



**La Sapienza**

Università degli Studi di Roma

# Automatic Composition of e-Services: The “Roman” way

Daniela Berardi

*Dipartimento di Informatica e Sistemistica*

*Università di Roma “La Sapienza”*

`berardi@dis.uniroma1.it`

`http://www.dis.uniroma1.it/~berardi/`

# Overview

- Activity based model: the “Roman” approach
- Composition results in the “Roman” model
- Message based model
- Activity vs Message based model
- Embedding Activity based model into SitCalc
- Embedding Activity based model into PSL

# e-Services and Community of e-Services: The Model used by “Roman” Results

- An *e*-Service is an **interactive program** that **exports its behavior** in terms of an **abstract description**
- A **client selects and interacts** with it according to the description exported
- A **community** of *e*-Services is:
  - a **set** of *e*-Services ...
  - ... that share implicitly a **common understanding** on a **common set of actions** and export their **behavior** using this **common set of actions**
- A **client** specifies needs as *e*-Service behavior using the **common set of actions** of the community

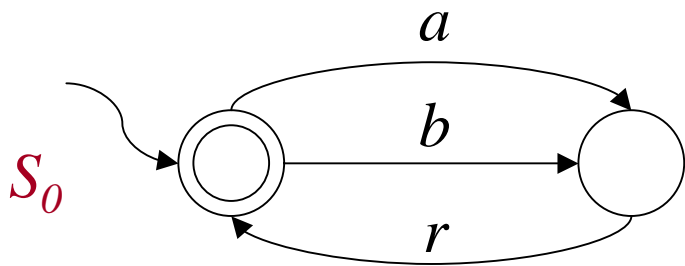
# e-Service Exports its Behavior ...

Many possible ways. **In this talk...**

- Behavior modeled by **finite state machines**  
*core of state chart, UML state-transition diagram, etc.*
  - in our FSMs, each transaction corresponds to an action (*e.g., search-by author-and-select, search-by title-and-select, listen-the-selected-song, ...*)
- In fact using a FSM we compactly describe all possible sequences of **deterministic** (atomic) **actions**: tree of all possible sequences of actions
- **Data** produced by actions **not explicitly** modeled  
*data are used by the client to choose next action*

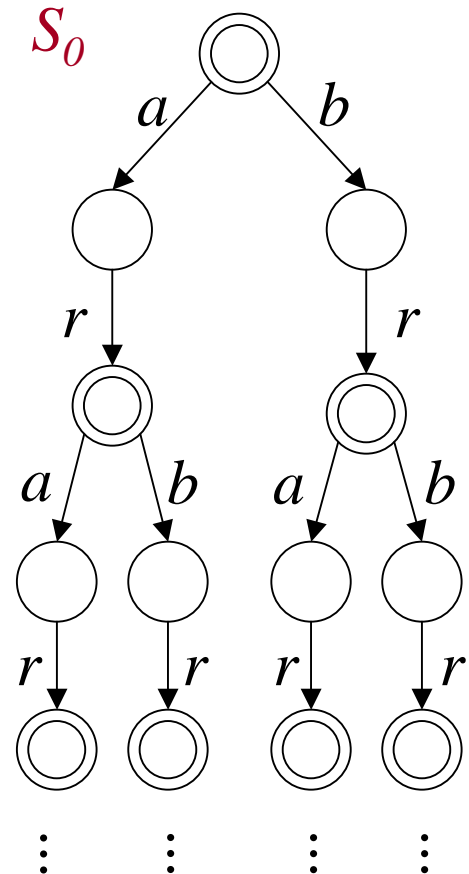
# e-Service as Execution Tree

*Required behavior represented as a FSM*



$a$ : “search by author (and select)”  
 $b$ : “search by title (and select)”  
 $r$ : “listen (the selected song)”

*Execution tree  
(obtained by FSM unfolding)*



# e-Service Composition in the “Roman Framework”

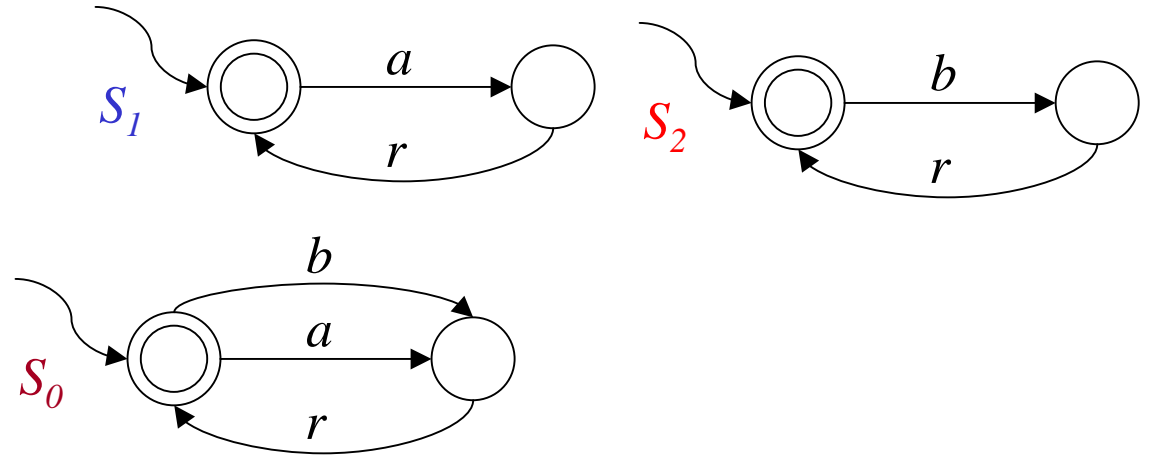
## Given:

- Community  $C$  of e-Services

(expressed as FSMs)

