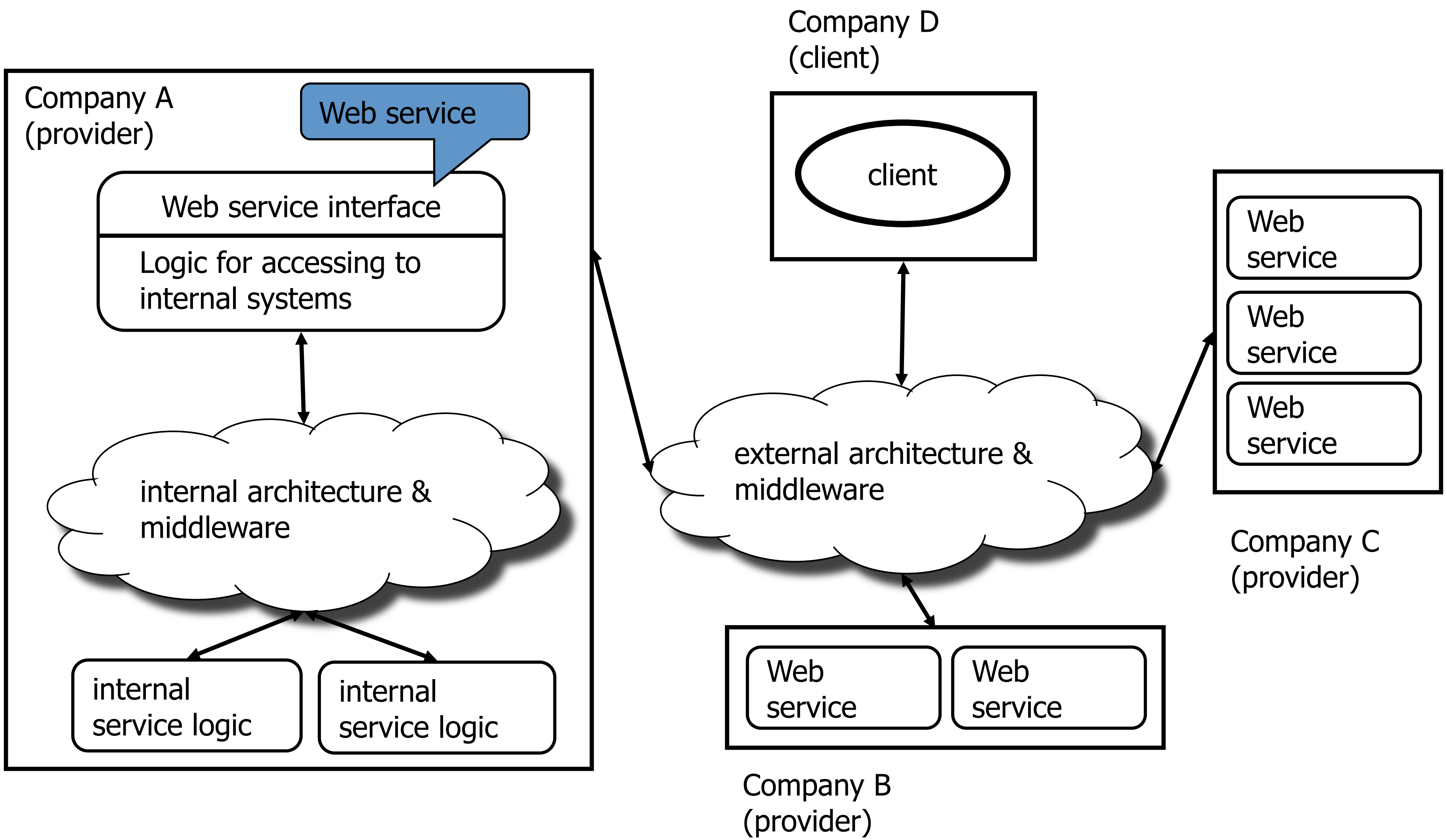
Automated Service Composition and Synthesis

Fabio Patrizi
SAPIENZA – Università di Roma
patrizi@dis.uniroma1.it
www.dis.uniroma1.it/~patrizi

What are services?

- Given, modular, decoupled SW blocks
- Typically, non terminating
- Common communication layer
- Intended to serve (human or sw) clients
- E.g.: travel agency, book seller, car rental

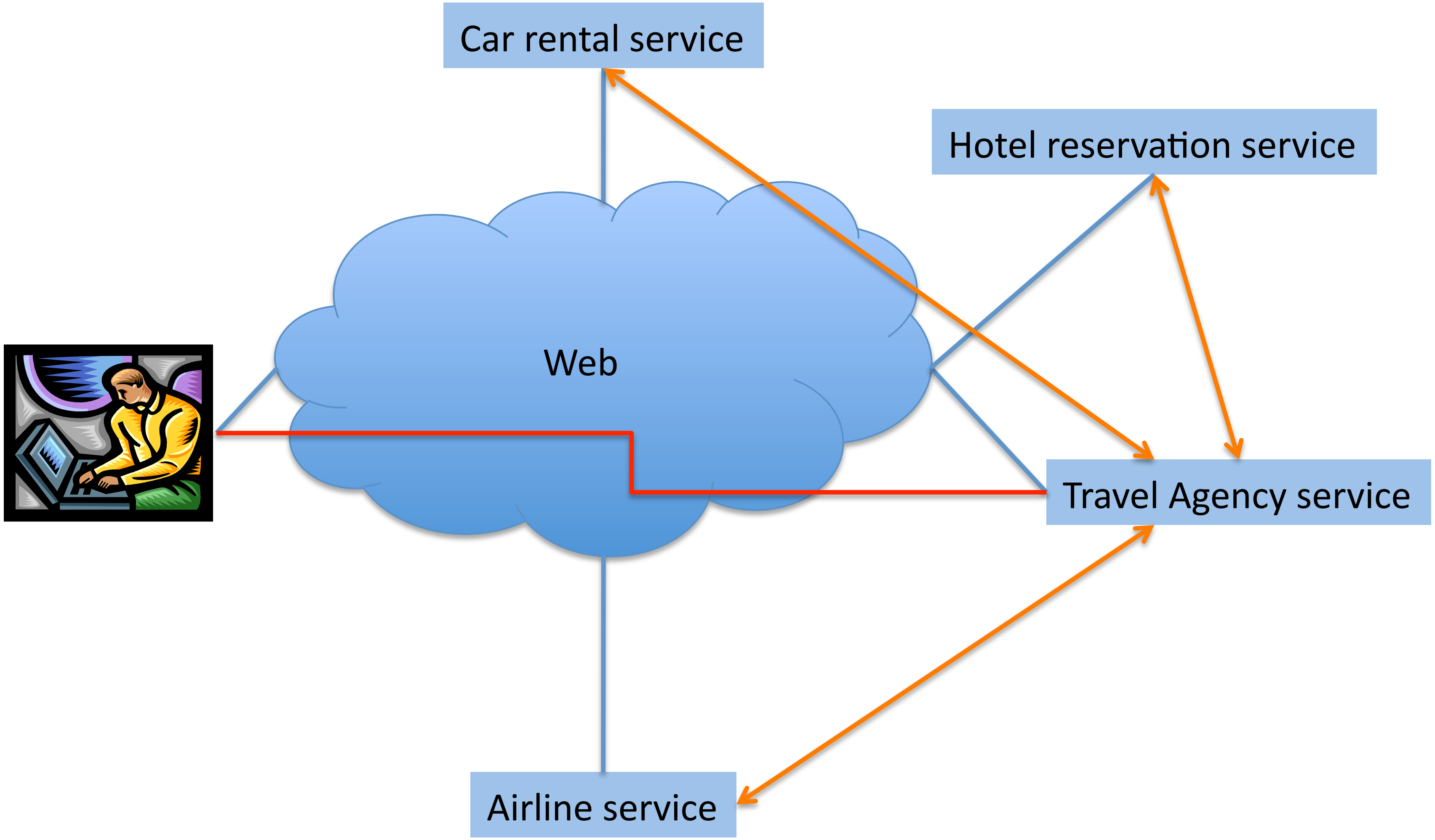
What are services? (2)



