Composition of Services that Share an Infinite-State Blackboard

Fabio Patrizi & Giuseppe De Giacomo
SAPIENZA Università di Roma, Italy
Basic ideas

Target service
Expressed as a Transition System
spec. of the desired service behavior

Shared set of atomic operations
+ shared blackboard

Available services
Each expressed as a Transition System
spec. of the behavior of available service processes

Actual available processes

Key points

• Services are stateful
• They share atomic operations
• They act over a shared blackboard
• No available process for the target service
• Must realize target service by delegating operation executions to available services ...
• ... by repurposing fragment of available services to realize the requested target service
Simple example of service composition
without the shared blackboard

For simplicity we don’t consider blackboard for now.
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

Target service

Orchestrator

Service 1

Service 2

Observe the actual state!
Simple example of service composition

target service

orchestrator

observe the actual state!

service 1

service 2
Simple example of service composition

target service

orchestrator

observe the actual state!

service 1

service 2
Simple example of service composition

- **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:

  $$\{(s_1^0,s_2^0,...,s_n^0) \ a_1 \ (s_1^1,s_2^1,...,s_n^1) \ ... \ a_k \ (s_k^1,s_2^k,...,s_n^k)\}$$

- 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 relation

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

$$(t, s_1, \ldots, s_n) \in R \text{ implies that }$$
for all $t \rightarrow_a t'$ in $T$, exists a $B_i \in C$ s.t.

- $\exists s_i \rightarrow_a s'_i$ in $B_i$ \wedge
- $\forall s_i \rightarrow_a s'_i$ in $B_i \Rightarrow (t', s_1, \ldots, s'_i, \ldots, s_n) \in R$

- If exists a simulation relation $R$ (such that $(t^0, s_1^0, \ldots, s_n^0) \in R$), then we say that or $T$ is simulated by $C$ (or $C$ simulates $T$).

- Simulated-by is
  - (i) a simulation;
  - (ii) the largest simulation.

*Simulated-by is a coinductive definition*
Using simulation for composition

- Given the largest simulation $R$ of $T$ by $C$, we can build every composition through the orchestrator generator (OG).

- **OG** = $< A, [1,\ldots,n], S_r, s^0_r, \delta_r, \omega_r,>$ with
  - $A$ : the *actions* shared by the behaviors
  - $[1,\ldots,n]$ : the *identifiers* of the available services in the community
  - $S_r = S_T \times S_1 \times \ldots \times S_n$ : the *states* of the orchestrator generator
  - $s^0_r = (t^0, s^0_1, \ldots, s^0_n)$ : the *initial state* of the orchestrator generator
  - $\omega : S_r \times A_r \rightarrow 2^{[1,\ldots,n]}$ : the *output function*, defined as follows:
    $$\omega(t, s_1,\ldots,s_n, a) = \{ i \mid \exists t \rightarrow_a t', \exists s_i \rightarrow_a s'_i \in B_i \land (t', s_1,\ldots,s_i,\ldots,s_n) \in R \}$$

- $\delta \subseteq S_r \times A \times [1,\ldots,n] \rightarrow S_r$ : the *state transition function*, defined as follows
  $$(t, s_1,\ldots,s_i,\ldots,s_n) \rightarrow_{a,i} (t', s_1,\ldots,s'_i,\ldots,s_n) \text{ iff } i \in \omega(t, s_1,\ldots,s_i,\ldots,s_n, a)$$
Adding data

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 are currently investigating **two orthogonal approaches to deal with them**.

• **Based on SitCalc** [see “Composition of ConGolog Programs” - IJCAI09 - next Wednesday , July 15]
• **Based on “symbolic abstraction”** [eg., the current paper]
Infinite-state shared blackboard

We consider a shared blackboard, where data can be added and removed.

- The blackboard is modeled as an associative list: set of pairs \((\text{attribute}, \text{value})\)
- The maximal size of the blackboard is fixed...
- ... but it can contain values an infinite, ordered \((\leq)\) and dense (interpretation) domain \(\Delta\) (e.g., alphanumeric strings).