- Target e-Service  $S_0$

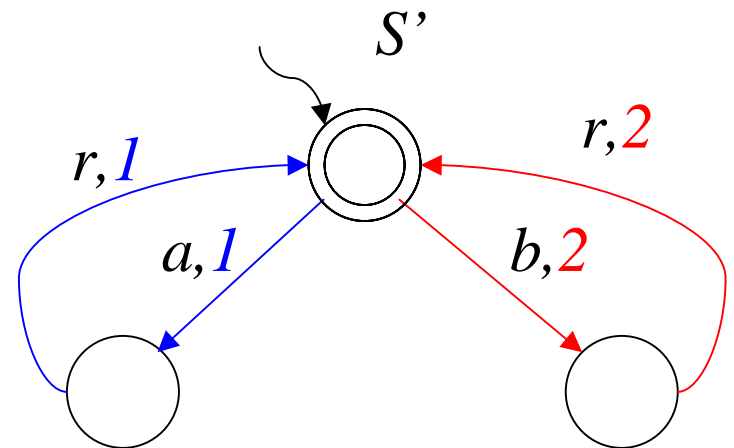
(again expressed as FSM)



## Find:

- new FSM e-Service  $S'$  (delegator):

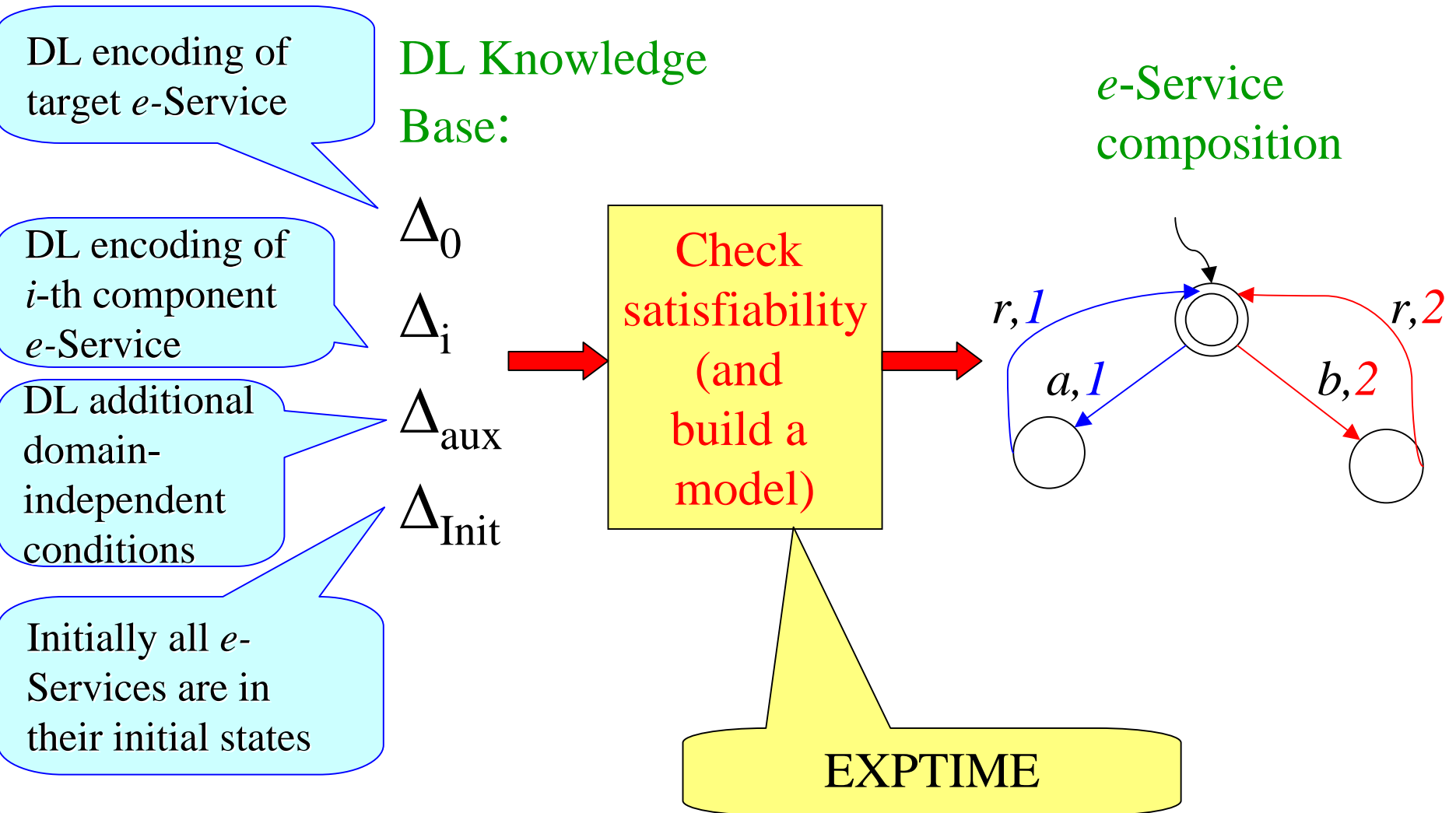
- new alphabet = actions  $x$  (sets of service identifiers)
- “accepts” same language as  $S_0$
- For each accepting run of  $S'$  on word  $w$ , and for each  $S$  in  $C$ , “projection” of this run onto moves of  $S$  is an accepting computation for  $S$



# Key Idea for Finding Composition: Exploit Description Logics (DLs)

- Description Logics:
  - represent knowledge in terms of classes and relationships between classes
  - equipped with decidable reasoning
- Interesting properties:
  - Tree model property
  - Small model property
  - EXPTIME decidability

# Results on Automatically Building e-Service Composition



# Results

**Thm 1:** Composition exists iff DL Knowledge Base satisfiable

*From composition labeling of the target e-Service one can build a tree model for the Knowledge Base, and vice-versa*

**Cor 1:** Composition existence of e-Services, expressible as FSMs, is decidable in EXPTIME

**Thm 2:** If composition exists then finite state composition exists.

*From a small model of a DL Knowledge Base, one can build a finite state composition*