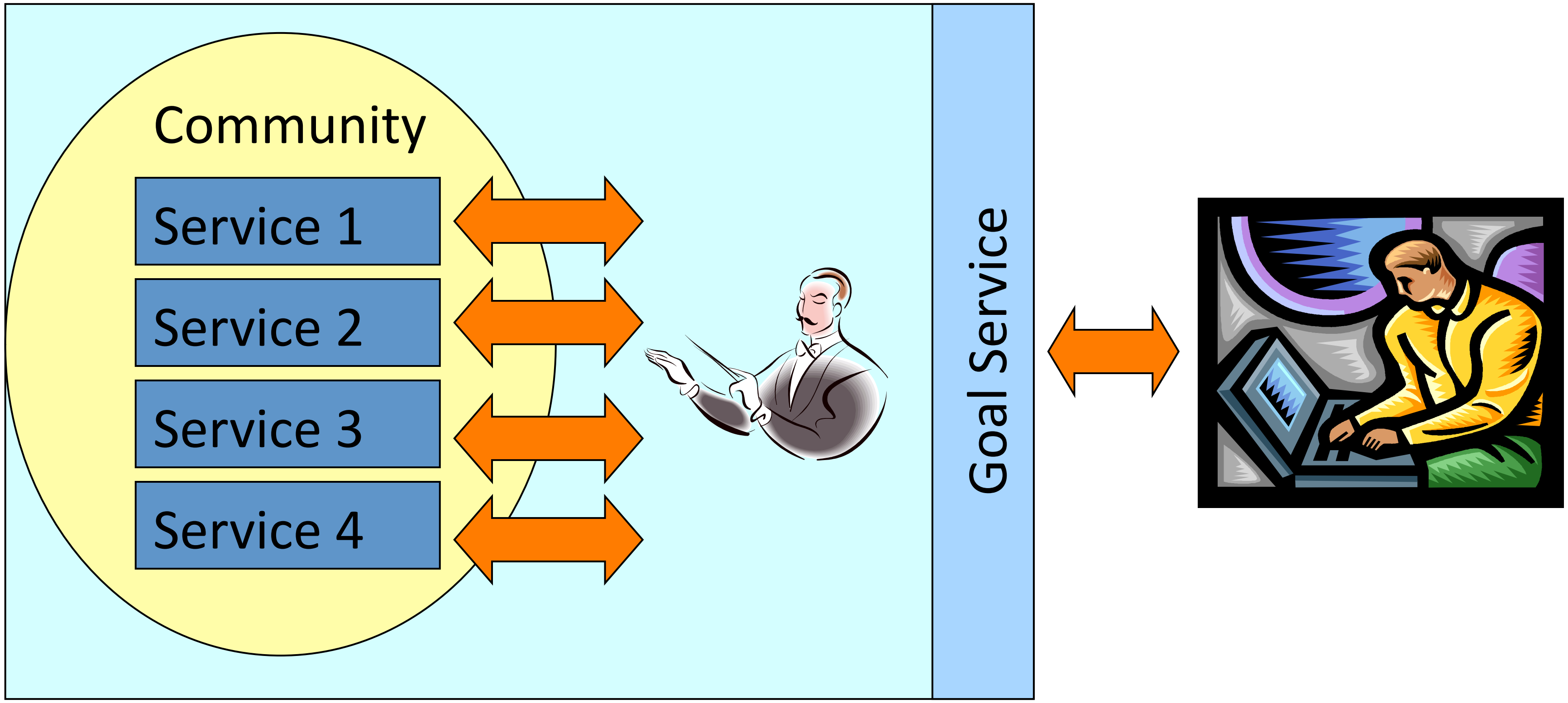
Technology

- Programs written in any language (Java, C++,…)
- Export a description (typically, WSDL: offered operations only)
- Common protocol (typically, SOAP over HTTP)
- Usually stateless, but we assume stateful

Composability



Service Composition



The Composition Problem

- Instance:
  - A set of available services
  - A (non available) goal service

- Solution:
  - An orchestrator which coordinates, through delegation, the available services so as to mimic the goal service

- Examples of *composed services*:
  - Expedia: orchestrates car rental, hotel reservation, etc.
  - Amazon: orchestrates book sellers

The Framework

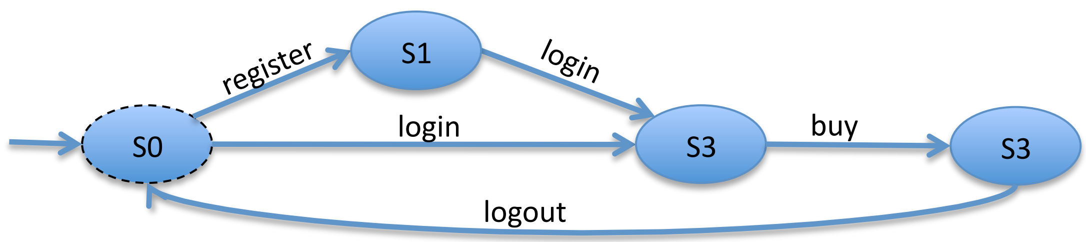
- A service (abstract) model
- A notion of solution (or orchestrator)

The Roman* Model

(* As referred to by R. Hull@SIGMOD'04)

Service Conversational Model:

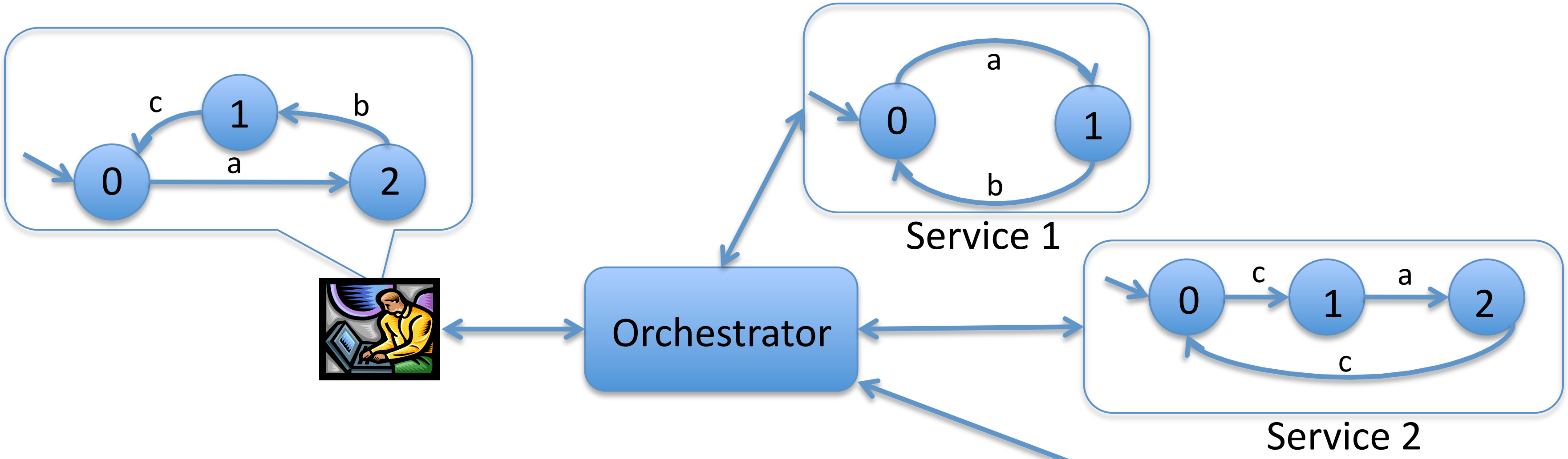
- Stateful behavior abstracted as a finite-state TS
- Transition labels: atomic operations (or actions)
- Final states: computation stops safely



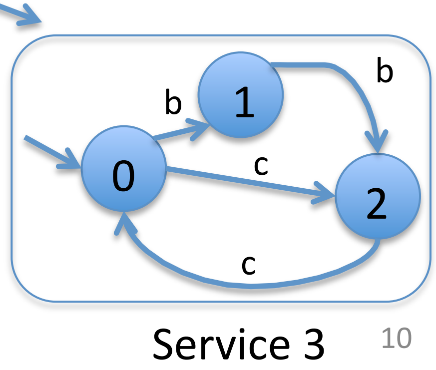
Very high-level abstraction!

Orchestrators

Orchestrator: from histories and current request to service indices
Composition: good orchestrator, i.e., consistent delegations



a	b	c	a	b	c	a	b	c	a	b	c	a	b	c
1	1	2	2	3	2	1	1	2	1	3	3	2	1	2



Advanced form of plan!

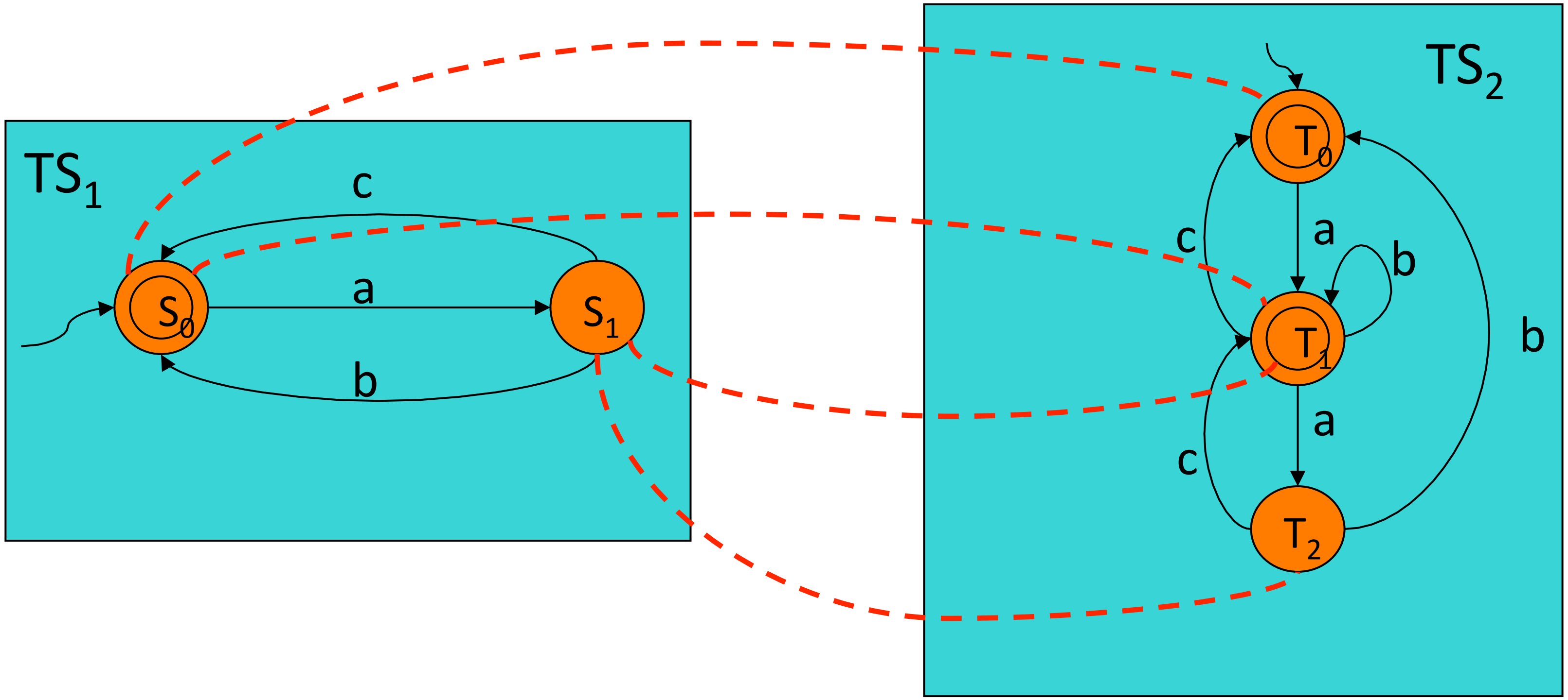
Orchestrators (2)

(Because everything is deterministic, action requests and delegations enable state reconstruction)

Req	a	b	c	a	b	c	a	b	c	a	b	c	a	b	c	...
Del	1	1	2	2	3	2	1	1	2	1	3	3	2	1	2	...
Goal	0	2	1	0	2	1	0	2	1	0	2	1	0	2	1	0
Serv.1	0	1	0	0	0	0	0	1	0	0	1	1	1	1	0	0
Serv.2	0	0	0	1	2	2	0	0	0	1	1	1	1	2	2	0
Serv.3	0	0	0	0	0	1	1	1	1	1	1	2	0	0	0	0

History

Simulation Relation (intuition)



(TS$_2$ behaviors “include” TS$_1$’s)

Simulation is over a possibly infinite horizon!

