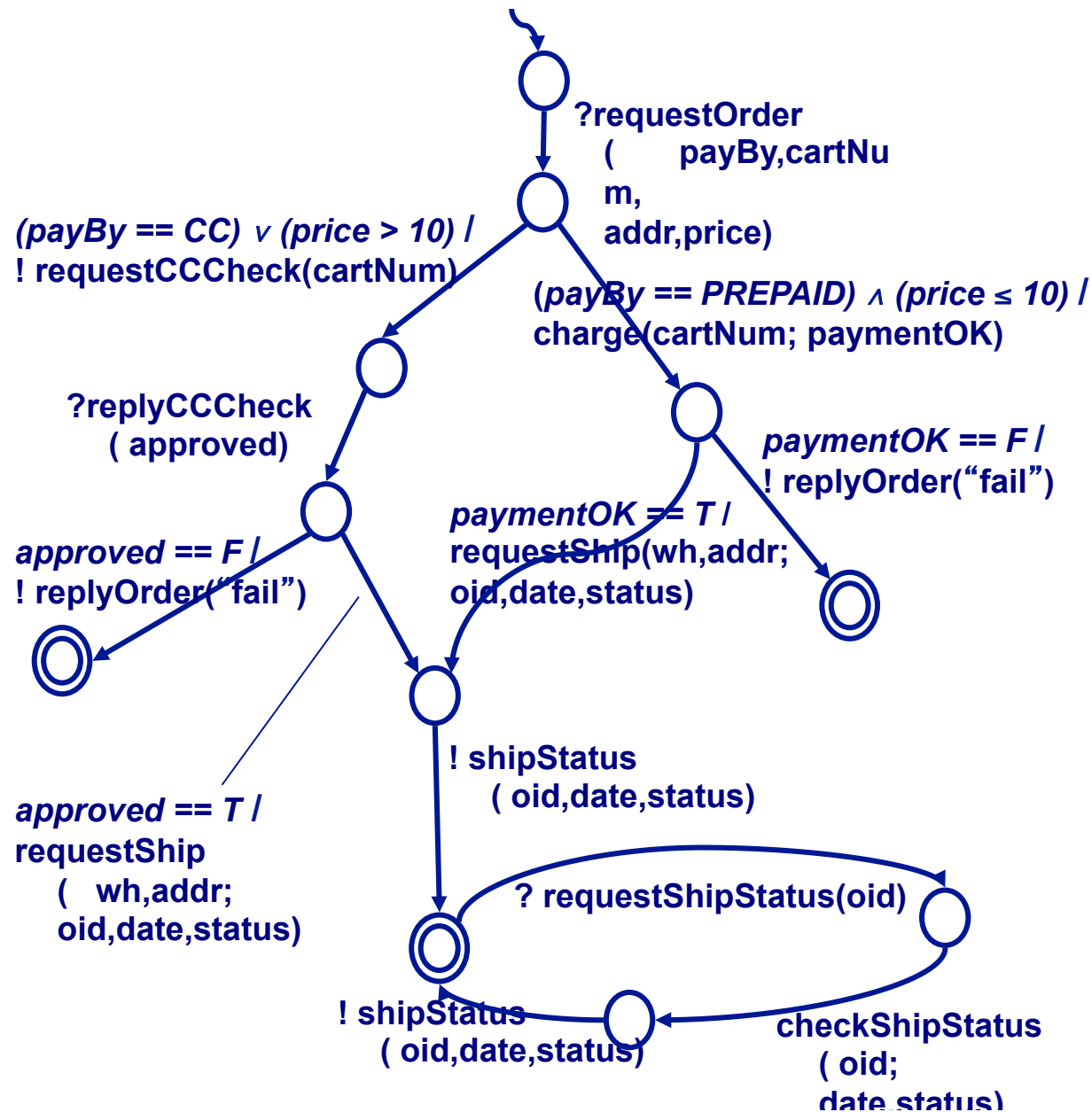
Services act on an integrated view of the world ...