**Cor 2:** Finite state composition existence of e-Services, expressible as FSMs, is decidable in EXPTIME

⇒ **Building finite state composition can be done in EXPTIME**

# Message Based Model

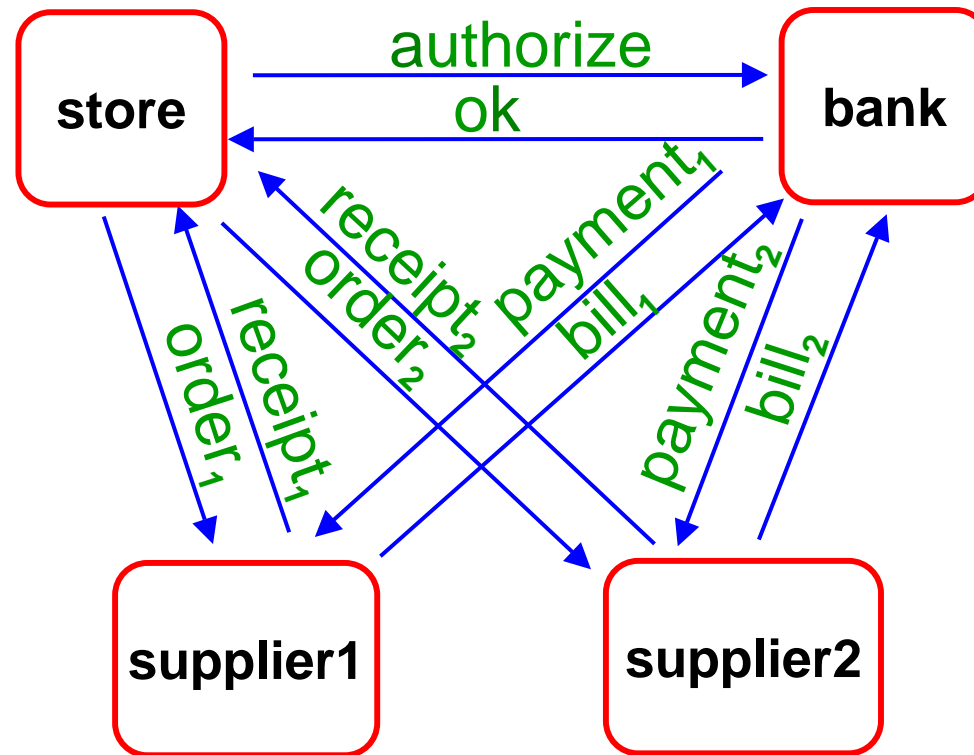
## ec-Schema:

- finite set of abstract **peers** (*e-Services*)
  - peers can be implemented as FSM with input/output
  - each peer has a (bounded) queue
  - ⇒ asynchronous communication between peers
- finite set of **channels**
  - i.e.,  $\{ \langle \text{sender}, \text{receiver}, \text{message\_type} \rangle \}$
- finite set of incoming and outgoing **messages** over some alphabet  $\Sigma$ 
  - input messages:  $?a, a \in \Sigma$
  - output messages  $!a, a \in \Sigma$
  - As technical simplification in theoretical model, each symbol “a” encodes a triple  $\langle \text{sender}, \text{receiver}, \text{message-type} \rangle$
- **Conversation language**: sequence of messages exchanged between peers

Model is peer-to-peer, but can restrict to **mediated** by assuming “hub-and-spoke” connection graph. (I.e., one **peer** acts as the

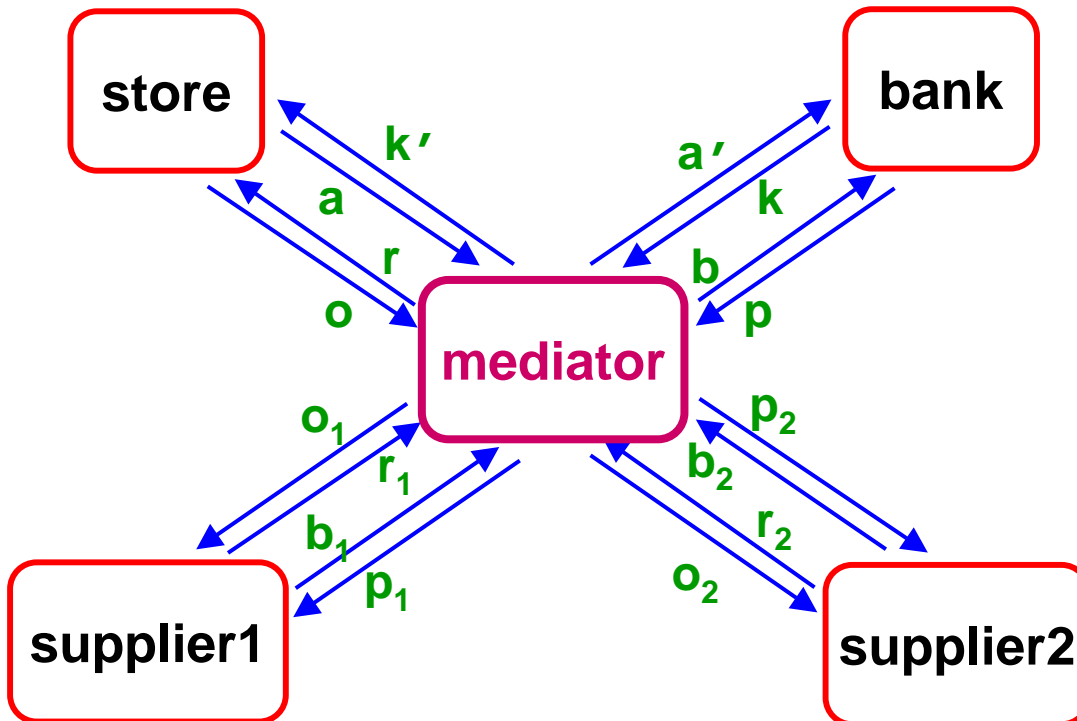
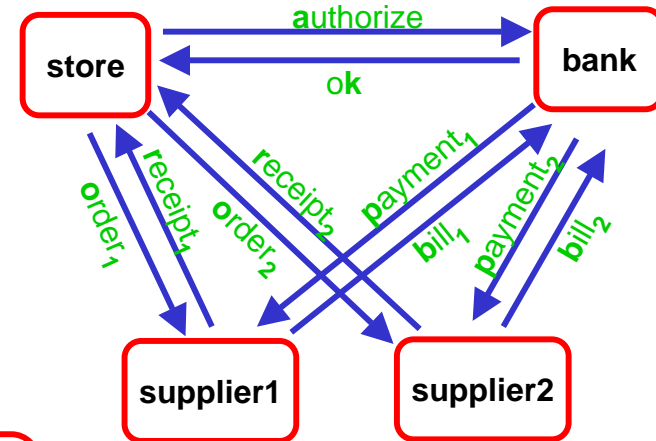
# E-Composition Schema

