Nondeterministic Available Services

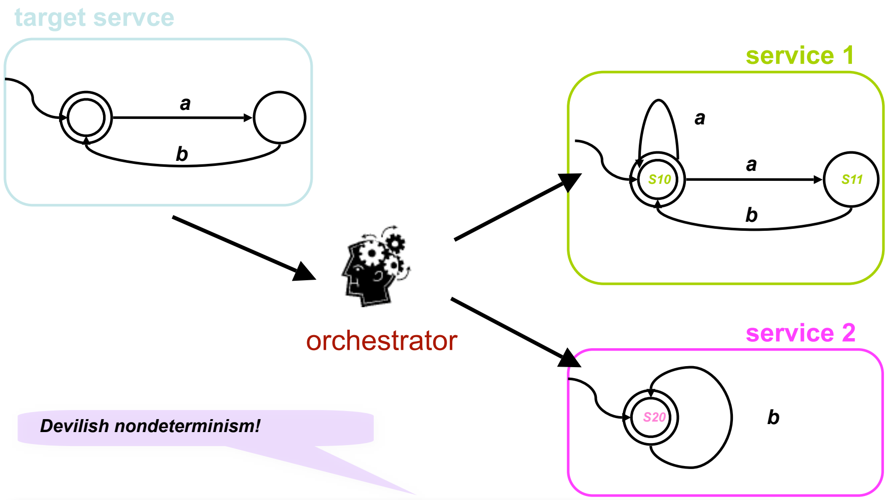
Nondeterminism in Available Services

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])

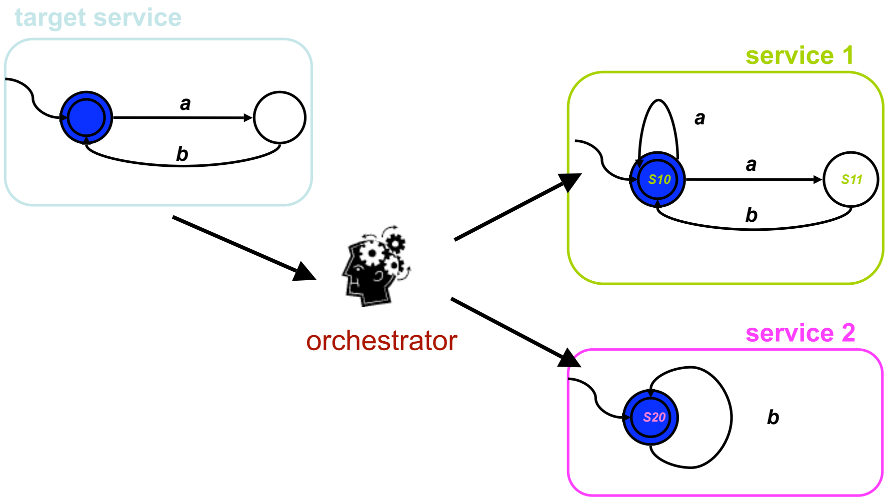
Example: Nondeterministic Available Services



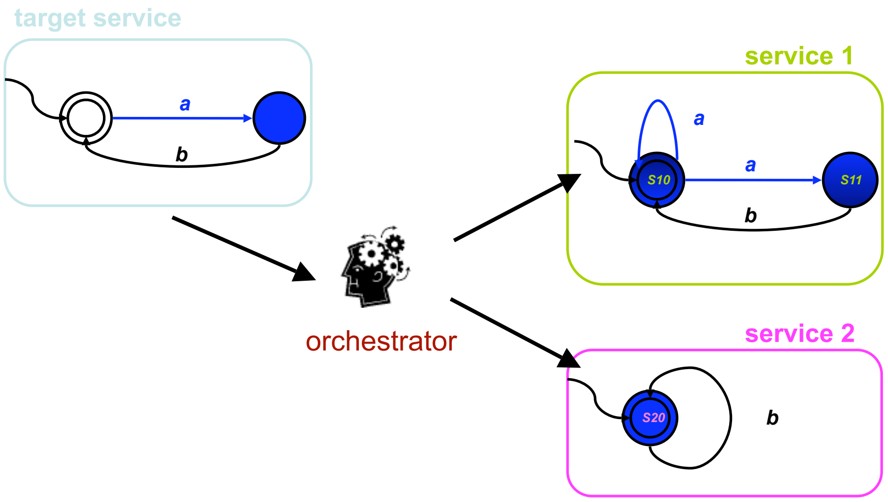
Devilish nondeterminism!

