Workshop on Software Engineering for Adaptive and Self-Managing Systems
SEAMS 2010

Live Goals for Adaptive Service Compositions

Luciano Baresi - Liliana Pasquale

Politecnico di Milano

Motivations

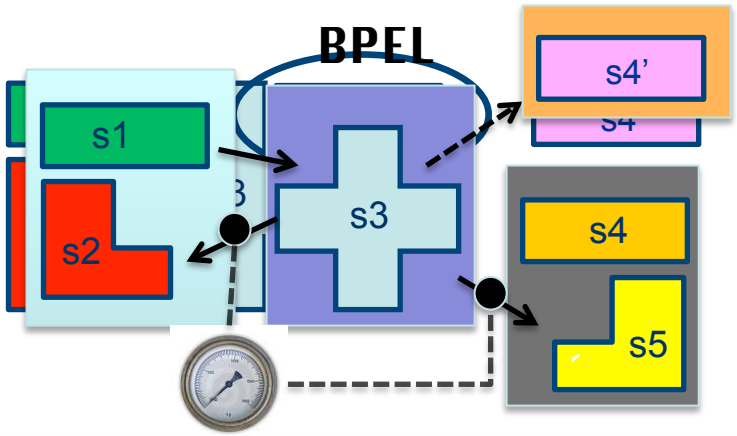
- The SOA paradigm is intrinsically self-adaptive
  - BPEL: technical solutions for monitoring/adaptation