- An E-C schema specifies the infrastructure of composition
- Assume finite domains  $\Rightarrow$  can model parameters



# Composition Infrastructure

- Peer-to-peer (distributed control)



- Hub-and-spoke (centralized control)

# Peer Synthesis Statement and Results

- Problem statement
  - Given: ec-schema and LTL formula  $\varphi$
  - Create: a FSM for each peer so that  $\varphi$  is satisfied
  - Note: not a composition problem, because this result is creating peers, not selecting them from a pre-existing “UDDI”
- Synthesis results for Mealy implementations with bounded queues
  - **Mealy peer synthesis:** decidable
    - Propositional LTL description  $\Rightarrow$  PSPACE
- (Also, results contrasting bounded vs. unbounded message queues)

# “Roman” Activity Based Composition Result vs Message Based Synthesis Result

- Activity based Model:
  - behavior modeled as FSM, with transitions labeled by actions
  - client/server model: “active” client: s/he **selects from a set of choices** presented by e-service
- Result
  - Start with community of activity-based FSMs (e-services)
  - FMSs define constraint on legal sequence of actions executed by each peer
  - given a **branching time** spec.  $\Psi$  of global behavior and “constrained” peers, **synthesize a delegator**
  - **peers communicate only with delegator**
  - determinism only (for the moment)
- Message based Model:
  - behavior modeled as FSM, with transitions labeled by input/output messages
  - peer-to-peer model; no notion corresponding to client in activity model
- Result
  - Start with “ec-schema” which establishes topology for message-passing
  - no constraint on legal sequences of actions executed by each abstract peer
  - given a **LTL** spec.  $\Phi$  of global behavior and “ec-schema”, **synthesize peers** such that  $\Phi$  is realized
  - **peer-to-peer communication**
  - non determinism over messages (i.e., same message labeling different transition from same state)

# “Roman” Activity Based vs Message Based

- “Roman” Activity based and Message based are complementary approaches:
  - Can merge them?
  - How?
- (other) “Roman” Activity based future work:
  - is our algorithm EXPTIME-hard?
  - currently we are working on a DL based prototype system that implements our composition algorithm
  - also working on notion of “k-look-ahead” compositions - gives more flexibility than first Roman results
  - add non determinism
  - data (i.e., parameters of actions)

# Situation Calculus Encoding of Roman Model -- Idea

- Each *e-Service*  $i$  as **Reiter's Basic Action Theory**  $\Gamma_i$ :
  - each action as a Situation Calculus action
  - each state of FSM is a fluent
  - special fluent *Final* to indicate situation when e-Service execution can stop.
    - $\Rightarrow$  In  $\Gamma_i$  we have complete information on the initial situation and hence on the whole theory.
- *e-Service composition*:
  - represent which *e-Services* (in the community) are executed, when an action of the target *e-Service* is performed, by predicates  $Step_i(a, s)$ , denoting that *e-Service*  $i$  executes action  $a$  in situation  $s$ .
    - $\Rightarrow$  Situation Calculus Theory (but not basic)
    - $\Rightarrow$  Incomplete information over  $Step_i(a, s)$
  - rename *Poss* to  $Poss_i$ , rename *Final* to  $Final_i$
  - suitably modify the successor axioms to cope with  $Step_i(a, s)$

## Sit Calc Encoding -- Details

- Target *e*-Service  $E_0 = (\Sigma, Q_0, q^0_0, \delta_0, \mathcal{F}_0)$

(Reiter Basic Action Theory)

- $F_{q^0_0}(S_0)$  *initial situation*
- $\forall s. F_q(s) \supset \neg F_{q'}(s)$  *for all pairs of distinct states  $q, q'$  in  $E_0$   
e-Service states are pair-wise disjoint*
- $\forall s. \text{Poss}(a, s) \equiv \bigvee_{q \text{ st } \delta(q, a) \text{ is defined}} F_q(s)$
- $\forall s \forall \alpha. F_{q'}(\text{do}(\alpha, s)) \equiv \bigvee_{a, q, \text{ st } q' = \delta(q, a)} (\alpha = a \wedge F_q(s)) \vee$   
 $(F_{q'}(s) \wedge \bigwedge_{b \text{ st } \delta(q', b) \text{ is defined}} \alpha \neq b)$   
*for each  $q' = \delta_0(q, a)$   
target e-Service can do an a-transition going to state  $q'$*
- $\forall s. \text{Final}(s) \equiv \bigvee_{q \in \mathcal{F}_0} F_q(s)$   
*denotes target e-Service final states*

## Sit Calc Encoding -- Details (cont.d)

- Community *e*-Services  $E_i = (\Sigma, Q_i, q_i^0, \delta_i, \mathcal{F}_i)$ 
  - $F_{q_i^0}(S_0^i)$  *initial situation*
  - $\forall s. F_q(s) \supset \neg F_{q'}(s)$  *for all pairs of distinct states  $q, q'$  in  $E_i$   
e-Service states are pair-wise disjoint*
  - $\forall s. \text{Poss}_i(a, s) \equiv \bigvee_{q \text{ st } \delta_i(q, a) \text{ is defined}} F_q(s)$
  - $\forall s \forall \alpha. F_{q'}(\text{do}(\alpha, s)) \equiv$   
 $(\bigvee_{a, q, \text{ st } q' = \delta_i(q, a)} (\alpha = a \wedge F_q(s) \wedge \text{Step}_i(\alpha, s))) \vee$   
 $(\neg \text{Step}_i(\alpha, s) \wedge F_{q'}(s))$   
*for each  $q' = \delta_i(q, a)$   
if e-Service moved then new state, otherwise old state*
  - $\forall s. \text{Final}_i(s) \equiv \bigvee_{q \in \mathcal{F}_i} F_q(s)$  *denotes community e-Service final states*