[BCDHM-VLDB05]



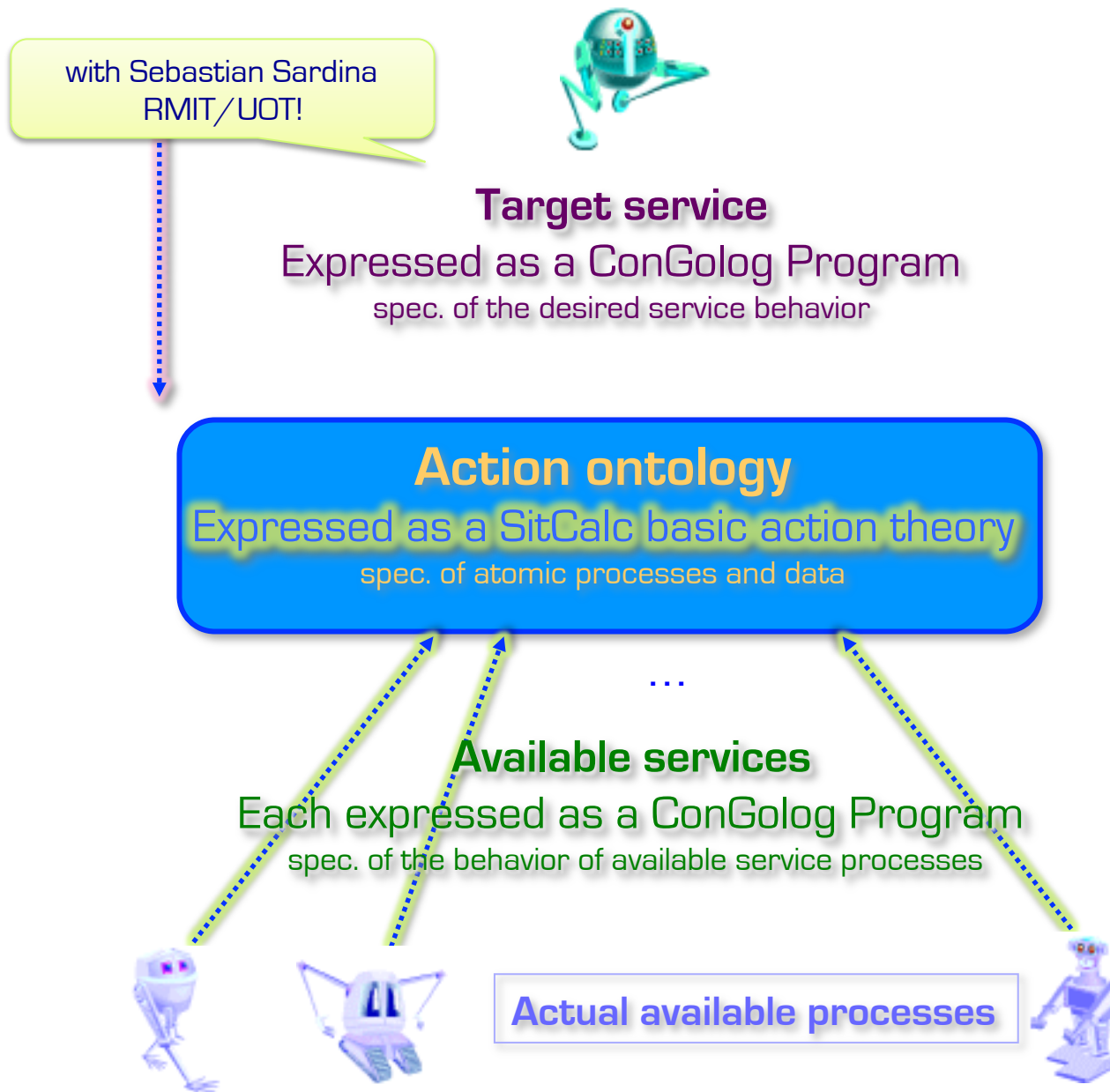
- Actions may impact “real world” – modeled as FOL relations
- Also actions may be messages between services

Service behavior of as abstract finite state machines that query and act on the infinite state world ...



- Local store
- Edge conditions based on local store (and incoming message)
- Edge actions
 - Atomic Process
 - acting on the world
 - set the local store
 - Create/send message
 - Read message

The Roman Model: *Australian/Canadian tweak*



Key points

No available process for the target service

Must realize **target service** by **delegating** actual actions to **available services**

Available services are **stateful**, hence must **realize** the **target** using **fragments** of their **computations**

Composition of ConGolog Programs

Composition of ConGolog Programs

Sebastian Sardina¹ Giuseppe De Giacomo²

¹Department of Computer Science and Information Technology
RMIT University, Melbourne, AUSTRALIA

²Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Sapienza Università di Roma, Rome, ITALY

Mixing data and service integration:

A real challenge for the whole CS

We have all the issues of data integration but in addition ...