When

adaptation must be activated at runtime

What

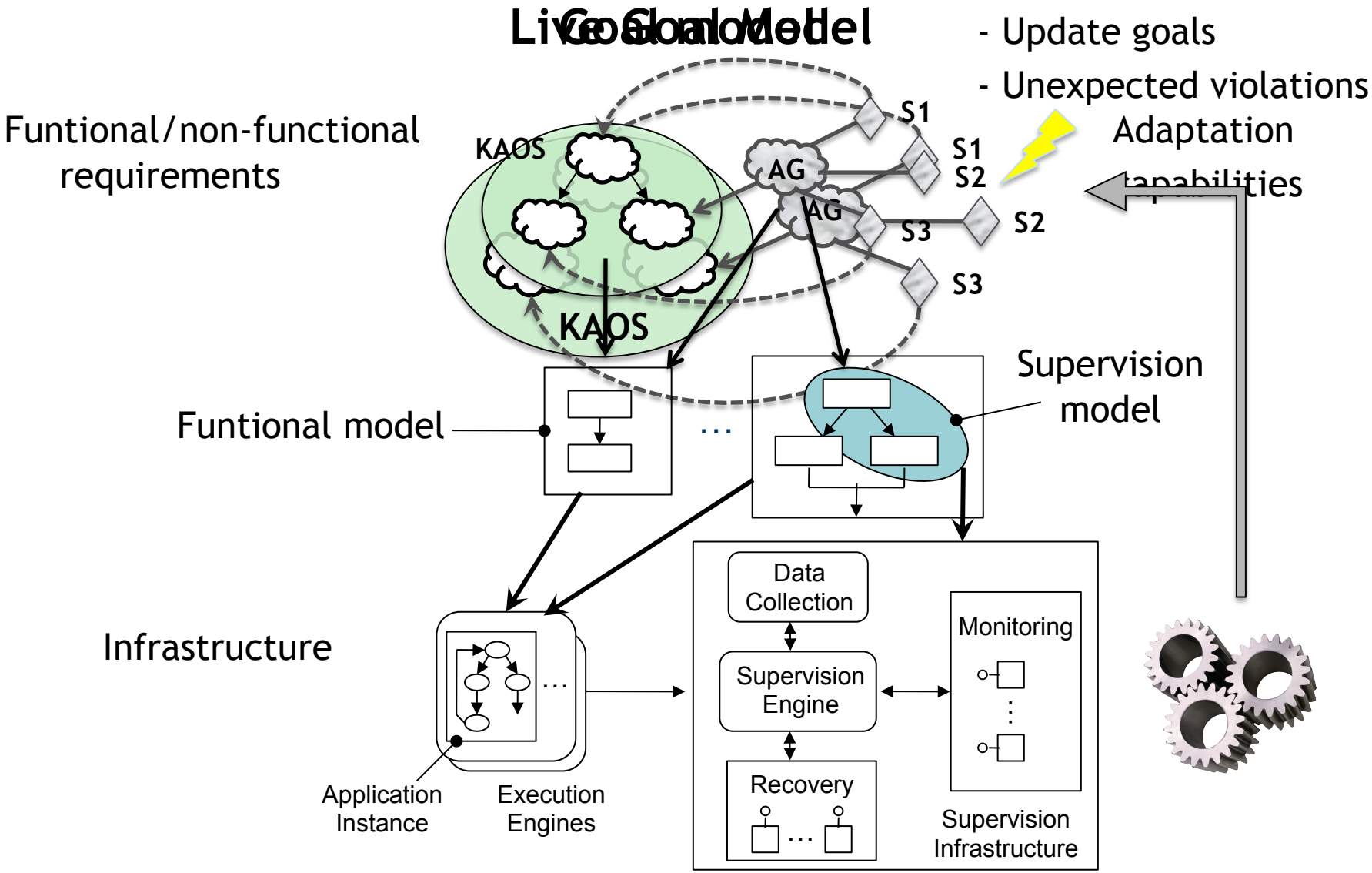
Adaptation capabilities are needed



How

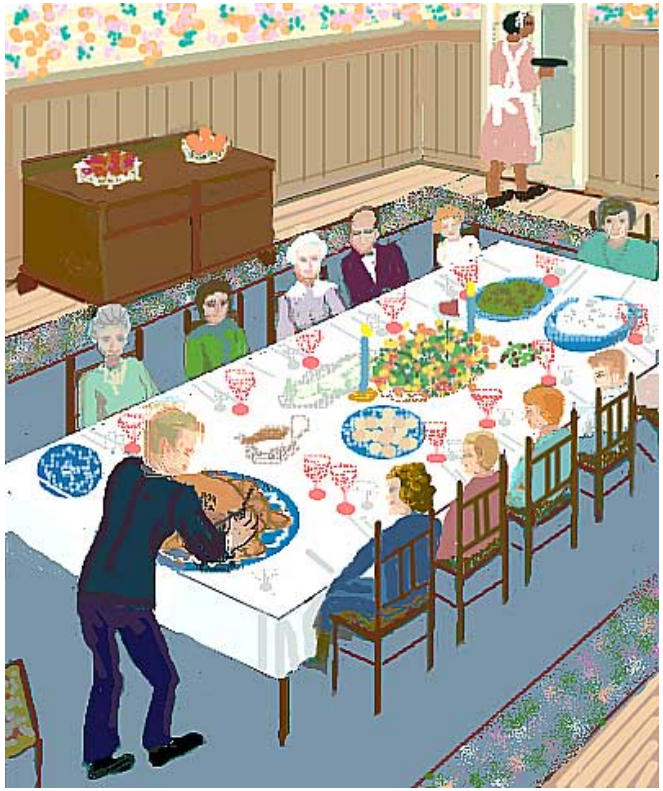
enrich BPEL processes with self/adaptation capabilities