## SitCalc Encoding -- Details (cont.d)

- Foundational, domain independent axioms:

- $\forall s, a. \text{Poss}(a, s) \wedge \neg \text{Final}(s) \rightarrow \bigvee_{i=1..n} \text{Step}_i(a, s) \wedge \text{Poss}_i(a, s)$   
*for each action a  
at least one of the community e-Services must move at each step*

- $\forall s. \text{Final}(s) \rightarrow \bigwedge_{i=1..n} \text{Final}_i(s)$   
*when target e-Service is final all comm. e-Services are final*

- $\bigwedge_{i=0..n} F_{qi0}(S_0^i)$   
*in the initial situation all e-Services are in their initial state*

# PSL Encoding of Roman Model -- Idea

- Based on Rick Hull and Michael Gruninger encoding of message based model in PSL
- Basic idea to model an *e-Service*:
  - fluents to denote:
    - initial situation (*Init*)
    - states of FSM ( $F_q$ ),
    - final states (*Final*),
  - one activity for each action
  -
- Component *e-Services*:
  - rename *poss* to *poss<sub>i</sub>*, rename *Final* to *Final<sub>i</sub>*,
  - fluent *Step<sub>ai</sub>* to denote which component *e-Service* “moves”

# PSL Encoding of Roman Model -- Idea

- Based on Rick Hull and Michael Gruninger encoding of message based model in PSL
- Basic idea to model an *e-Service*:
  - fluents to denote:
    - initial situation (*Init*)
    - states of FSM ( $F_q$ ),
    - final states (*Final*),
  - one activity for each action
- Component *e-Services*:
  - rename *poss* to *poss<sub>i</sub>*, rename *Final* to *Final<sub>i</sub>*,
  - fluent *Step<sub>ai</sub>* to denote which component *e-Service* “moves”

very  
similar to  
Sit Calc !

## PSL Encoding -- Details

- Target *e*-Service  $E_0 = (\Sigma, Q_0, q^0_0, \delta_0, F_0)$ 
  - $\forall o.\text{prior}(F_q \supset \neg F_{q'}, o)$   
*for all pairs of distinct states  $q, q'$  in  $E_0$   
 $e$ -Service states are pair-wise disjoint*
  - $\forall o.\text{holds}(F_q, o) \supset \text{poss}(a, o)$  (*prec*)  
 $\forall o.\text{occurrence\_of}(o, a) \wedge \text{prior}(F_q, o) \supset \text{holds}(F_{q'}, o)$  (*eff*)  
*for each  $q' = \delta_0(q, a)$   
target  $e$ -Service can do an  $a$ -transition going to state  $q'$*
  - $\forall o.\text{holds}(F_q, o) \wedge \text{poss}(a, o) \supset \text{false}$  *for each  $\delta_0(q, a)$  undef.*  
*target  $e$ -Service cannot do an  $a$ -transition*
  - $\text{Final} \equiv \bigvee_{q \in F_0} F_q$   
*denotes target  $e$ -Service final states*

## PSL Encoding -- Details

- Target *e*-Service  $E_0 = (\Sigma, Q_0, q_0^0, \delta_0, F_0)$

- $\forall o. \text{prior}(F_q \supset \neg F_{q'}, o)$

- $\forall o. \text{holds}(F_q, o) \supset \text{poss}(a, o)$  (*prec*)

- $\forall o. \text{occurrence\_of}(o, a) \wedge \text{prior}(F_q, o) \supset \text{holds}(F_q, o)$  (*eff*)

- $\forall o. \text{holds}(F_q, o) \wedge \text{poss}(a, o) \supset \text{false}$

- $\text{Final} \equiv \bigvee_{q \in \mathcal{F}_0} F_q$

similar to  
Sit Calc !

## PSL Encoding -- Details (cont.d)

- Community *e*-Services  $E_i = (\Sigma, Q_i, q^0_i, \delta_i, F_i)$ 
  - $\forall o. \text{prior}(F_q \supset \neg F_{q'}, o)$  *for all pairs of distinct states  $q, q'$  in  $E_i$   
e-Service states are pair-wise disjoint*
  - $\forall o. \text{holds}(F_q, o) \supset \text{poss}_i(a, o)$  *(prec)*
  - $\forall o. \text{occurrence\_of}(o, a) \wedge \text{prior}(F_q, o) \supset$  *(eff)*  
 $(\text{holds}(F_{q'}, o) \wedge \text{holds}(\text{Step}_{ia}, o)) \vee (\text{holds}(F_q, o) \wedge \neg \text{holds}(\text{Step}_{ia}, o))$   
*for each  $q' = \delta_i(q, a)$   
if e-Service moved then new state, otherwise old state*
  - $\forall o. \text{holds}(F_q, o) \wedge \text{poss}_i(a, o) \supset \text{false}$
  - $\forall o. \text{occurrence\_of}(o, a) \wedge \text{prior}(F_q, o) \supset$   
 $\text{holds}(F_q, o) \wedge \neg \text{holds}(\text{Step}_{ia}, o)$  *for each  $\delta_i(q, a)$  undef.  
if e-Service cannot do  $a$ , and  $a$  is performed then it did not move*
  - $\text{Final}_i \equiv \bigvee_{q \in F_i} F_q$  *denotes community e-Service final states*