Formally

(Co-inductive definition: no base case)

Given TS_1 and TS_2

$s_1 \preccurlyeq s_2$ iff:

1. "s_1 final" implies "s_2 final"
2. For each transition $s_1 \rightarrow^a s'_1$ in TS_1, there exists a transition $s_2 \rightarrow^a s'_2$ in TS_2 s.t. $s'_1 \preccurlyeq s'_2$

Computing a Simulation Relation

Algorithm ComputeSimulationRelation
Input: transition system TS_S = < A, S, S^0, δ_S, F_S> and
transition system TS_T = < A, T, T^0, δ_T, F_T>
Output: the **simulated-by** relation (the largest simulation)

Body

$R = S \times T$

$R' = S \times T - \{(s,t) \mid s \in F_S \wedge \neg(t \in F_T)\}$

while ($R \neq R'$) {

$R := R'$

$R' := R' - \{(s,t) \mid \exists\, s',a.\ s \rightarrow_a s' \wedge \neg\exists\, t'.\ t \rightarrow_a t' \wedge (s',t') \in R'\}$

}

return R'

Ydob

- Fixpoint computation
- Time Cost: $O(n^4)$

Orchestrators, formally

Community TS: asynchronous product of available services

An orchestrator is a witness of:

the Community TS simulates the goal service

The composition problem can be reduced to searching for a simulation of the target service by the Community TS[Berardi,Cheikh,DeGiacomo,P@IJFCS ('08)]

Complexity

Finding an orchestrator in the Roman Model is an EXPTIME-complete problem

- Membership:
  - Reduction to PDL-SAT

  [Berardi,Calvanese,De Giacomo,Lenzerini,Mecella@ICSOC03]
- Hardness

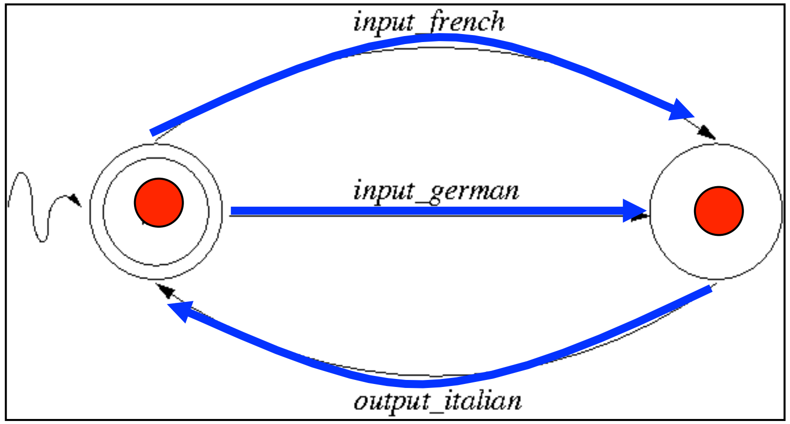
  [Muscholl,Walukiewicz@FoSSaCS07]:
  - Reduction from existence of an infinite computation in LB ATM (EXPTIME-hard)

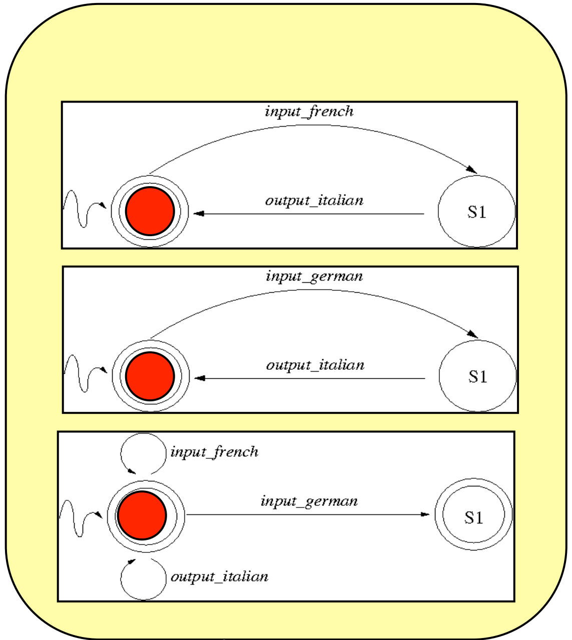
Computing Orchestrators

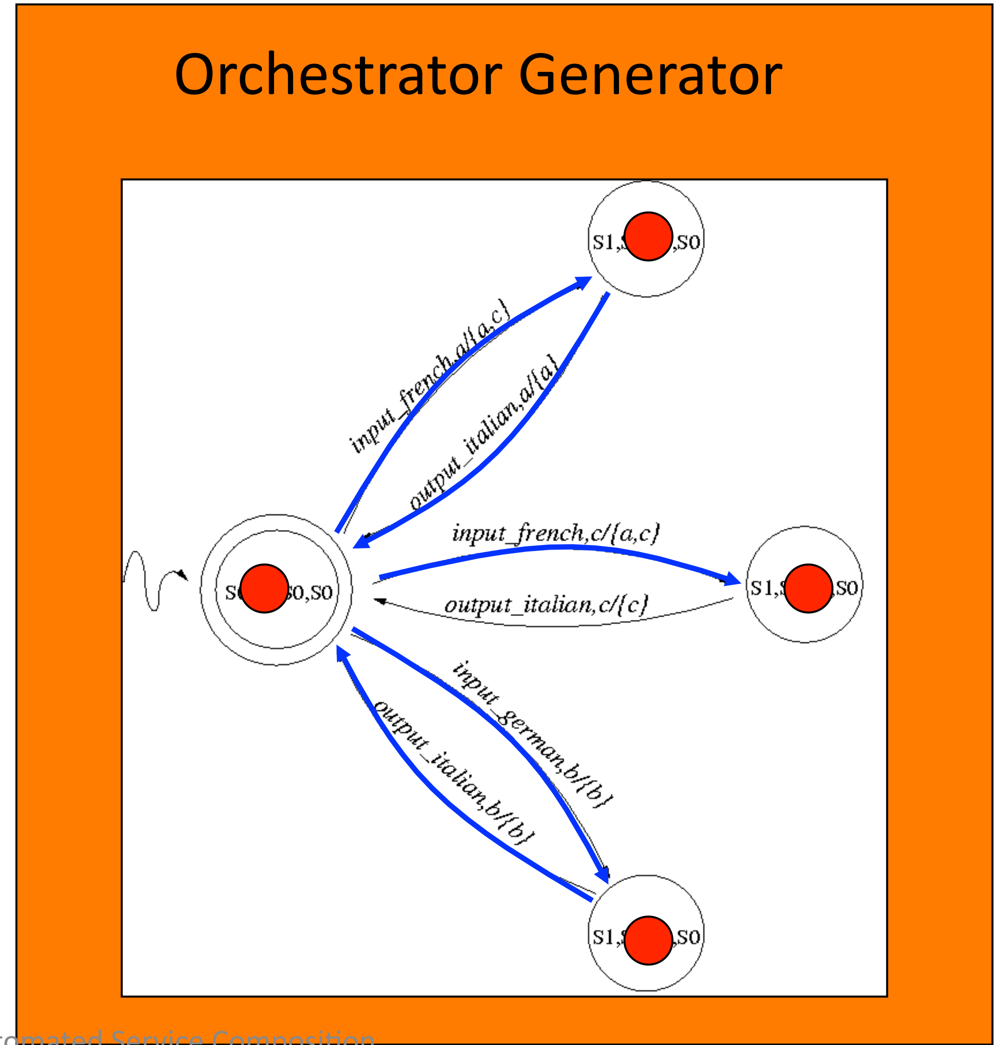
- Orchestrators can be seen as (possibly infinite) state machines
- In general, there may exist an infinite # of orchestrators
- Th.: *if an orchestrator exists, then there exists one which is finite*

  [Berardi,Calvanese,DeGiacomo,Lenzerini,Mecella@ICSOC03]
- A finite structure (Orchestrator Generator) can be computed that represents all, even infinite, orchestrators[Berardi,Cheikh,DeGiacomo,P@IJFCS]

Orchestrator Generators







Computing Orchestrators (2)

Simulation-based approach (Orch Gen):

Based on largest simulation computation

Optimal wrt worst-case time complexity

[Berardi,Cheikh,DeGiacomo,P@IJFCS]

Provides flexible solutions [Sardina,P,De Giacomo@KR08]

The simulation can be computed directly or a game-based approach can be adopted (see next part)

Symbolic MC technology available!