Our solution



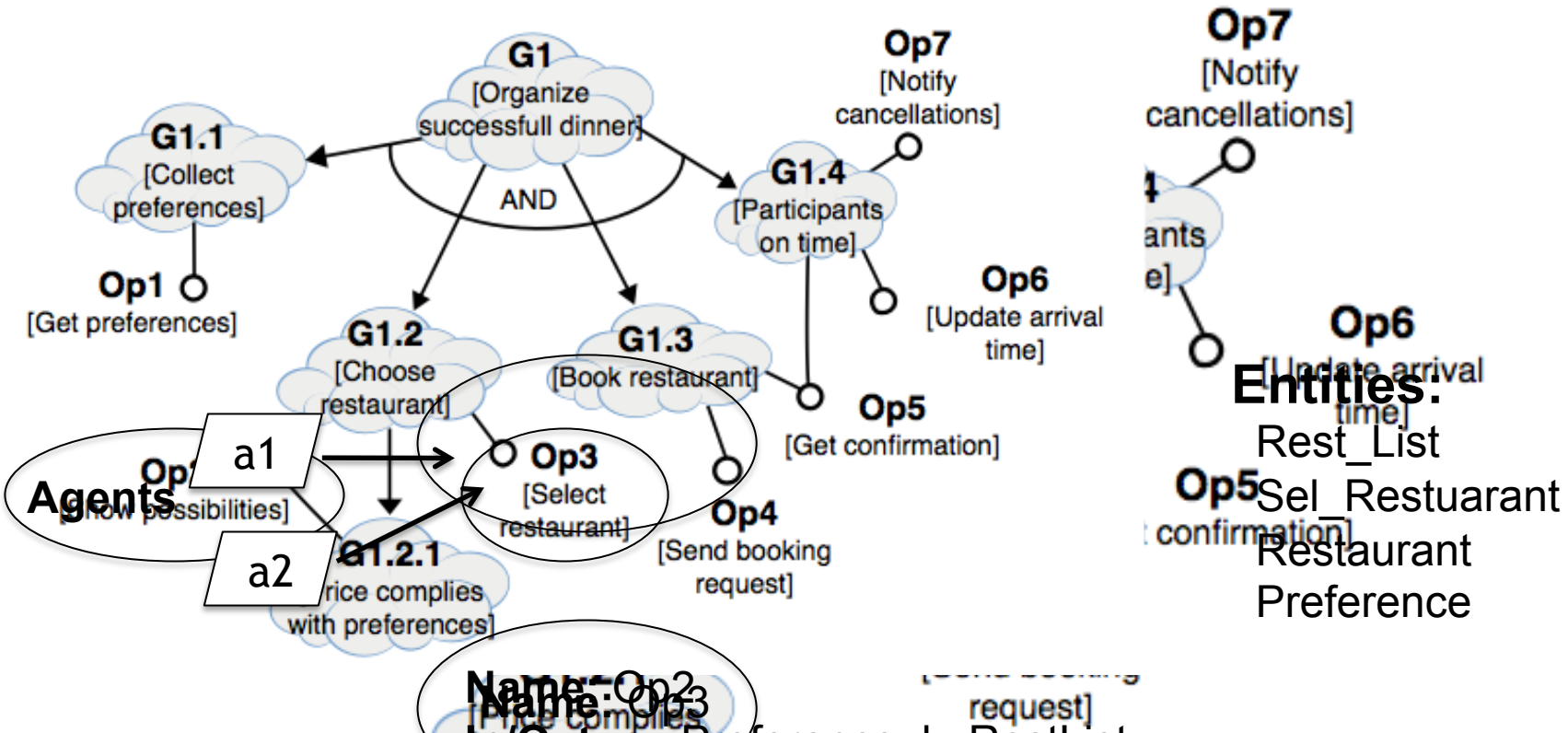
Our approach by example

- **Example**
  - Model an application for organizing dinners



- **Requirements**
  - Book a restaurant that matches users' preferences (cost)
- **Adaptation capabilities**
  - Try to meet user preferences when possible.
  - Otherwise choose the cheapest solution (takeaways)