## PSL Encoding -- Details (cont.d)

- Community *e*-Services  $E_i = (\Sigma, Q_i, q_i^0, \delta_i, \mathcal{F}_i)$

$$- \forall o. \text{prior}(\mathbf{F}_q \supset \neg \mathbf{F}_{q'}, o)$$

$$- \forall o. \text{holds}(\mathbf{F}_q, o) \supset \text{poss}_i(a, o) \quad (\text{prec})$$

$$\forall o. \text{occurrence\_of}(o, a) \wedge \text{prior}(\mathbf{F}_q, o) \supset \quad (\text{eff})$$
$$(\text{holds}(\mathbf{F}_{q'}, o) \wedge \text{holds}(\text{Step}_{ia}, o)) \vee (\text{holds}(\mathbf{F}_q, o) \wedge \neg \text{holds}(\text{Step}_{ia}, o))$$

$$- \forall o. \text{holds}(\mathbf{F}_q, o) \wedge \text{poss}_i(a, o) \supset \text{false}$$

$$\forall o. \text{occurrence\_of}(o, a) \wedge \text{prior}(\mathbf{F}_q, o) \supset$$
$$\text{holds}(\mathbf{F}_q, o) \wedge \neg \text{holds}(\text{Step}_{ia}, o)$$

$$- \text{Final}_i \equiv \bigvee_{q \in \mathcal{F}_i} \mathbf{F}_q$$

similar to  
Sit Calc !

## PSL Encoding -- Details (cont.d)

- Additional assertions:

- $\forall o. \text{poss}(a, o) \wedge \text{occurrence\_of}(o, a) \supset \bigvee_{i=1..n} \text{step}_{ia}(o) \wedge \text{poss}_i(a, o)$   
*for each action a  
at least one of the community e-Services must move at each step*

- $\forall o. \text{prior}(\text{Final} \supset \bigwedge_{i=1..n} \text{Final}_i, o)$   
*when target e-Service is final all comm. e-Services are final*

- $\text{Init} \equiv \bigwedge_{i=0..n} \mathbf{F}_{qi0}$   
*Initially all e-Services are in their initial state*

## PSL Encoding -- Details (cont.d)

- Additional assertions:

$$- \forall o. \text{poss}(a, o) \wedge \text{occurrence\_of}(o, a) \supset \bigvee_{i=1..n} \text{step}_{ia}(o) \wedge \text{poss}_i(a, o)$$

$$- \forall o. \text{prior}(\text{Final} \supset \bigwedge_{i=1..n} \text{Final}_i, o)$$

$$- \text{Init} \equiv \bigwedge_{i=0..n} \mathbf{F}_{qi0}$$

similar to  
Slt Calc!

# Info & Contacts

- Thesis dissertation scheduled for January 2005

**Daniela Berardi**

*e-mail:* **berardi@dis.uniroma1.it**

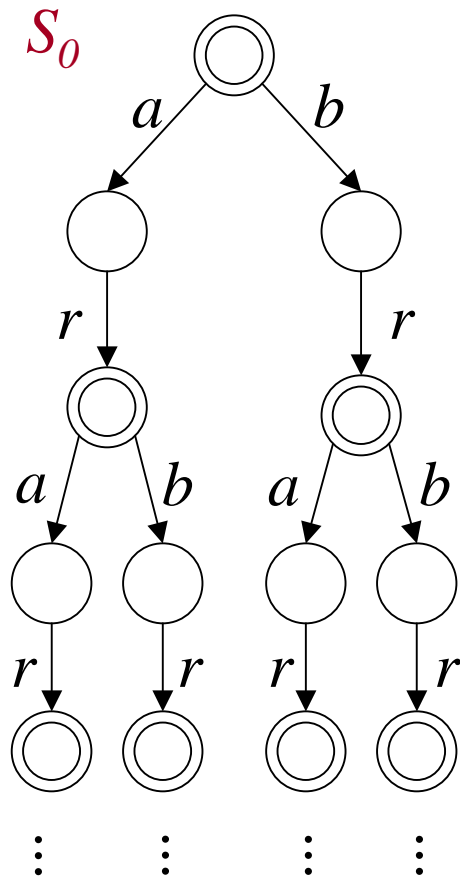
*home page:* **<http://www.dis.uniroma1.it/~berardi>**

*address:* **Via Salaria, 113 (2 piano)  
I-00198 Rome (Italy)**

Back up

# Execution tree

*An execution tree*



*a: “search by author (and select)”*

*b: “search by title (and select)”*

*r: “listen (the selected song)”*

- *Nodes: history* (sequence) of actions executed so far
- *Root: no action yet performed*
- *Successor node  $x \cdot a$  of  $x$ : action  $a$  can be executed after the sequence of action  $x$*
- *Final nodes: the e-Service can terminate*

# e-Service composition

- Added value of the community:

*when a client request cannot be satisfied by any available e-Service, it may still be possible to satisfy it by combining “pieces” of e-Services in the community*

- Two issues arise:

- support for **synthesizing composition**:
  - automatic synthesis of a **coordinating program (composition)** ...
  - ... that realizes the **target e-Service** (client request) ...
  - ... by suitably **coordinating** available e-Services

*addressed here*

- support for **orchestration**: execution of the coordinating program

*not addressed here*

# Formalizing *e*-Service composition

## Composition:

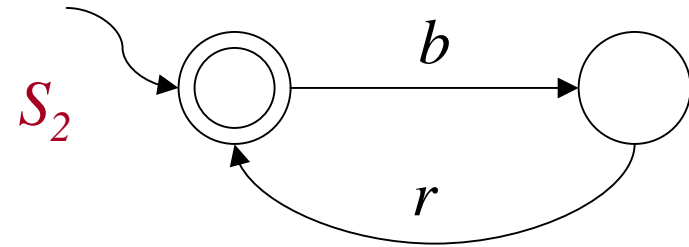
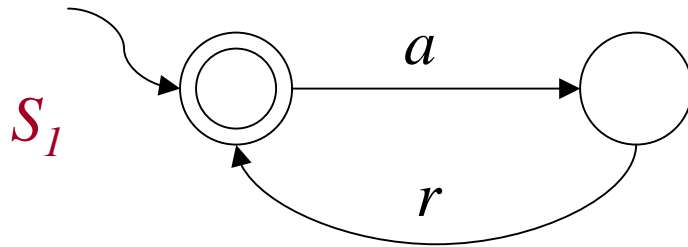
- coordinating program ...
- ... that realizes the target *e*-Service ...
- ... by suitably coordinating available *e*-Services