On Service Abstraction

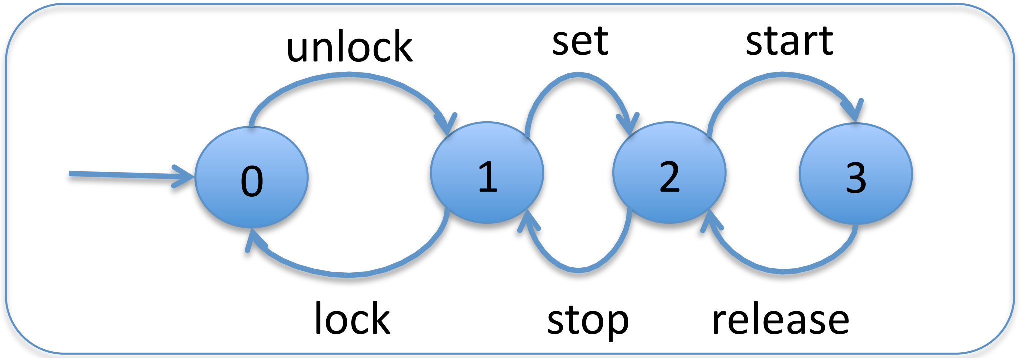
- Services can be used to abstract a variety of systems, not only web services
- In general, entities that offer services to external clients can be seen as services
- We think of a service as the abstraction of a device, behavior or agent internal logic

On Service Actions

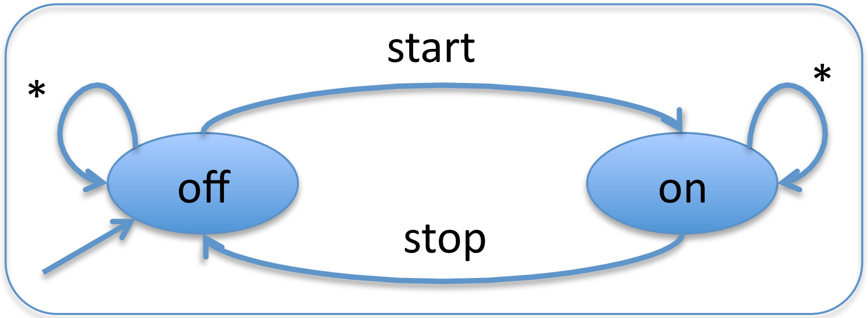
- So far, we considered actions that affect only service states
- In general, service actions:
  - Affect available service state
  - Change the state of the domain that the service acts in

On Service Actions (2)

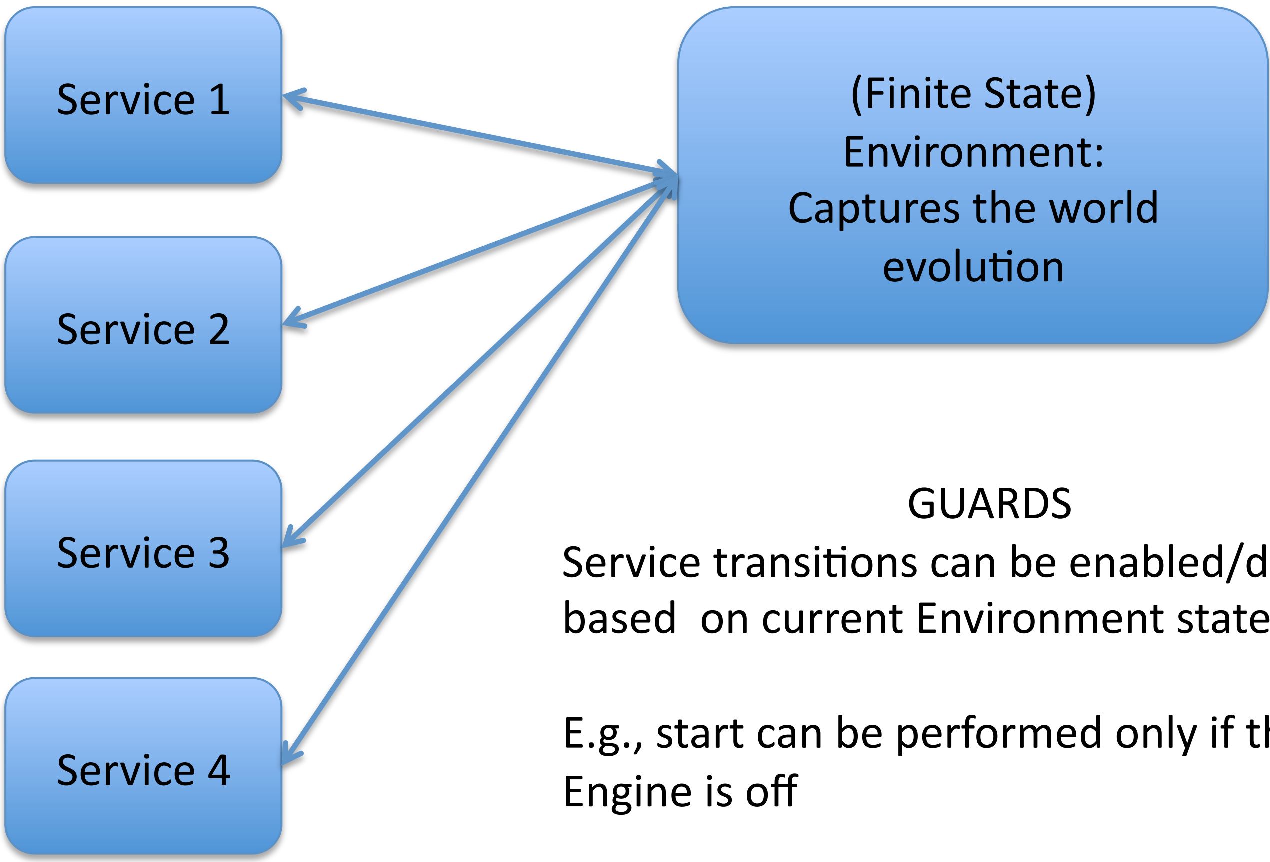
Ignition service



Car engine



Environment



GUARDS

Service transitions can be enabled/disabled based on current Environment state

E.g., start can be performed only if the Engine is off

Action Compatibility

- So far, only matching actions are considered ``compatible''
- We can explicitly define an
  <u>Action-Compatibility Relation</u>

  Comp(a,a',⟨t,s1,…,sn,db⟩)

  When the target service is in state t, the available services in ⟨s1,…,sn⟩ and the environment, if present, in db: *action a' can replace a*
- Straightforward adaptation of both:
  - Simulation relation definition
  - Algorithm ComputeSimulationRelation

Extensions

Variants of this problem:

- Nondeterministic available services
- Partially observable available services
- Distributed orchestrator
- Data-aware services

Further (composition) problems:

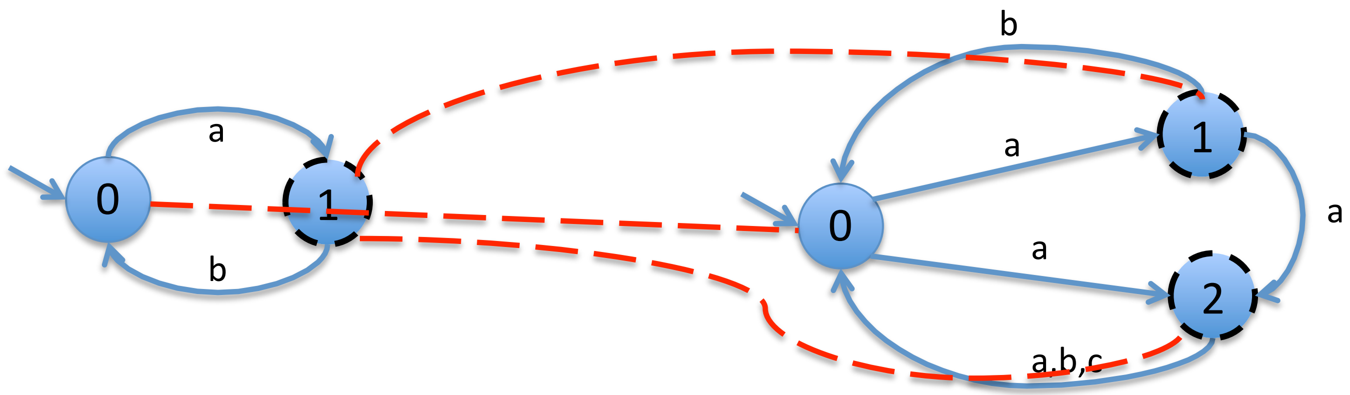
- Multi-target composition
- Agent planning programs

ND available services

- Nondeterminism: from partial knowledge or very high-level abstraction
- Goal services still deterministic (we know what we want!)
- ``Conditional'' form of composition
- New notion of simulation needed, in order to define orchestrators

ND-Simulation relation and orchestrators

- Idea: preserve simulation regardless of outcomes of available service transitions



- An ND-orchestrator is a witness of:

the Community TS ND-simulates the goal service

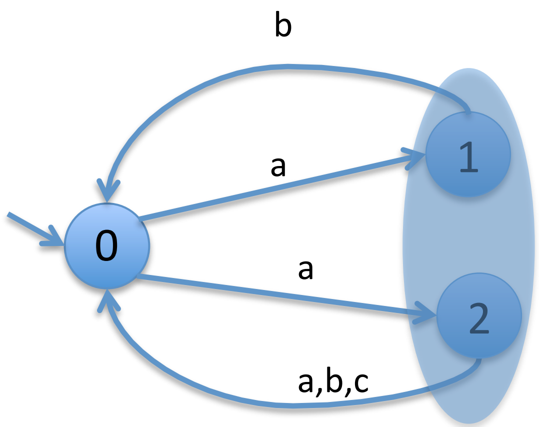
Composition with ND services

Essentially as complex as when services are deterministic (EXPTIME-complete)