KAOS goals



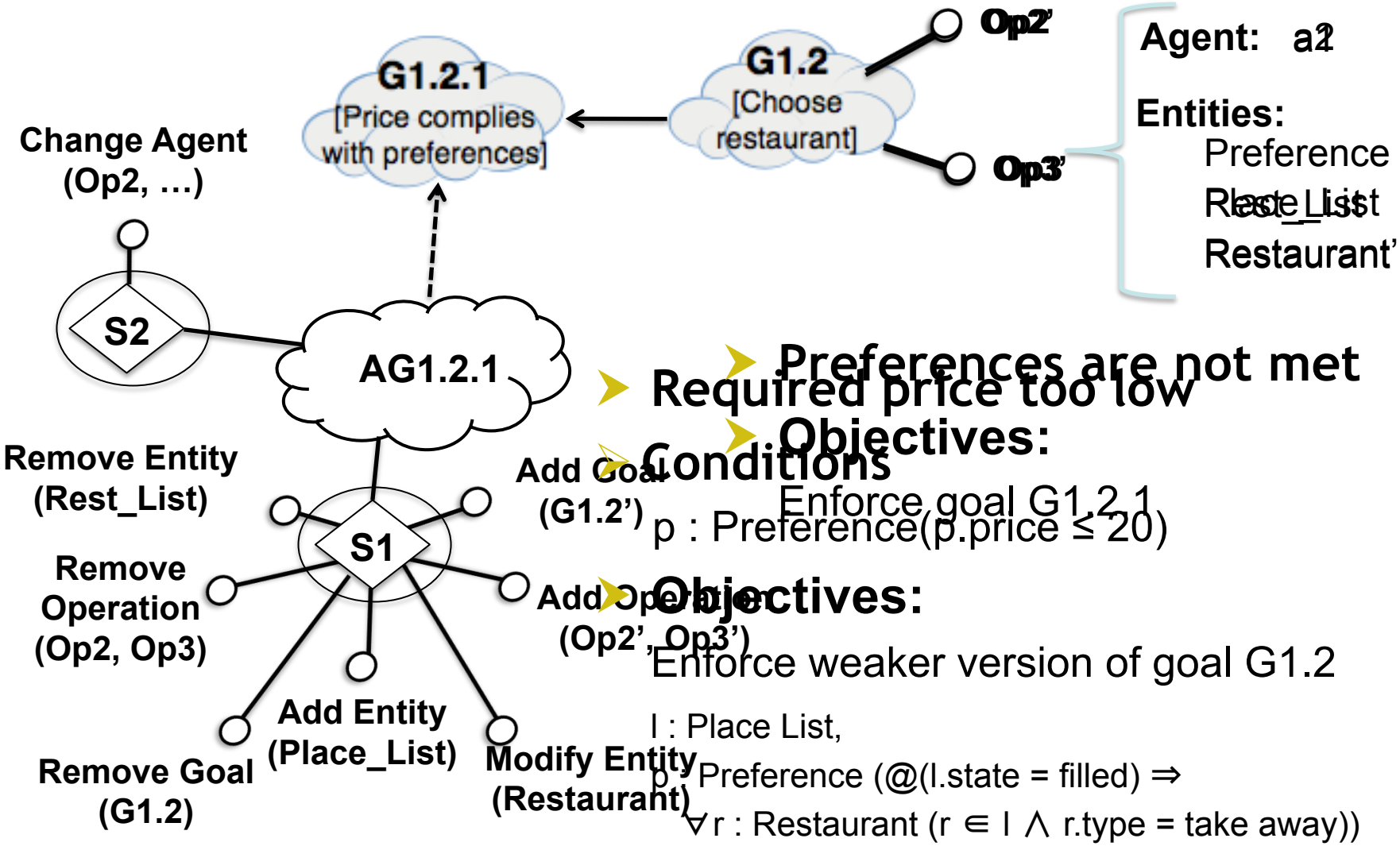
Name: Op2
In/Out: p : Preference, l : RestList
DomPre: l.state = empty
DomPost: l.state = filled
TrigPre: @(p.state = filled)
ReqPost: ∀r ∈ l(r.id ≠ null ∧ r.name ≠ "" ∧ r.address ≠ "")

Name: Op3
In/Out: Rest_List, s : SelRestaurant
DomPre: s.state = selected
DomPost: s.state = selected
TrigPre: @(l.state = filled) ⇒ ∃r : Restaurant(r ∈ l) ∧ s.restaurant = r.id
ReqPost: ∃r : Restaurant ∈ l.restaurants(r.id = s.restaurant))

l : Rest_List

∀r ∈ l(r.id ≠ null ∧ r.name ≠ "" ∧ r.address ≠ "")

Adaptive goals

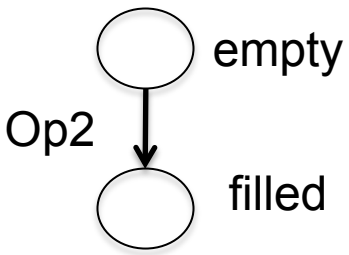


- Preferences are not met
- Required price too low
- Objectives:
  - Enforce goal G1.2.1
- Conditions
  - p : Preference(p.price ≤ 20)
- Objectives:
  - Enforce weaker version of goal G1.2
- Conditions
  - l : Place List,
  - p : Preference (@(l.state = filled) ⇒ ∀r : Restaurant (r ∈ l ∧ r.type = take away))