Available services represented as nondeterministic transition systems

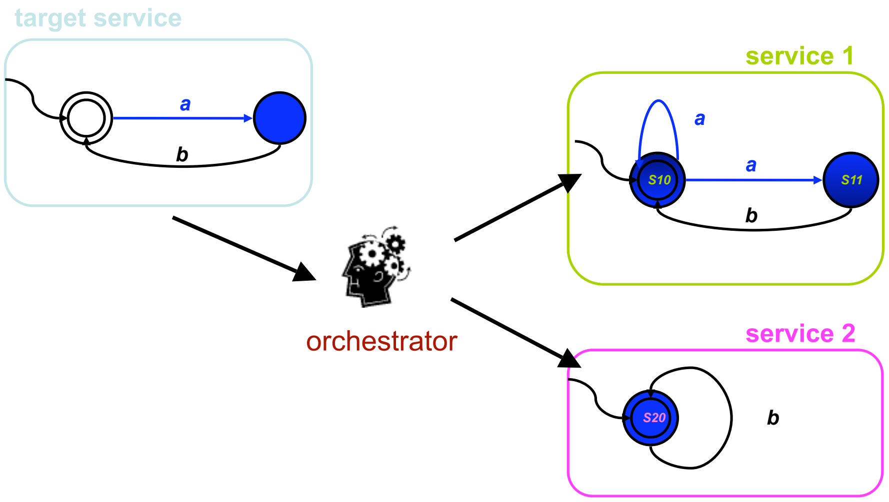
Example: Nondeterministic Available Services



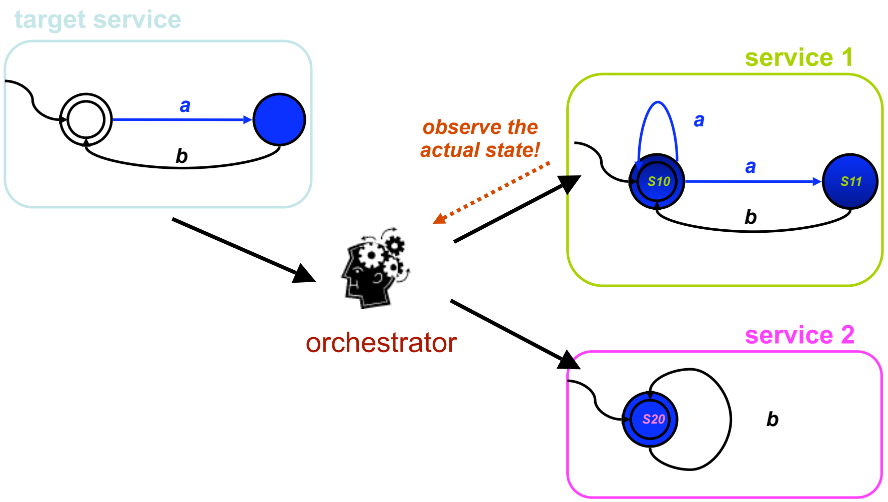
Example: Nondeterministic Available Services



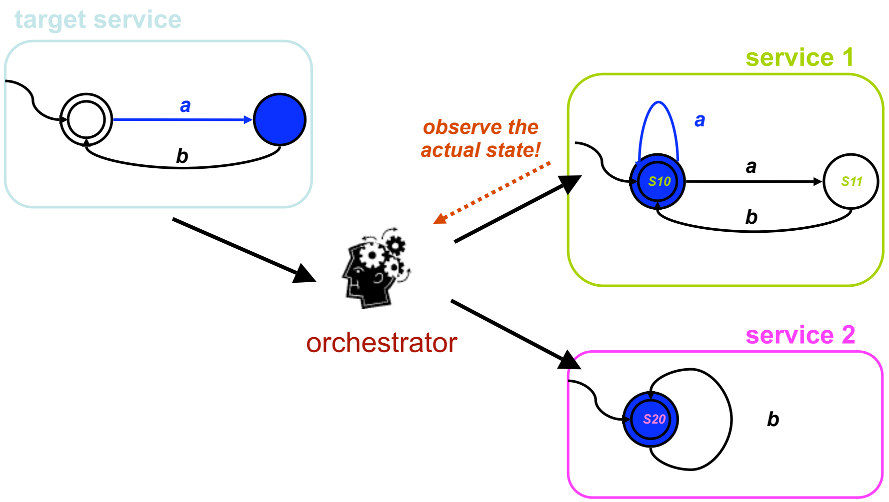
Example: Nondeterministic Available Services