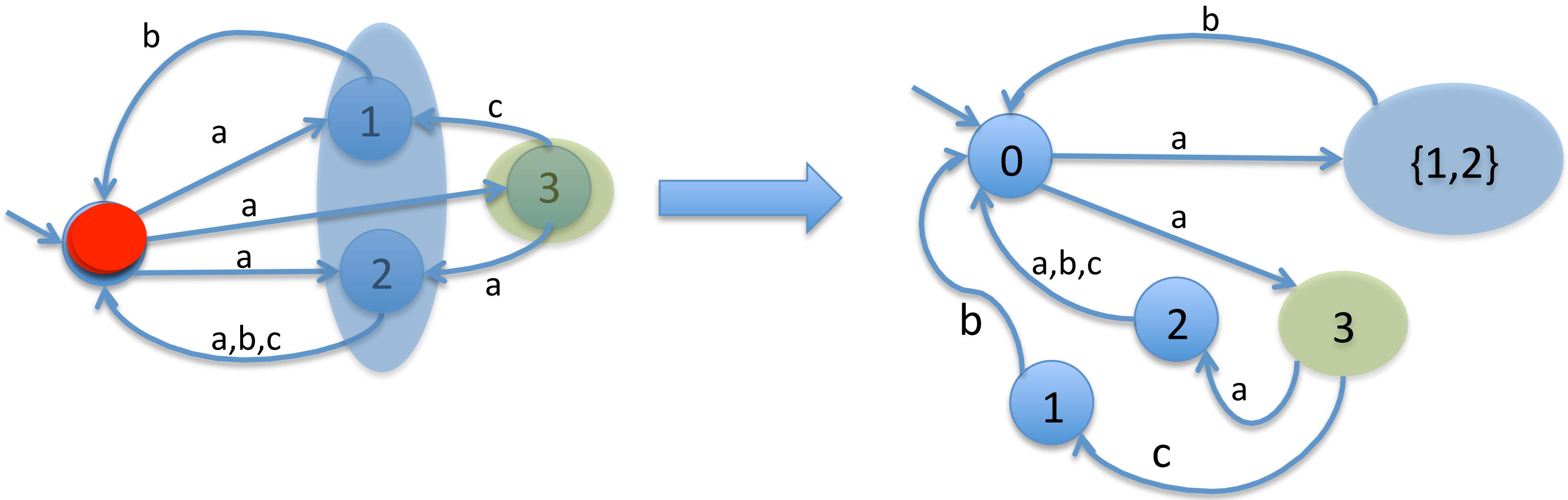
Remark: at each step, after a transition, we need to know the state that each service is in (Full observability)

Partially observable services

- ``Conformant'' (i.e., PO) form of composition [DeGiacomo,DeMasellis,P@ICAPS09]:
  - ND available services
  - There might be undistinguishable states



Partially observable services (2)

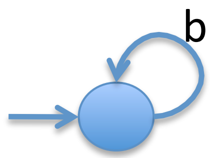


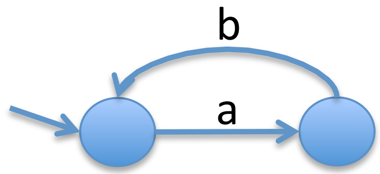
In general, exponential growth!

Orchestrators under partial observability

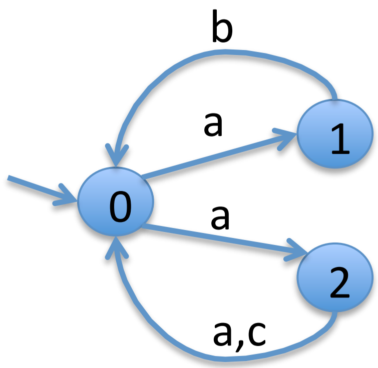
- Orchestrators rely only on observations, not on actual current states
- Function of observed histories (and current request)

An example





Goal Service



Inability to distinguish between states 1 and 2 prevents goal service composition!

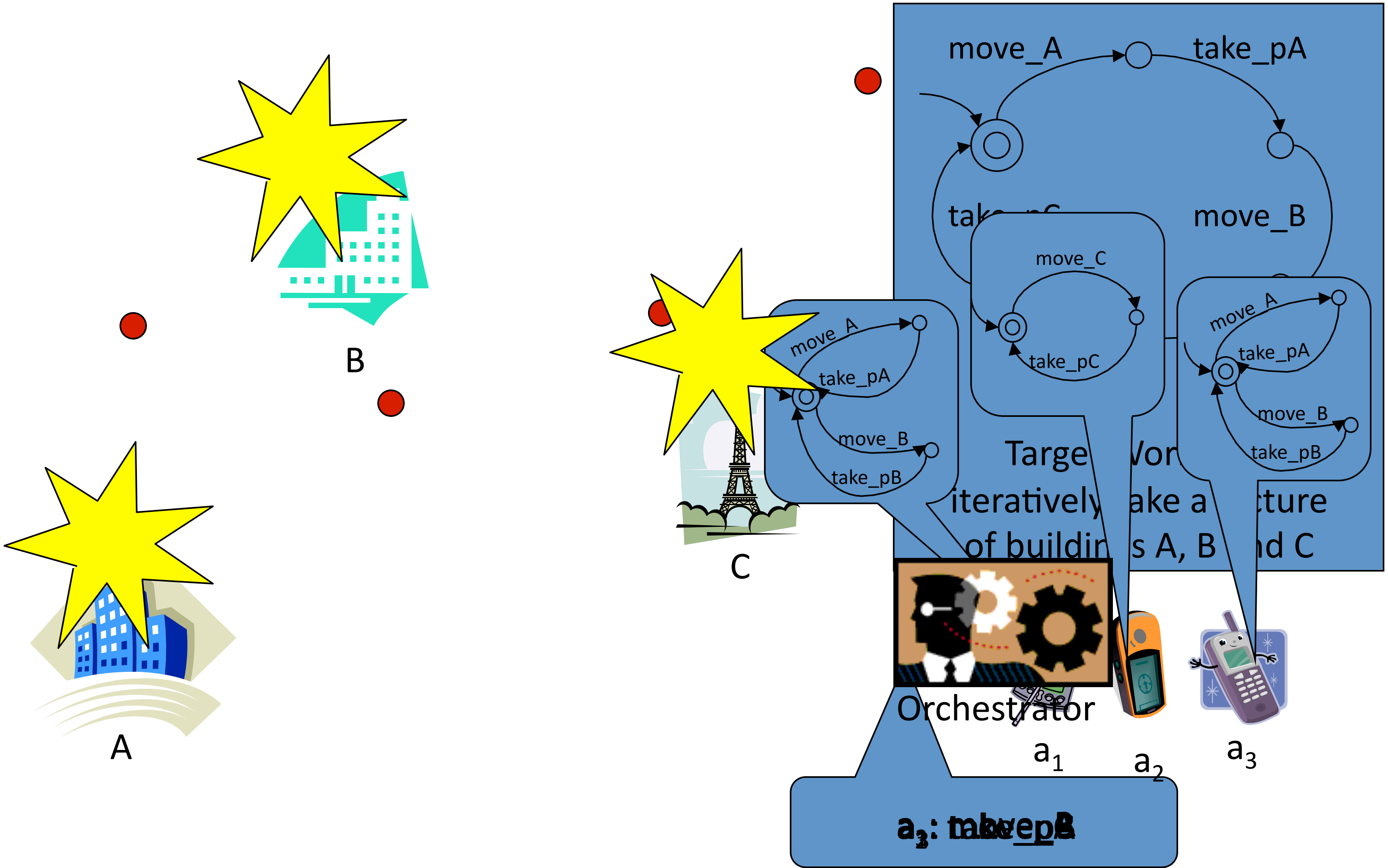
Building Orchestrators under PO

- Approach based on belief construction
  1. Transform all PO services into FO ones (exponential in # of states)
  2. Compute the orchestrator as in the ND case
- Complexity:
  - EXPTIME-complete
  - (Singly) Exponential in both # of services and their size

Distributed Orchestrators

- What if a central coordinating entity is not conceivable?
  [Sardina,P,DeGiacomo@AAAI07;DeGiacomo,deLeoni,Mecella,P@ICWS07]