Functional model (1/4)

1. **Lifecycle of each entity:**
   - state transitions
   - operations causing state transitions
   - precedence rules among operations

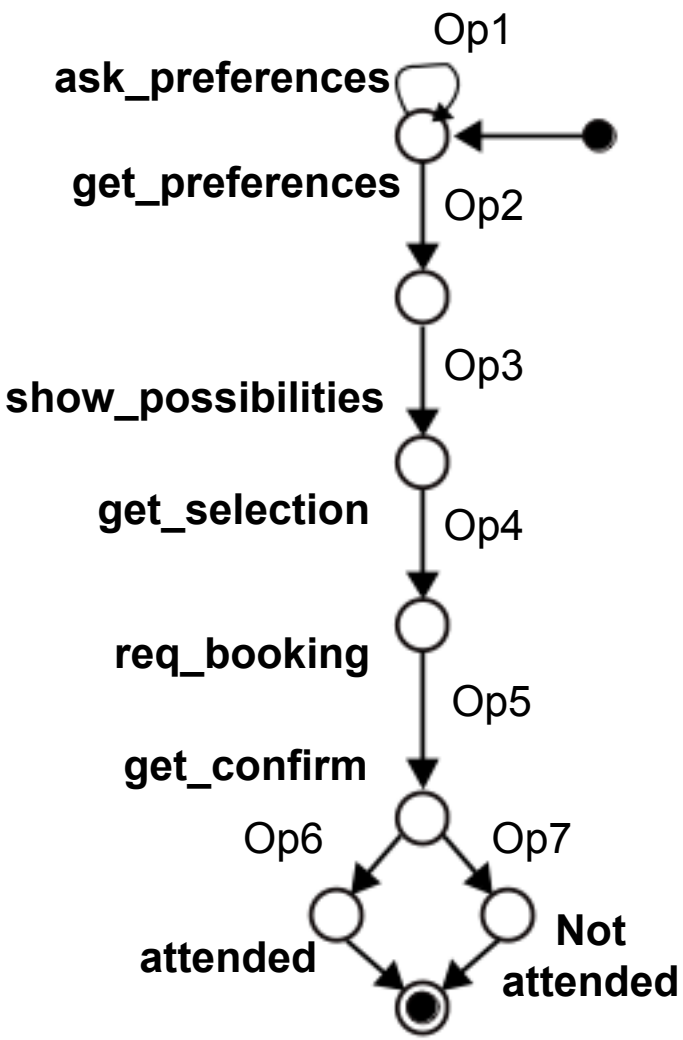
Rest_List



Op1 < Op2
DomPost(Op1) ⇒ DomPre(Op2)
ReqPost(Op1) ⇒ TrigPre(Op2)

Op2 < Op3
DomPost(Op2) ⇒ DomPre(Op3)
ReqPost(Op2) ⇒ TrigPre(Op3)

Functional model (2/4)