Example of blackboard \(R\):

<table>
<thead>
<tr>
<th>person</th>
<th>Giuseppe De Giacomo</th>
</tr>
</thead>
<tbody>
<tr>
<td>person2</td>
<td>Fabio Patrizi</td>
</tr>
</tbody>
</table>

The blackboard is a sort of artifact, see [Deutsch,Hull,Patrizi,Vianu-ICDT09]
Atomic operations on the blackboard

- tuple insertion/modification: $R(\chi) = \nu$
- tuple deletion: $\neg R(\chi)$

Examples

Del:  

<table>
<thead>
<tr>
<th>lastname1</th>
<th>De Giacomo</th>
<th>$\neg R(lastname2)$</th>
<th>lastname1</th>
<th>De Giacomo</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastname2</td>
<td>Patrizi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mod:  

<table>
<thead>
<tr>
<th>lastname1</th>
<th>De Giacomo</th>
<th>$R(lastname1) = Rossi$</th>
<th>lastname1</th>
<th>Rossi</th>
</tr>
</thead>
</table>

Ins:  

<table>
<thead>
<tr>
<th>lastname1</th>
<th>Rossi</th>
<th>$R(lastname3) = Patrizi$</th>
<th>lastname1</th>
<th>Rossi</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastname3</td>
<td>Patrizi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Attributes can be added and removed
- Atomic operations can be arbitrarily concatenated
Atomic operations on the blackboard (cont.d)

Operations with formal parameters:

\[ o(q) = \{ \langle \phi_1(q), \nu_1(q) \rangle, \ldots, \langle \phi_m(q), \nu_m(q) \rangle \} \]

- \( \phi_i(q) \), condition over \( R, \Delta, \leq \)
  - e.g.: \( \text{isDef}(R(name)) \land R(name) \leq q \land q \leq R(name) \)
- \( \nu_i(q) \), sequence of atomic operations
  - e.g.: \( R(name) = q, \neg R(lastname), \ldots \)
- the formal parameter \( q \) is resolved with actual parameter given by the client at run time.

Successor relation:

\[ \bar{R} \xrightarrow{\circ \vec{q}} \bar{R}' (q \in \Delta) \text{ iff:} \]

- \( \exists \phi_i(q) \mid \langle \bar{R}, \leq \rangle \models \phi_i(\vec{q}) \)
- \( \bar{R} \nu_i(\vec{q}) \xrightarrow{\nu} \bar{R}' \)

- Nondeterministic: several \( \phi_i \)'s can be satisfied at the same time
- Not input-bounded: client can choose any value from \( \Delta \) as actual parameter
- For simplicity we use 1 parameter per operation in this talk
Composition

Given:
- an initial state of the blackboard $R_0$
- a deterministic target service $S_t$
- a set of $n$ available nondeterministic services $\{S_1, \ldots, S_n\}$

Find a composition, i.e., a simulation $S_t$ by the asynchronous product of $S_1, \ldots, S_n$ $\Sigma$, such that $\langle s_{t0}, \langle s_{10}, \ldots, s_{n0} \rangle, R_0 \rangle \in \Sigma$

As before, the core problem amounts to building a simulation relation.
From infinite to finite states

**Objective**: build a finite abstraction on the infinite blackboard configurations and adopt finite-state reasoning

- The blackboard is infinite-state
- But for every blackboard state $\bar{R}$ we have $|\text{adom}(\bar{R})| \leq b$
- We get a finite representation of the infinite-state system by abstracting over actual values in the blackboard.
Abstracting over actual values

**Intuition:** since $|\text{adom}(\bar{R})| \leq b$...

- replace $\text{adom}(\bar{R})$ with a symbolic version $\hat{\text{adom}}(\bar{R}) = \{\hat{a}_1, \ldots, \hat{a}_b\}$
- define a mapping $m : \text{adom}(\bar{R}) \longrightarrow \hat{\text{adom}}(\bar{R})$ which preserves $\leq$ and $\bar{R}$ (resp. $\hat{\leq}$ and $\hat{R}$)

**Example**

\[
\begin{array}{c|c}
12 & 3 \\
1 & 15 \\
3 & 3 \\
\end{array}
\]

$\text{adom}(\bar{R}) = \{1, 3, 12, 15\}$

$1 \leq 3 \leq 12 \leq 15$

$\bar{R}$

$\hat{R}$

\[
\begin{array}{c|c}
\hat{a}_1 & \hat{a}_2 \\
\hat{a}_3 & \hat{a}_4 \\
\hat{a}_2 & \hat{a}_2 \\
\end{array}
\]

$\hat{\text{adom}}(\bar{R}) = \{\hat{a}_1, \hat{a}_2, \hat{a}_3, \hat{a}_4\}$

$a_3 \leq a_2 \leq a_1 \leq a_4$

$m(1) = \hat{a}_3, m(3) = \hat{a}_2$

$m(12) = \hat{a}_1, m(15) = \hat{a}_4$

$\leftarrow m \rightarrow$
Non-symbolic vs. symbolic simulation

Q: What is the relation between (non-symbolic) simulation and symbolic simulation (the simulation performed on the symbolic abstraction)?

A: they are equivalent (!)

Theorem:
A (non-symbolic) simulation of the target service by the available services exists iff the symbolic simulation does.

Finite-state techniques apply!

*From the orchestrator generator associated to the symbolic simulation one easily extracts the orchestrator generator for the original (non-symbolic) setting.*
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