Example

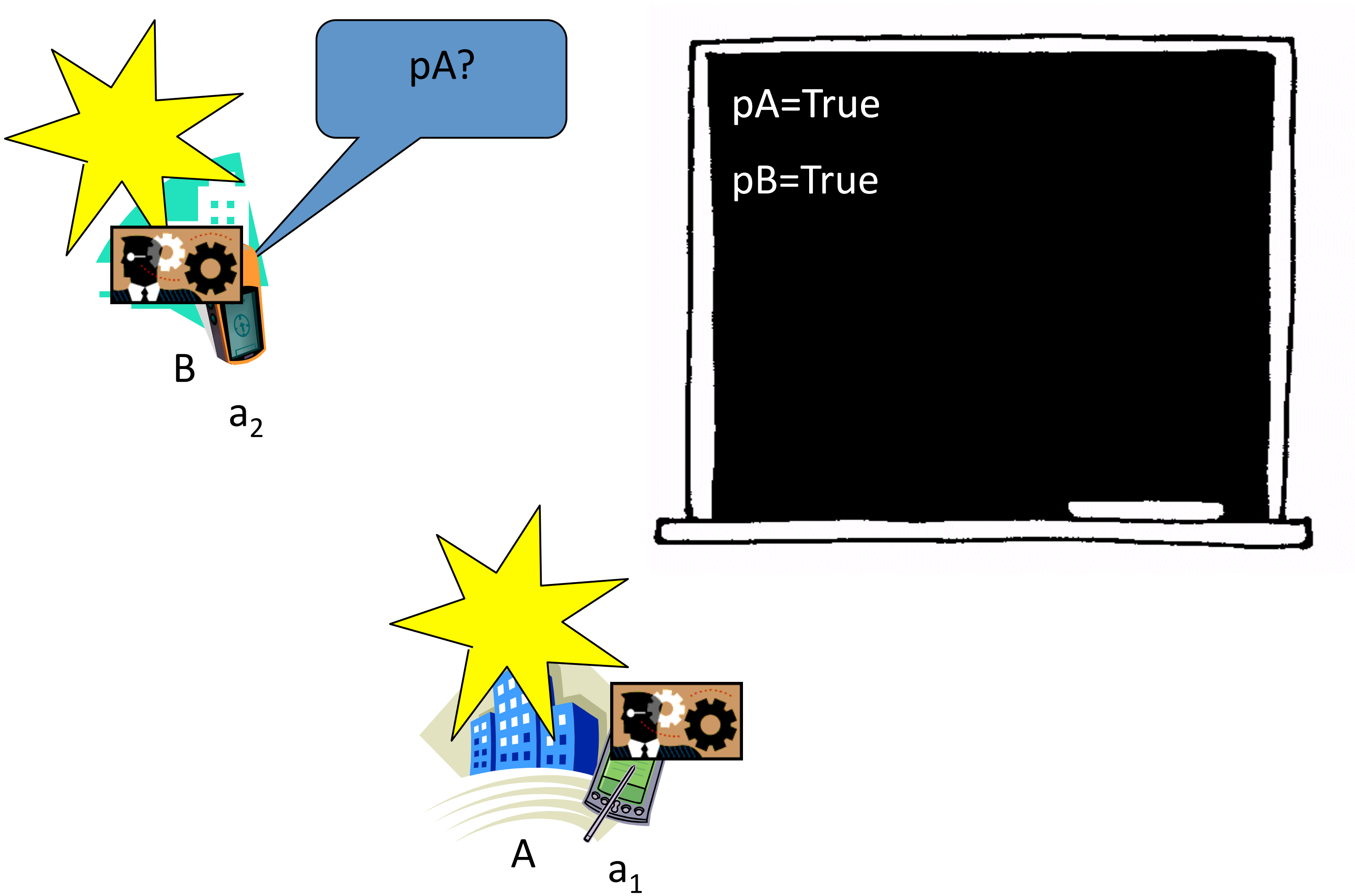


B
A
C
move_A
take_pA
move_B
move_C
take_pC
take_pB
Orchestrator
a_1
a_2
a_3

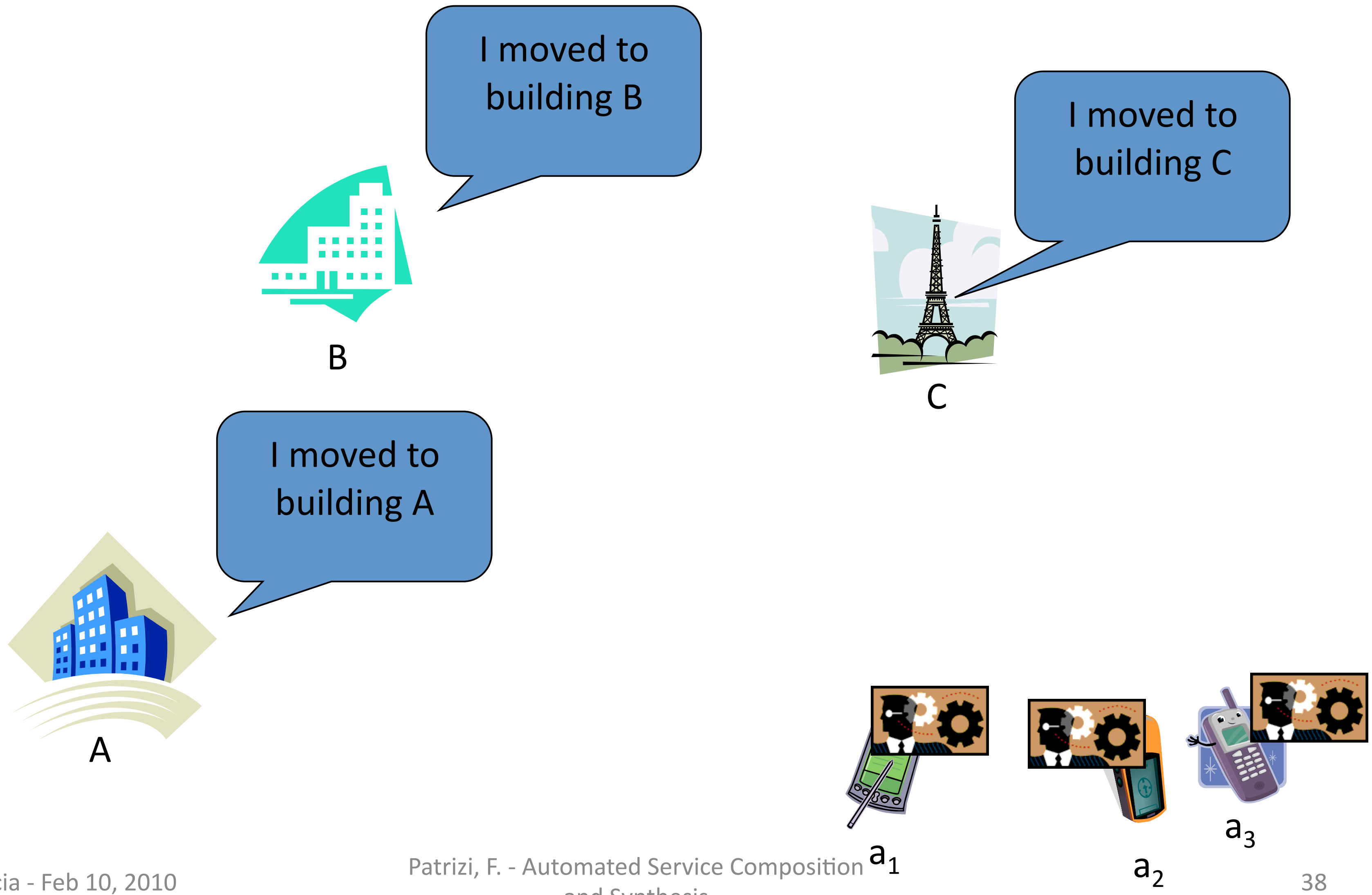
Local Orchestrators

- Use a local orchestrator for each device
- Local Orchestrators exchange messages
- OBJECTIVE: Local orchestrators behave as if they were, as a whole, centralized
- Need for a (distributed) shared memory (blackboard), modeled as Environment
- Assumption: local orchestrators have FO on their service state

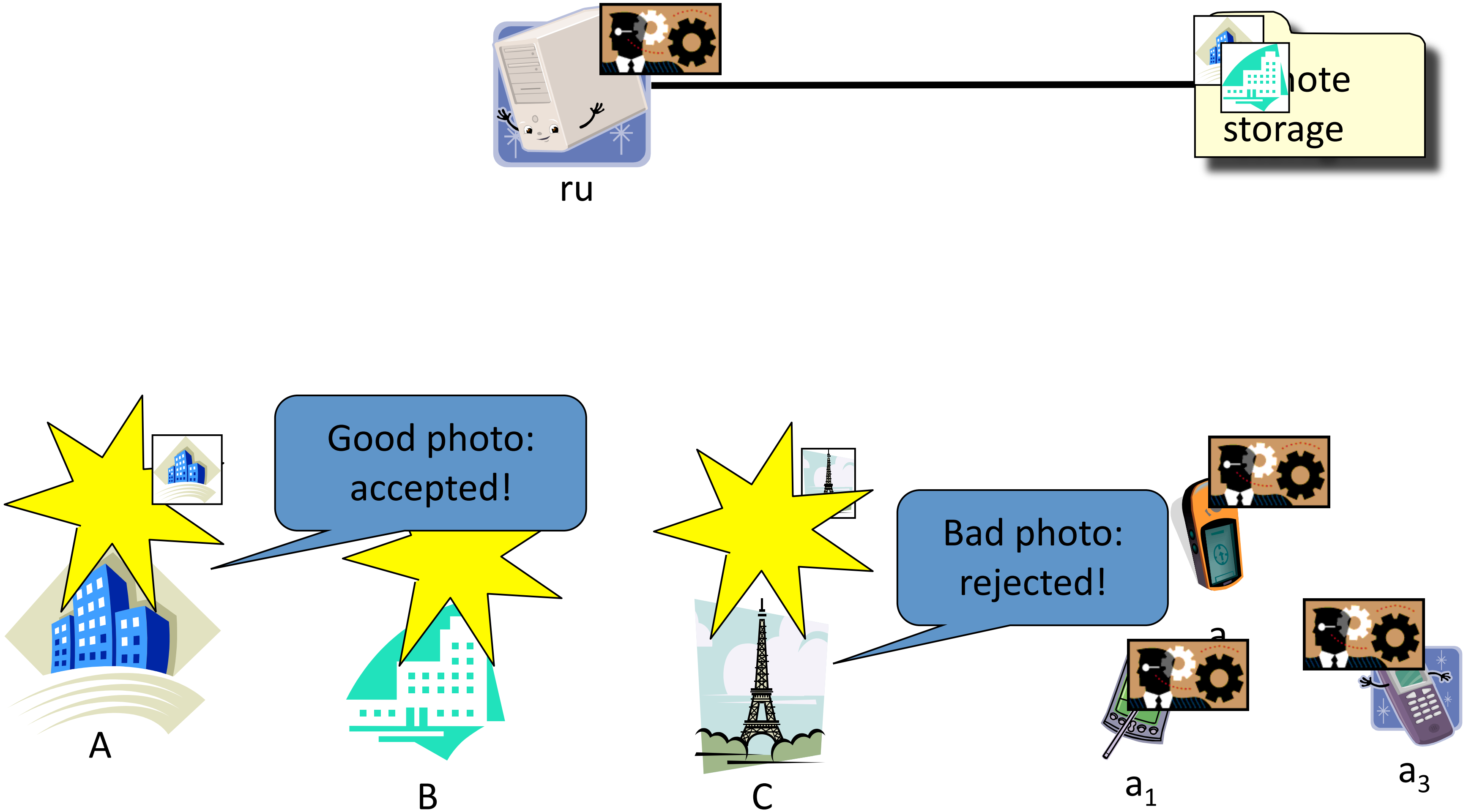
Blackboard



Message Broadcasting



Example



ru
ote
storage
Good photo: accepted!
Bad photo: rejected!
A
B
C
a1
a3

Computing Local Orchestrators

Th.: *A centralized Orch exists iff Local ones exist*
[Sardina,P,DeGiacomo@AAAI07]