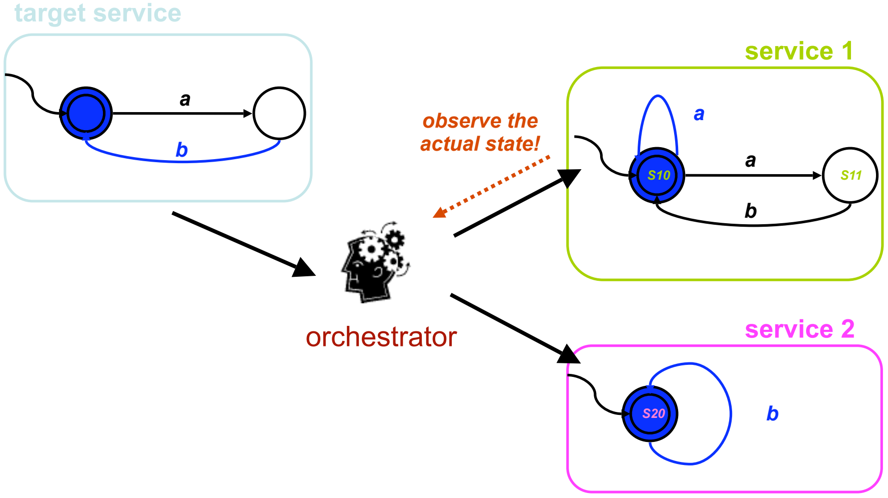
Example: Nondeterministic Available Services



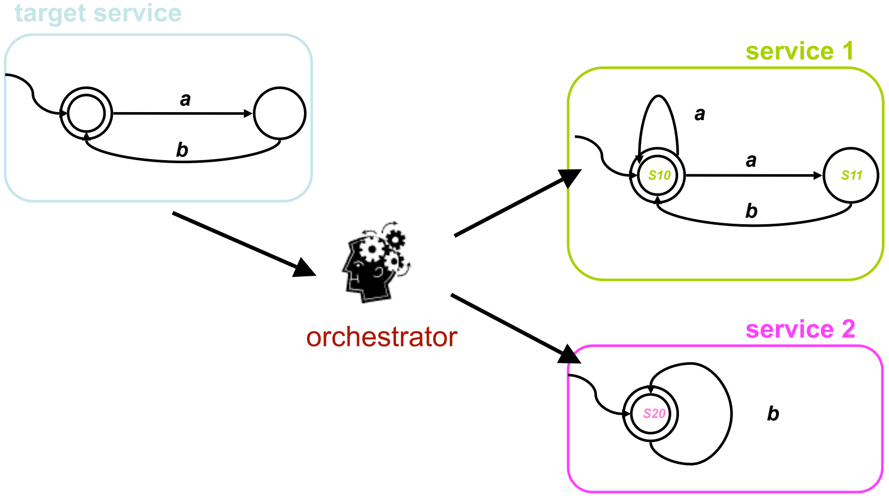
Example: Nondeterministic Available Services