⇒ Composition can be **formalized** as:

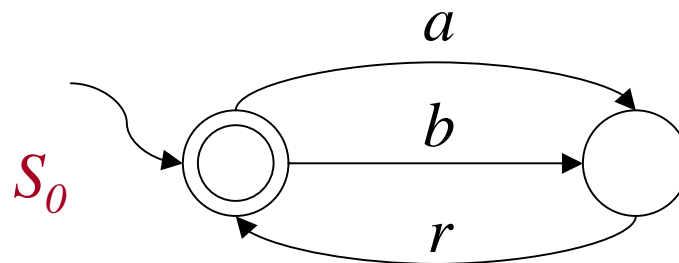
- a **labeling of the execution tree** of the target *e*-Service such that ...
- ... each **action** in the execution tree is labeled by the community *e*-Service that executes it ...
- ... and each possible sequence of actions on the target *e*-Service execution tree corresponds to possible sequences of actions on the community *e*-Service execution trees, **suitably interleaved**.

# Example of composition

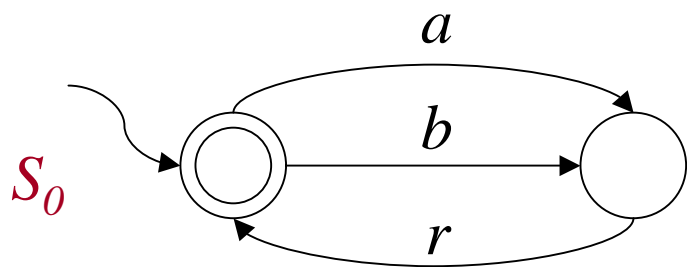
- Community *e*-Services (expressed as FSMs)



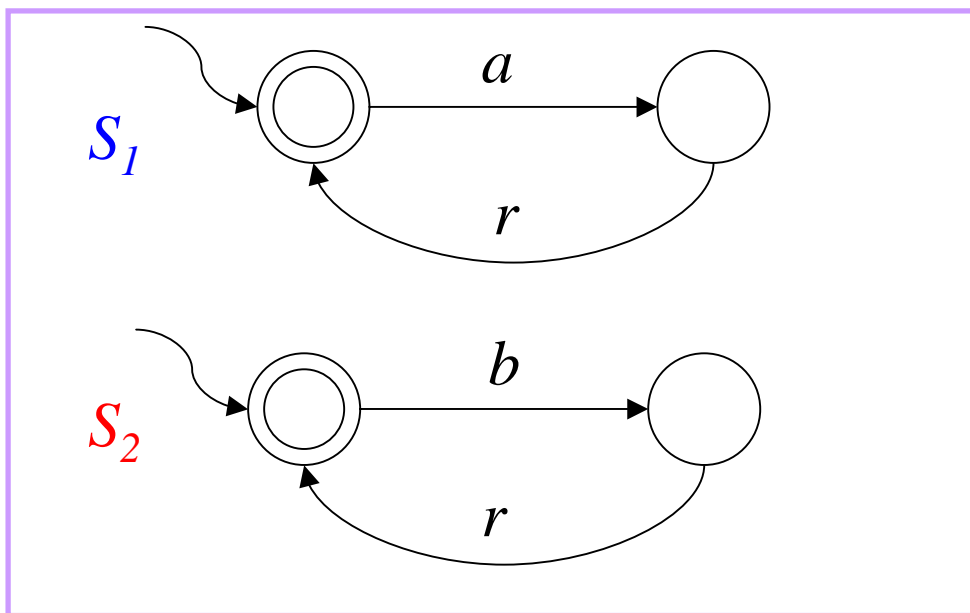
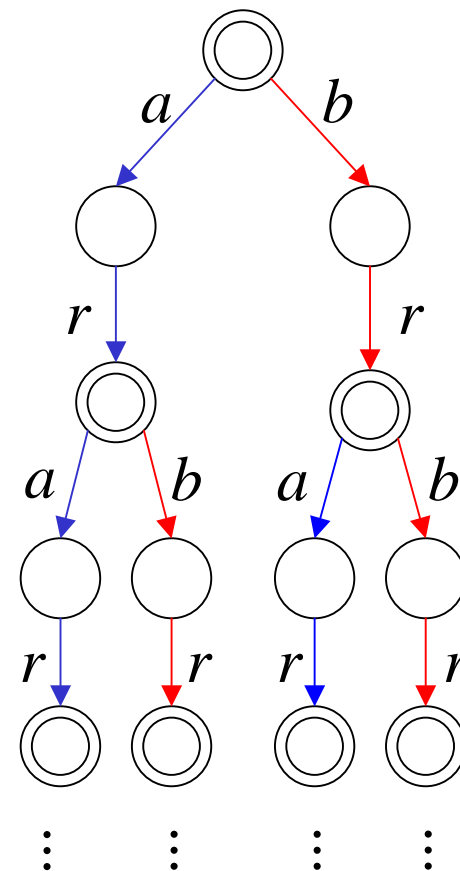
- Target *e*-Service (again expressed as FSM)