So:

1. Build the centralized Orch (w/ any technique)
2. Split it into local ones (PTIME in C Orch size)
3. Attach each local orchestrator to a service

Multiple-Target Composition

- Generalization of Composition
  [Sardina,DeGiacomo@ICAPS08]
  – Realize a set of goal services, to be executed concurrently, under a fair schedule
  – Available services can switch the goal service they are realizing

Multiple-Target Composition (2)

Goal Services

Available Services

Nondet:
Actions can be assigned to goal services after execution

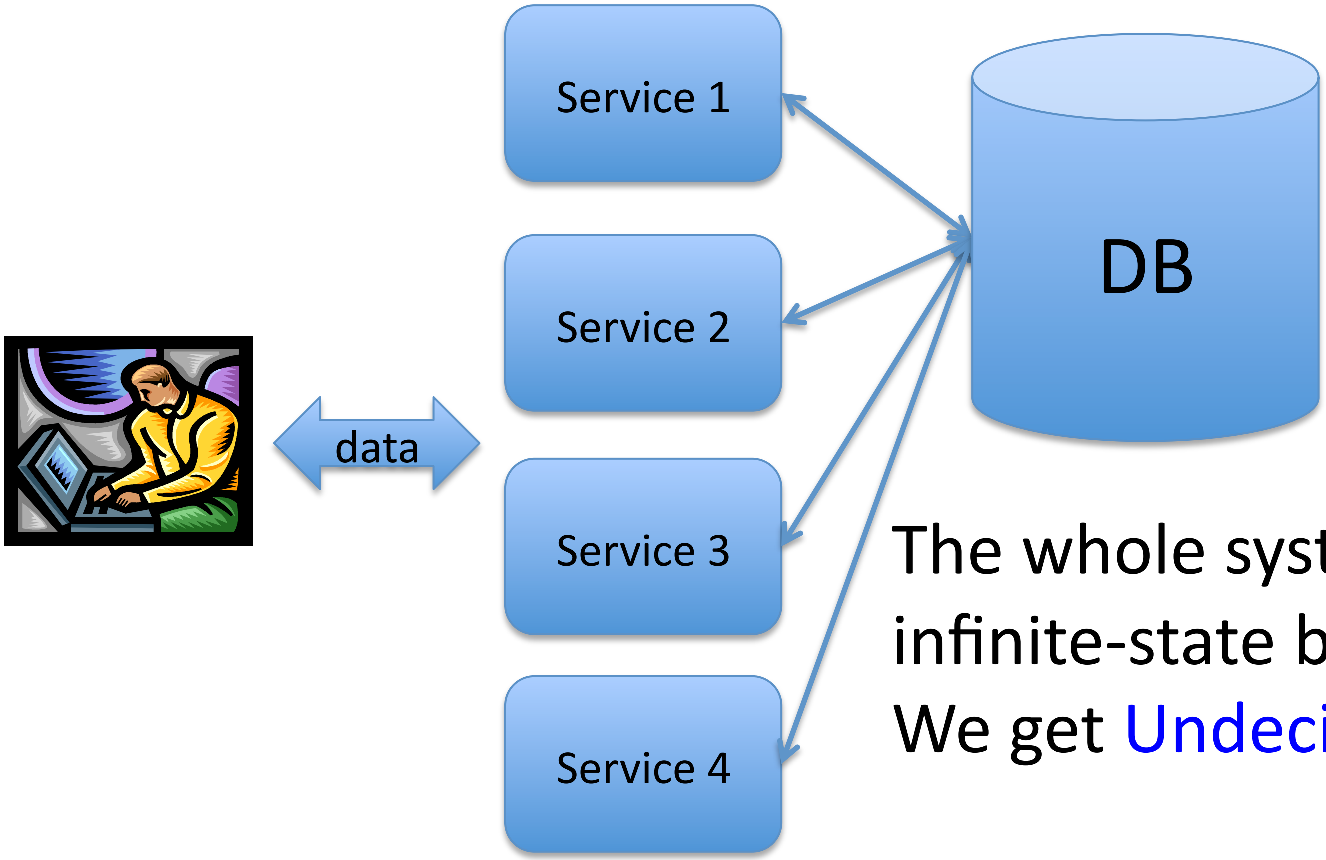
Solving Service Composition Problems

- Previous problems can be reduced to finite-state, ND composition under Nondeterminism and Full Observability
- Approaches based on LTL synthesis have been adopted (we see a generalization in next part)
- The cost increases together w/ the ability to capture richer scenarios
- All problems are in the same complexity class
- In fact, all EXPTIME-complete

Data-Aware Services

- So far, we considered very high level action abstractions, but:
  - Agents may need to exchange messages (e.g., position, battery level,…)
  - Web services often take input messages(e.g., users subscribe) and return output messages (e.g., pricelist)
- Services may need data manipulation
- Topic of interest in DB research, too

Web Service Example



The whole system shows an infinite-state behavior

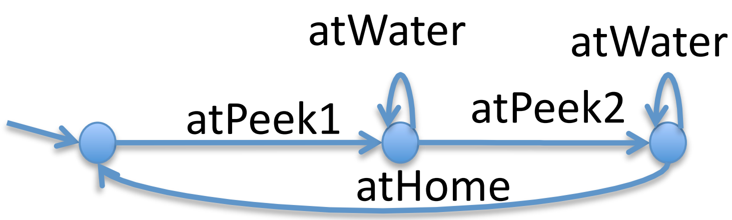
We get Undecidability!

Data-Aware Services (2)

- The presence of data is probably the major obstacle in Service Science
- Results essentially based on data-abstraction (reduction to symbolic data):
  - [Deutsch,Sui,Vianu@JCCS-07]: (Temporal) Verification of web applications
  - [Deutsch,Hull,P,Vianu@ICDT09]: Verification of data-centric Business Processes
  - [Berardi,Calvanese,DeGiacomo,Hull,Mecella@VLDB05]: PDL-based Composition w/ data
  - [P,DeGiacomo@IIWeb09]: Generalization of the notion of Simulation in the presence of data

Agent Planning Programs

- High-level programs built from goals
- To be executed in a dynamic domain
- Branches represent goal selections

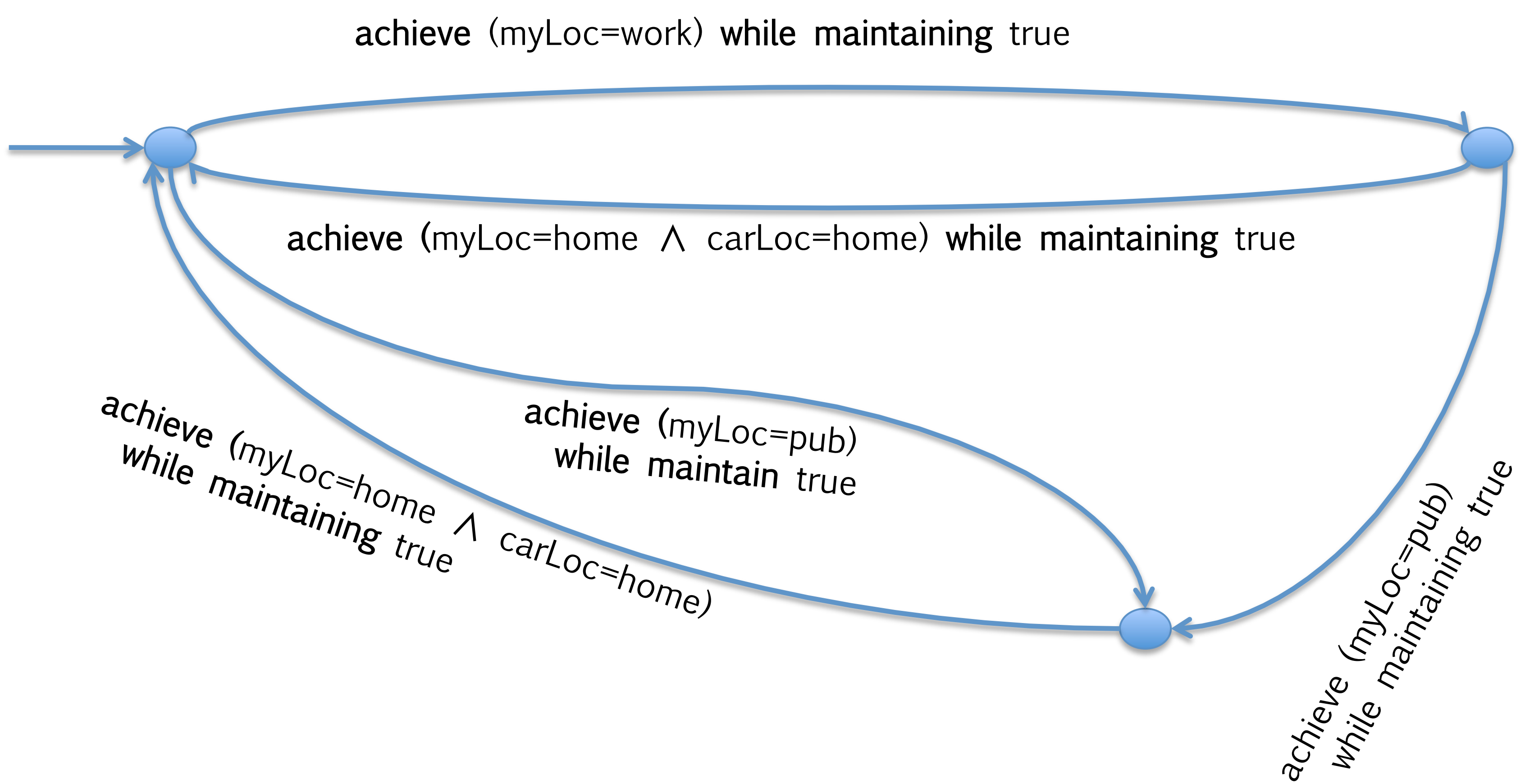


Agent Planning Programs (2)