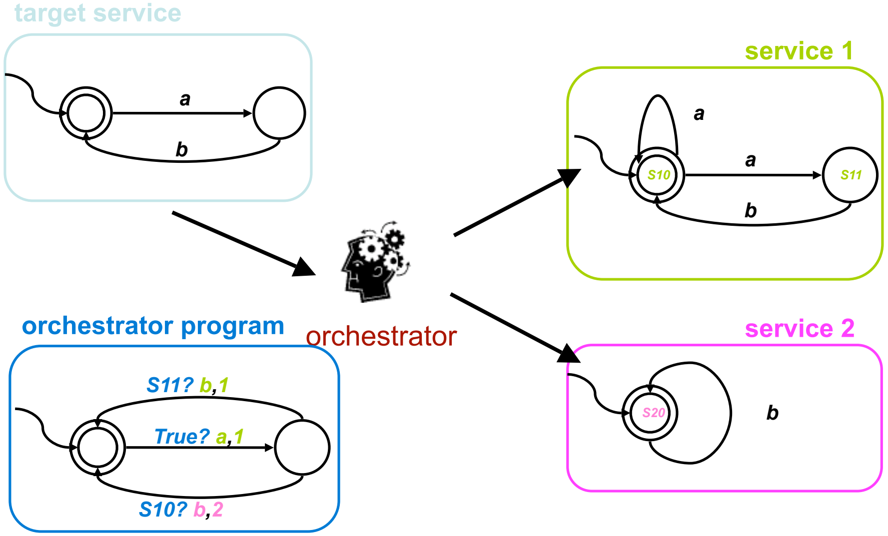
Example: Nondeterministic Available Services



An Orchestrator Program Realizing the Target Service



An Orchestrator Program Realizing the Target Service



Orchestrator Programs

contains all the observable information up the current situation

- **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 of the form, which alternate states of the available services with actions performed:

  $(s_1^0,s_2^0,\dots,s_n^0)\ a_1\ (s_1^1,s_2^1,\dots,s_n^1)\ \dots\ a_k\ (s_1^k,s_2^k,\dots,s_n^k)$

- Observe that to take a decision *P* has **full access to the past**, but no access to the future
- *Problem: synthesize a orchestrator program P that realizes the target service making use of the available services*

Technique1: Reduction to PDL

Basic idea:

- A orchestrator program *P* realizes the target service *T* iff at each point:
  - ∀ transition labeled *a* of the target service *T* …
  - … ∃ an available service B_i (the one chosen by *P*) which can make an *a*-transition …
  - … and ∀ *a*-transition of B_i realize the *a*-transition of *T*
- Encoding in PDL:
  - ∀ transition labeled *a* … use **branching**
  - ∃ an available service B_i … use underspecified predicates **assigned through SAT**
  - ∀ *a*-transition of B_i … : use **branching** again

Technical Results: Theoretical

Thm[IJCAI'07] Checking the existence of orchestrator program realizing the target service is **EXPTIME-complete.**

EXPTIME-hardness due to Muscholl&Walukiewicz07 for deterministic services

Thm [IJCAI'07] If a **orchestrator program exists** there exists one that is **finite state.**

Exploits the finite model property of PDL

Note: same results as for deterministic services!

Technical Results: Practical

Reduction to PDL provides also a practical sound and complete technique to compute the orchestrator program also in this case

eg, PELLET @ Univ. Maryland

- Use state-of-the-art tableaux systems for OWL-DL for checking SAT of PDL formula Φ coding the composition existence
- *If SAT, the tableau returns a finite model of* Φ

exponential in the size of the behaviors

- Project away irrelevant predicates from such model, and possibly minimize
- *The resulting structure is a finite orchestrator program that realizes the target behavior*

polynomial in the size of the model

Nondeterministic Available Services: Technique based on Composition via ND-Simulation

Composition via ND-Simulation

- We consider binary relations *R* satisfying the following co-inductive condition (ND-similarity):

  $(t,(s_1, .., s_n)) \in R$ implies that
  - if t is *final* then s_i, with i=1, .., n, is *final*
  - for **all** actions a
    - If $t \to_a t'$ then $\exists k \in 1..n.$
      - $\exists s_k' .\ s_k \to_a s_k'$
      - $\forall s_k'.\ s_k \to_a s_k' \supset (t',(,s_1,..,s_k', .., s_n)) \in R$

  Note similar in the spirit to simulation relation!
  But more involved, since it deals with

  - *the existential choice (as the simulation) of the service, and*
  - *the universal condition on the nondeterministic branches!*

- A composition realizing a target service TS TS_t exists if there **exists** a relation *R* satisfying the above condition between the initial state t^0 of TS_t and the initial state $(s_1^0, .., s_n^0)$ of the community big TS TS_c.