2. Generate a possible operation sequence

3. Associate entities with process variables:

   Preference → p

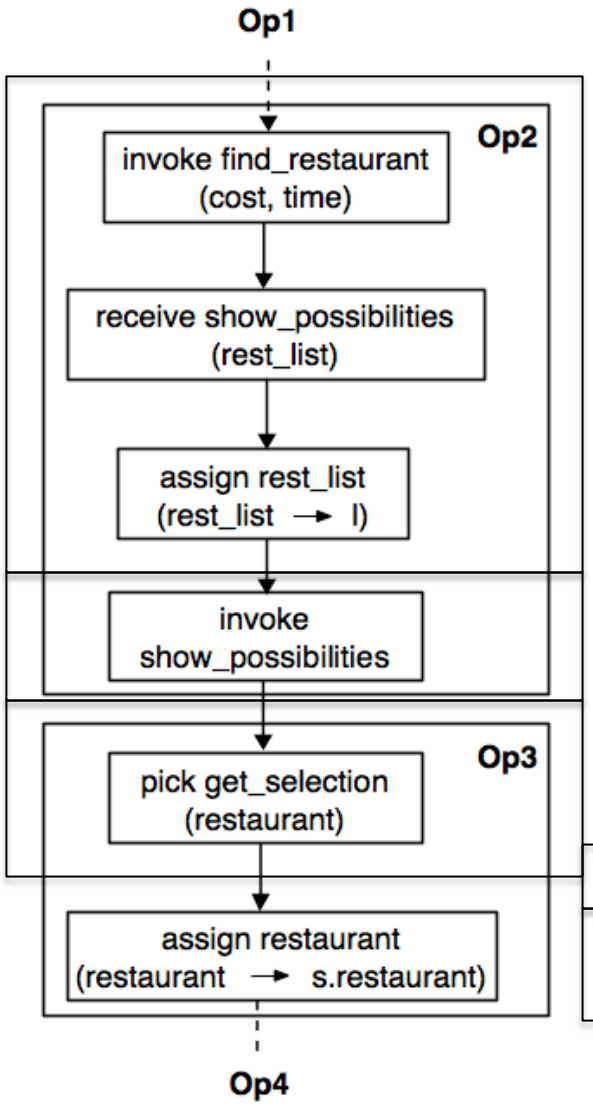
   Rest_List → l

   Sel_Restaurant → s

1. Map agents to partner services:

   a1 → p1, a2 → p2

Functional model 3/4



5. Operations and events are translated into concrete process activities

Triggering Preconditions

- defines an event

Postconditions

- defines an entities

Mapping to an Invoke/Receive activity

Mapping to an Invoke/Reply activity

Mapping to a Pick activity

Mapping to a Invoke(-Receive) activity

Name: Op2
In/Out: p : Rest_Request, l : Rest_List
DomPre: l.state = empty
DomPost: l.state = filled
TrigPre: show_possibilities
ReqPost: $\exists r \in l(r.id \neq null \wedge r.name \neq "" \wedge r.address \neq "")$

Supervision Model

- Supervision directives
  - Monitoring directive:
    - decide when adaptation must be carried out
    - translation of goals in the language of a monitor
      - Dynamo: FOL properties
      - Albert: LTL properties
  - Adaptation directives:
    - Scope
    - Conflicts with other strategies
    - Actions:
      - change the process instance/process definition
      - change the goal model

Conflicting strategies

- Conflicts Types
  1. Applied on the same goal with overlapping conditions
  2. Applied on the benefit of conflicting goals
     - the conflict is specified in the goal model
  3. May generate incoherent processes
- Conflicts Resolution
  1. The strongest strategy is triggered
     - strategies are defined in a hierarchical way
  2. The strategy associated with the most critical goal is triggered

Example (1/2)

S1

- **Conditions**

  p : Preference(p.price ≤ 20)

- **Objectives:**

  Enforce weaker version of goal G1.2

  l : Place List,

  p : Preference (@(l.state = filled) ⇒

  ∀r : Restaurant (r ∈ l ∧ r.type = take away))