- Planning programs are possibly non-terminating finite state programs whose atomic instructions are requests for ***achieve*** *a goal* φ ***while maintaining*** *a goal* ψ

- The agent executing a planning program chooses at each point in time which atomic instruction to execute among those that the program makes available at that point

Agent Planning Programs (3)



Planning Program Environment

Planning programs are executed in a planning domain (or Environment)

- State vars:

carLoc, myLoc : {home, work, pub, parking}, strike : {true,false}

- Operators:

goByCar(x) *with* x : {home, parking, pub}
  pre : myLoc=carLoc ∧ carLoc≠pub ∧ myLoc≠x
  post : myLoc=x ∧ carLoc=myLoc

goByBus(x) *with* x : {home, work, pub}
  pre : !strike ∧ myLoc≠x
  post : myLoc=x

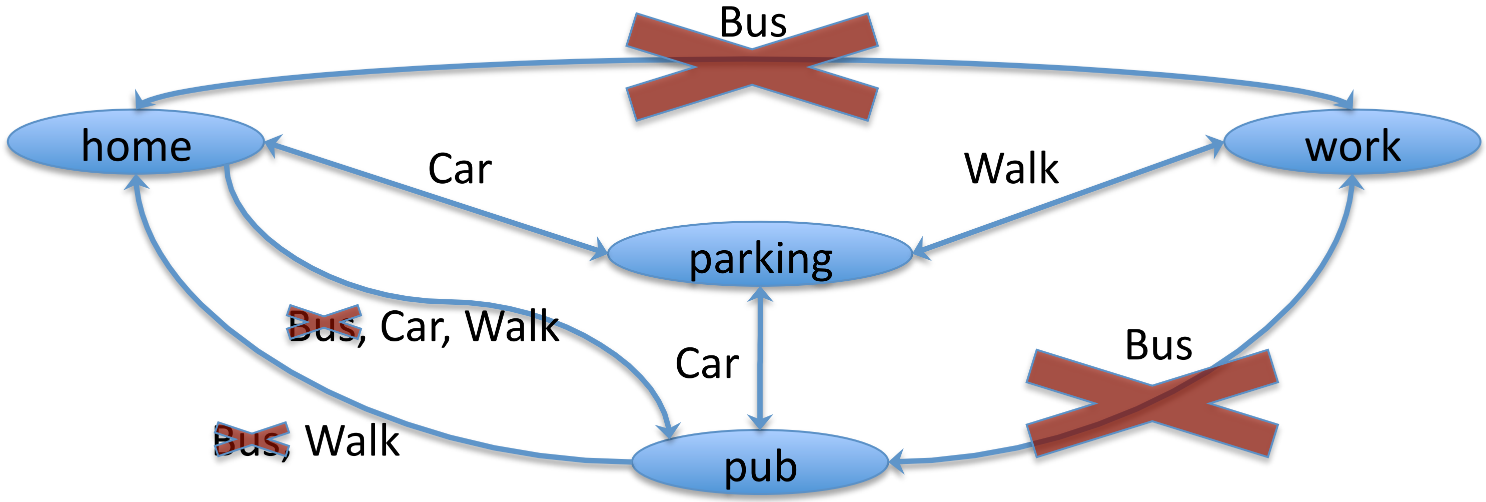
walk(x,y) *with* x,y : {(parking, work), (work, parking), (home, pub), (pub, home)}
  pre : myLoc=x
  post : myLoc=y

- Initial state:

myLoc=home, carLoc=home, strike=true

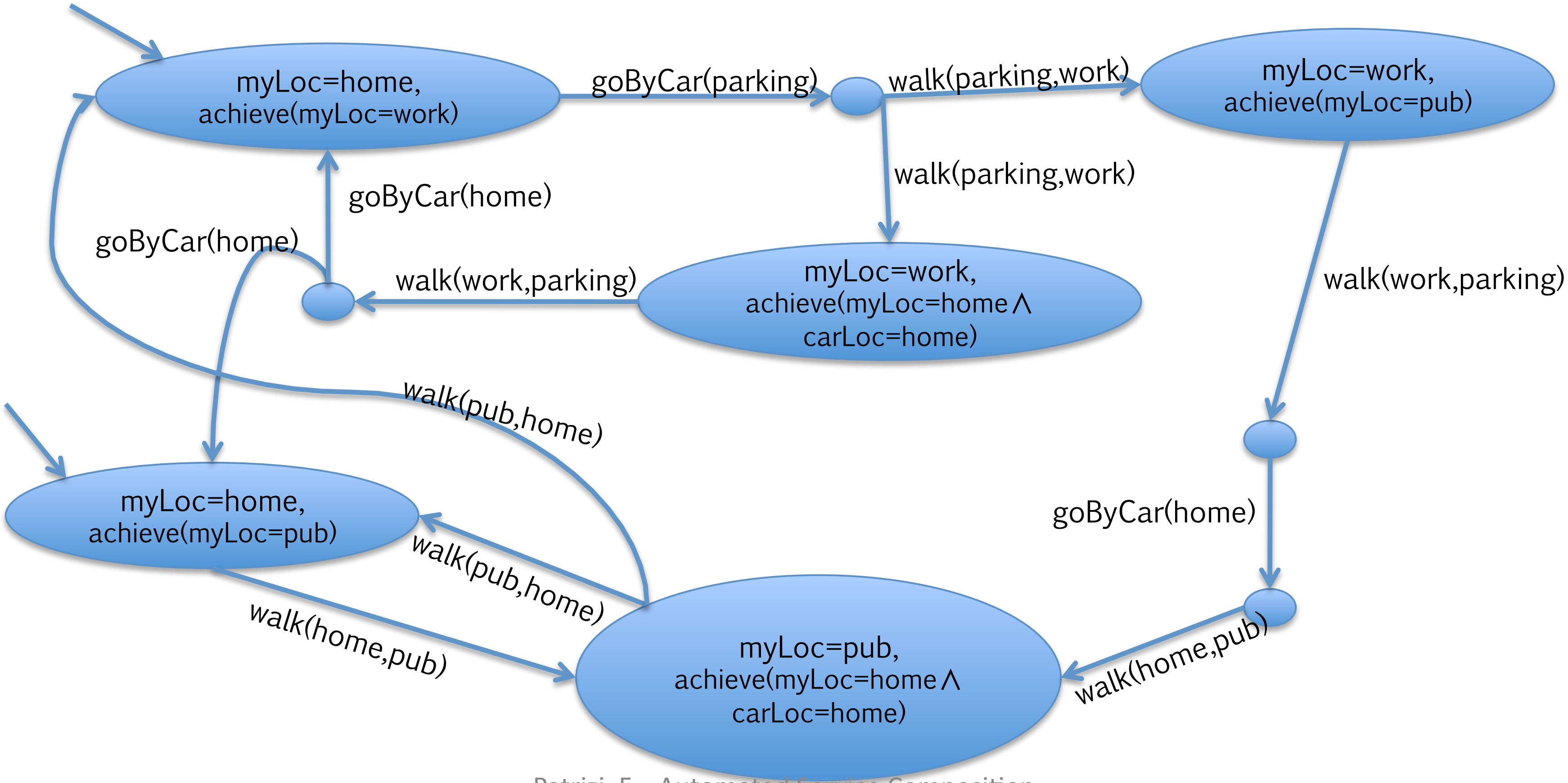
Planning Program Environment (2)

Possible evolution of MyLoc when Strike=true



Planning Program Solution

To execute a planning program we must find plans for all goals in the atomic instructions of the program



Plan-based Simulation Relation

A binary relation R is a ***plan-simulation relation*** *iff:*

- $(t,s) \in R$ implies that

  for all $t \rightarrow_{\text{achieve } \varphi \text{ while maintaining } \psi} t'$

  exists $a_1 a_2 \ldots a_n$ s.t.

  - $s \rightarrow_{a1} s_1 \rightarrow \ldots \rightarrow s_{n-1} \rightarrow_{an} s_n$ (the plan is executable)
  - $s_i \models \psi$, for $s_i = s, s_1 \ldots s_{n-1}$ (the maintenance goal is satisfied)
  - $s_n \models \varphi$ (the achievement goal is satisfied)
  - $(t', s_n) \in R$ (the simulation holds in resulting states)

Planning Program Solution (2)

- The solution of planning programs is based on the computation of the plan-based simulation relation
- Again, the problem is EXPTIME-complete

Conclusion

- Services offer an interesting opportunity for research: need for formal foundations
- Several interesting problems, related to other areas in CS:
  - Database
  - (Generalized) Planning
  - Formal verification and synthesis
- The complexity of the problem calls for efficient solution techniques
- Open problem: How to deal with data?