- Notice if we take the union of all such relation *R* then we get the largest relation ***RR*** satisfying the above condition.

- A composition realizing a target service TS T exists iff $(t^0, (s_1^0, .., s_n^0)) \in$ ***RR***.

Algorithm for ND-simulation

Algorithm Compute (ND-)simulation
Input: target service $T = \langle A, S_T, t^0, \delta_T, F_T \rangle$ and ..
available services $\mathcal{S}_i = \langle A, S_i, s_i^0, \delta_i, F_i \rangle$, i = 1,..,n

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

Body

```
R = ∅
R' = S_T × S_1 ×..× S_n
while (R ≠ R') {
    R := R'
    R' := R'  -  {(t, s_1,..,s_n) | ∃ t →_a t' in T ∧ ¬∃ k = 1,..,n s.t.
                  (∃ s_k →_a s_k' ∧ ∀ s_k →_a s'_k ⊃ (t', s_1,..,s'_k,..,s_n) ∈ R' )}
}
return R'
```

End

Composition via ND-Simulation

- Given the maximal ND-simulation ***RR*** form TS_t to TS_c (which includes the initial states), we can build the ***orchestrator generator***.

- This is an orchestrator program that can change its behavior reacting to the information acquired at run-time.

- Def: $OG = \langle A, [1,\ldots,n], S_r, s_r^0, \omega_r, \delta_r, F_r \rangle$ with
  - A : the **actions** shared by the community
  - [1,…,n]: the **identifiers** of the available services in the community
  - $S_r = S_t \times S_1 \times \cdots \times S_n$: the **states** of the orchestrator program
  - $s_r^0 = (s^0_t, s^0_1, \ldots, s^0_m)$: the **initial state** of the orchestrator program
  - $F_r \subseteq \{ (s_t , s_1 , \ldots, s_n) \mid s_t \in F_t$: the **final states** of the orchestrator program
  - $\omega_r : S_r \times A_r \to [1,\ldots,n]$: the **service selection function**, defined as follows:

    ω_r(t, s_1,..,s_n, a) = { i | TS_t and **TS_i** can do **a** and remain in **RR}**

    i.e. …$=\{i \mid s_t \to_{a,} s'_t \wedge \exists s_i'.\ s_i \to_{a,} s_i' \wedge \forall s_i'.\ s_i \to_{a,} s_i' \supset (s_t', (s_1 , \ldots, s_i' ,.., s_n)) \in$ ***RR***$\}$
  - $\delta_r \subseteq S_r \times A_r \times [1,\ldots,n] \times S_r$: the **state transition relation**, defined as follows:
    - Let $k \in \omega_r(s_t, s_1 , \ldots, s_k , \ldots, s_n, a)$ then
      $(s_t, s_1 , \ldots, s_k , \ldots, s_n) \to_{a,k} (s_t', s_1 , \ldots, s'_k , \ldots, s_n)$ for each $s_k \to_{a,} s'_k$

Composition ND-Simulation

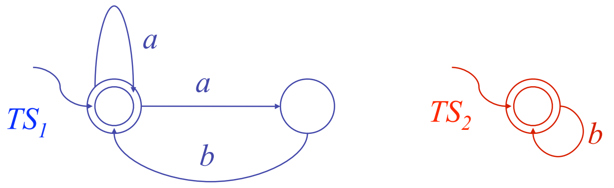
- Computing ***RR*** is polynomial in the size of the target service TS and the size of the community TS…
- … composition can be done in EXPTIME in the size of the available services

- For generating OG we need only to compute ***RR*** and then apply the template above

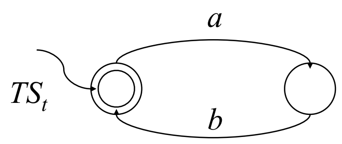
- For running the OG we need to store and access ***RR***
  (polynomial time, exponential space) …
- … and compute ω_r and δ_r at each step *(polynomial time and space)*

Example of Composition

Available Services



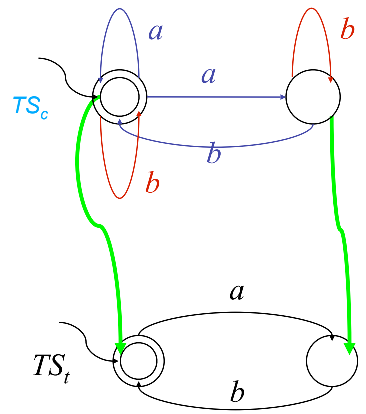
Target Service



Example of Composition

Community TS

Target Service



Composition exists!