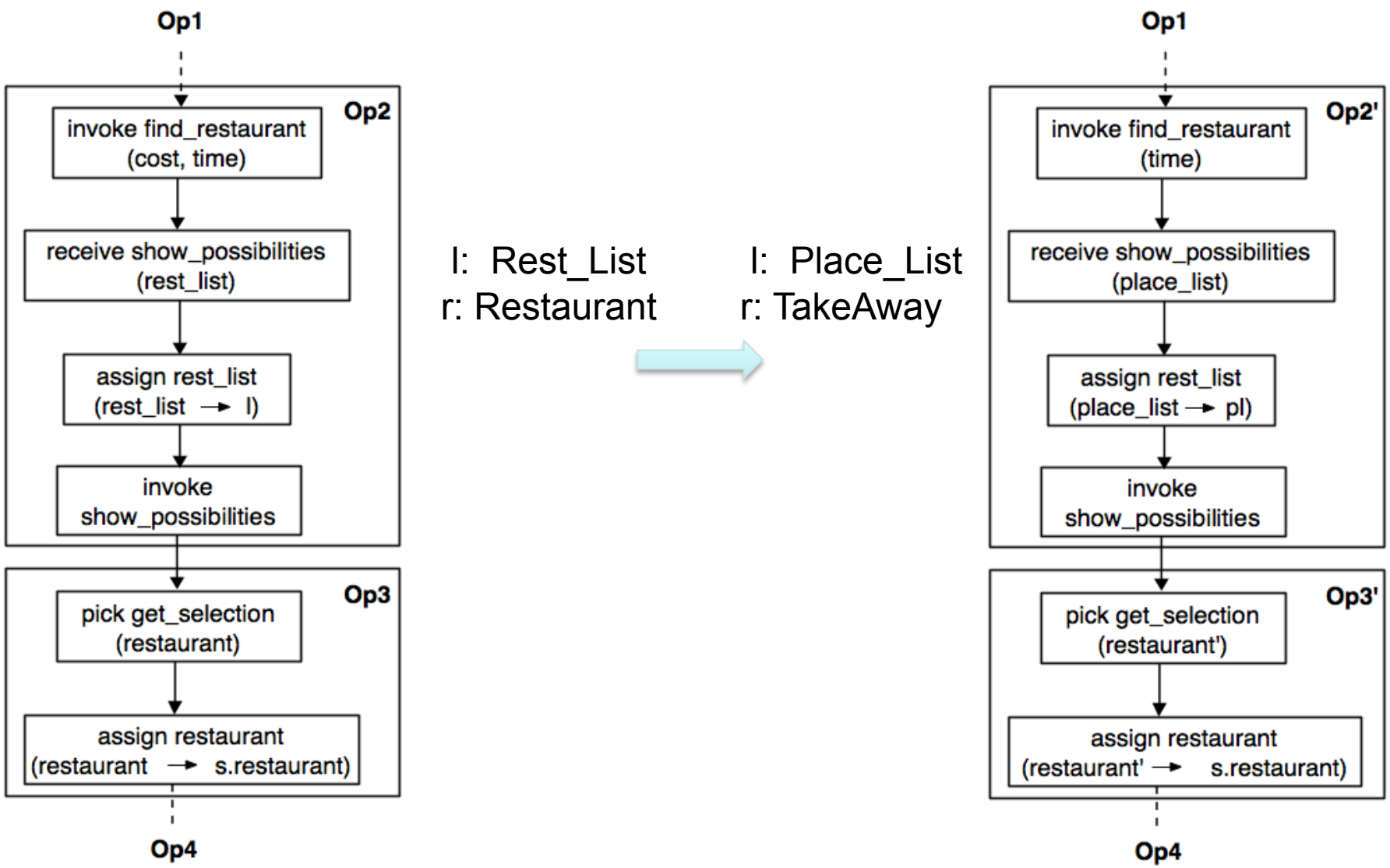
S2

- **Objectives:**

  Enforce goal G1.2.1

AG1.2.1		
Strategy	**Scope**	**Conflicts**
S1	instance	S2
	process	
S2	instance	S1

Example (2/2)


Op1
Op2
invoke find_restaurant
(cost, time)
receive show_possibilities
(rest_list)
assign rest_list
(rest_list → l)
invoke
show_possibilities
Op3
pick get_selection
(restaurant)
assign restaurant
(restaurant → s.restaurant)
Op4
l: Rest_List
r: Restaurant
l: Place_List
r: TakeAway
Op1
Op2'
invoke find_restaurant
(time)
receive show_possibilities
(place_list)
assign rest_list
(place_list → pl)
invoke
show_possibilities
Op3'
pick get_selection
(restaurant')
assign restaurant
(restaurant' → s.restaurant)
Op4

Conclusions and Future Work

- **Conclusions**
  - An approach to deal with self-adaptive BPEL processes from requirements elicitation
    - Adaptive goals address adaptation and changes at goal level
- **(Parallel/)Future Work**
  - Fuzzy goals to have a notion of satisfaction level for requirements, and trigger adaptation accordingly
  - Design tools to model realistic service compositions
  - Improve the conflict resolution methodology

Questions