- Behavior: description of the **dynamics** of the process!
- Behavior should be formally and **abstractly** described: conceptual modeling of dynamics (not a la OWL-S). *Which?*
 - Workflows community may help
 - Business process community may help
 - Services community may help
 - Process algebras community may help
 - AI & Reasoning about actions community may help
 - DB community may help
 - ... may help
- Techniques for **analysis/synthesis** of **services** in presence of **unbounded data** can come from different communities:
 - Verification (CAV) community: abstraction to finite states
 - AI (KR) community: working directly in FOL/SOL, e.g., SitCalc
 - DB (PODS) community: including theory of conjunctive queries

Artifact-centric approach
promising!

The Roman Model: *Italian dream*

Very preliminary ideas in DL07



Target service

Expressed in **conceptual process**
description language
spec. of the desired service behavior

Action ontology

Expressed as an ontology over the data
+ related conceptual atomic actions
spec. of atomic processes and data

...

Available services

Each expressed **conceptual process** description language
spec. of the behavior of available service processes



Actual available processes



Key points

No available process for
the target service

Must realize **target**
service by **delegating**
actual actions to
available services

Available services are
stateful, hence must
realize the **target**
using **fragments** of their
computations

References

- [ICSOC'03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: Automatic Composition of E-services That Export Their Behavior. ICSOC 2003
- [WES'03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: A Foundational Vision of e-Services. WES 2003
- [TES'04] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: : A Tool for Automatic Composition of Services Based on Logics of Programs. TES 2004
- [ICSOC'04] Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, Diego Calvanese: Synthesis of underspecified composite e-services based on automated reasoning. ICSOC 2004
- [IJCS'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: Automatic Service Composition Based on Behavioral Descriptions. Int. J. Cooperative Inf. Syst. 14(4): 333-376 (2005)
- [VLDB'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, Massimo Mecella: Automatic Composition of Transition-based Semantic Web Services with Messaging. VLDB 2005
- [ICSOC'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Massimo Mecella: Composition of Services with Nondeterministic Observable Behavior. ICSOC 2005
- [SWS'06] Fahima Cheikh, Giuseppe De Giacomo, Massimo Mecella: Automatic web services composition in trustaware communities. Proceedings of the 3rd ACM workshop on Secure web services 2006.
- [AISC'06] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Massimo Mecella. Automatic Web Service Composition: Service-tailored vs. Client-tailored Approaches. In Proc. AISC 2006, International Workshop jointly with ECAI 2006.
- [FOSSACS'07] Anca Muscholl, Igor Walukiewicz: A lower bound on web services composition. Proceedings FOSSACS, LNCS, Springer, Volume 4423, page 274–287 - 2007.
- [ICWS07] Giuseppe De Giacomo, Massimiliano De Leoni, Massimo Mecella, Fabio Patrizi.. Automatic Workflows Composition of Mobile Services. ICWS 2007.
- [IJCAI'07] Giuseppe De Giacomo, Sebastian Sardiña: Automatic Synthesis of New Behaviors from a Library of Available Behaviors. IJCAI 2007.
- [AAAI'07] Sebastian Sardiña, Fabio Patrizi, Giuseppe De Giacomo: Automatic synthesis of a global behavior from multiple distributed behaviors. AAAI 2007.
- [DL07] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati. Actions and programs over description logic ontologies. DL 2007.
- [IJFCS08] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, Fabio Patrizi: Automatic Service Composition via Simulation. IJFCS, 2008
- [KR08] Sebastian Sardiña, Fabio Patrizi, Giuseppe De Giacomo: Behavior composition in the presence of failure. KR 2008.
- [ICAPS08] Sebastian Sardiña, Giuseppe De Giacomo: Realizing Multiple Autonomous Agents through Scheduling of Shared Devices. ICAPS 2008.
- [IEEEBul08] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, Fabio Patrizi. Automatic service composition and synthesis: the Roman Model. Bull. of the IEEE Computer Society Technical Committee on Data Engineering, 31(3):18-22, 2008.