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).

Just-in-time composition

- Once we have the controller generator ...
- ... we can **avoid choosing any particular composition** apriori ...
- ... and **use directly *ωr*** 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.

Parsimonious failure recovery (1)

Algorithm Computing ND-simulation - parameterized version

Input: - target service T = <A, S_T, t^0, δ_T, F_T>

- available services $\mathcal{S}_i$= <A, S_i, s_i^0, δ_i, F_i> , i = 1,..,n
- relation $\mathbf{R_{raw}}$ including the simulated-by relation
- relation $\mathbf{R_{sure}}$ included the simulated-by relation

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

Body

$Q = \emptyset$

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

while ($Q \neq Q'$) {

$Q := Q'$

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

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

}

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

End

Parsimonious failure recovery (2)

- Let [1,.., n] = W∪F be the available services.
- Let $R_{W \cup F}$ be the **simulated-by** relation of target by services W∪F.
- Then the following holds:
- $R_W \subseteq \Pi_W(R_{W \cup F})$
  - $\Pi_W(R_{W \cup F})$ is the **projection on W** of a relation: easy to compute
  - *Note: $\Pi_W(R_{W \cup F})$ is not a simulation of target by services W*
- $R_W \times F \subseteq R_{W \cup F}$
  - $R_W \times F$ is the **cartesian product** of 2 relations (F is trivial): easy to compute
  - *Note: $R_W \times F$ is a simulation of target by services W∪F*

Extension to the Roman Model

Parsimonious failure recovery (3)

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

- Remember:
  - $R_W \subseteq \Pi_W(R_W \text{UF})$
  - $R_W \times F \subseteq R_W \text{UF}$ and $R_W \times F$ is a simulation of target by services W∪F

Extensions

- **Nondeterministic** (angelic) **target** specification
  - Loose specification in client request
  - **Angelic (don't care)** vs devilish (don't know) nondeterminism
  - See [ICSOC'04]
- **Distributing** the orchestration
  - Often a centralized orchestration is unrealistic: eg. services deployed on mobile devices
    - too tight coordination
    - too much communication
    - orchestrator cannot be embodied anywhere
  - Drop centralized orchestrator in favor of **independent controllers** on single available services (exchanging messages)
  - Under suitable conditions: a distributed orchestrator exists iff a centralized one does
  - Still decidable (EXPTIME-complete)
  - See [AAAI'07]

See later

- Dealing with **data**
  - This is the single most difficult issue to tackle
    - First results: actions as DB updates, see [VLDB'05]
    - Literature on Abstraction in Verification
  - From finite to **infinite transition** systems!
- **Security and trust** aware composition [SWS'06]
- Automatic **Workflows** Composition of Mobile Services [ICWS'07]

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: 43-58

[WES'03] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: A Foundational Vision of e-Services. WES 2003: 28-40

[TES'04] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella: : A Tool for Automatic Composition ofServices Based on Logics of Programs. TES 2004: 80-94

[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: 105-114

[IJCIS'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: 613-624

[ICSOC'05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Massimo Mecella: Composition of Services with Nondeterministic Observable Behavior. ICSOC 2005: 520-526

[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. Pages: 43 - 52.

[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.

[IJCAI'07] Giuseppe De Giacomo, Sebastian Sardiña: Automatic Synthesis of New Behaviors from a Library of Available Behaviors. IJCAI 2007: 1866-1871

[AAAI'07] Sebastian Sardiña, Fabio Patrizi, Giuseppe De Giacomo: Automatic synthesis of a global behavior from multiple distributed behaviors. In Proceedings of the Conference on Artificial Intelligence (AAAI), Vancouver, Canada, July 2007.