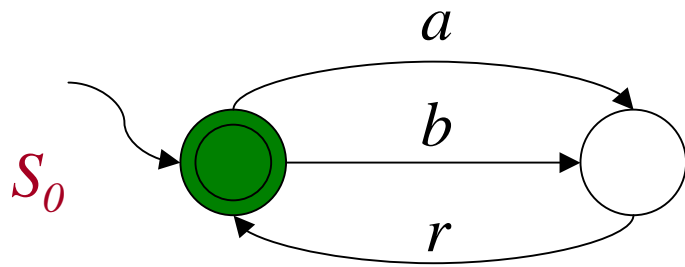
# Example of composition



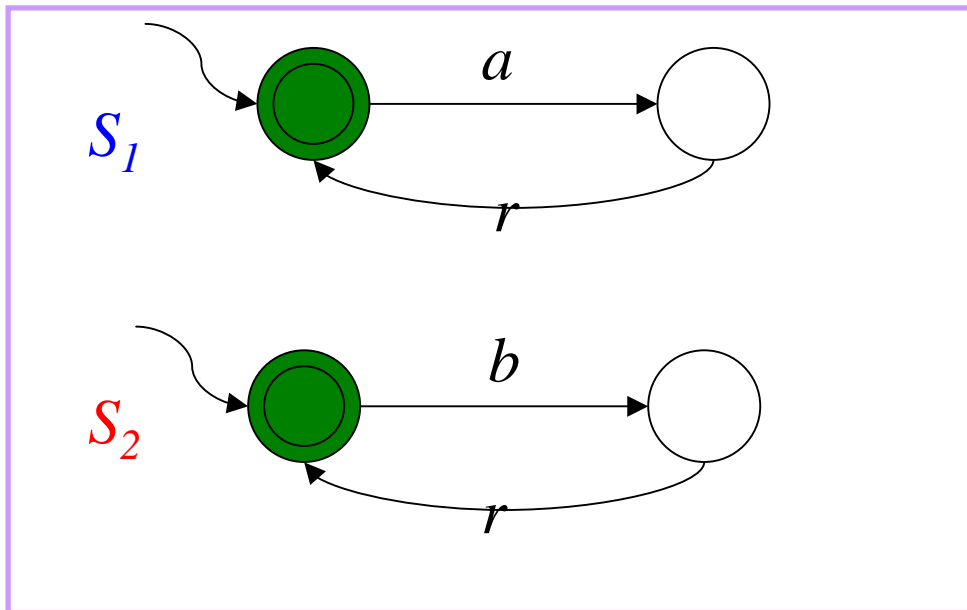
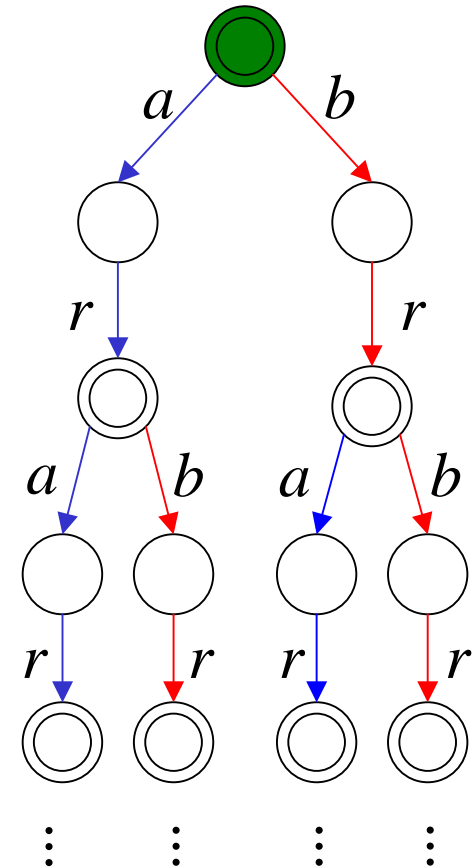
coordinating program (composition)



# Example of composition

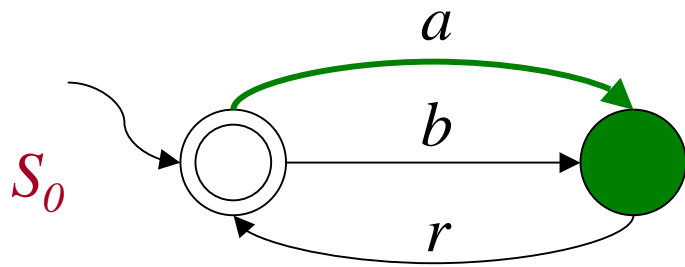


coordinating program (composition)

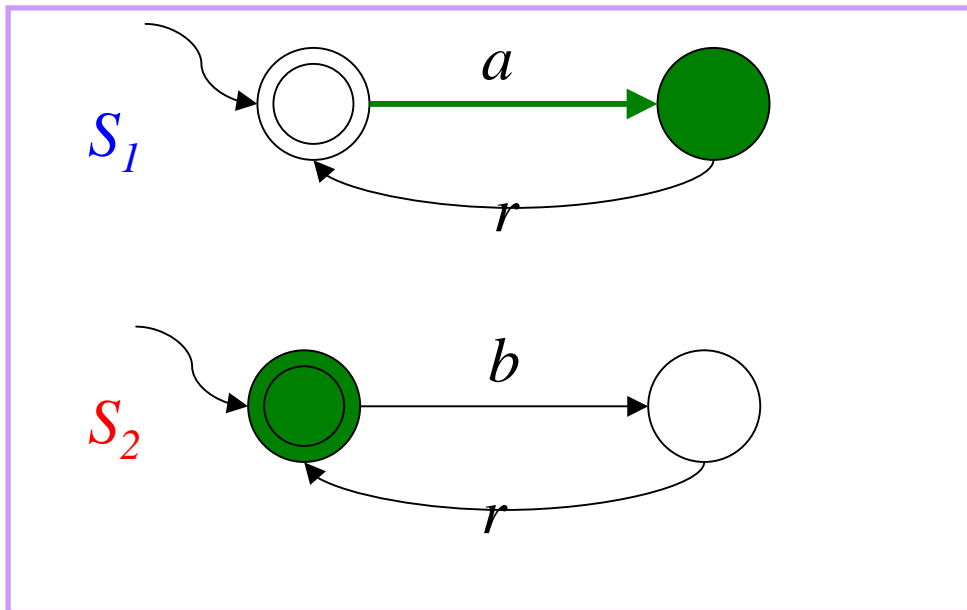
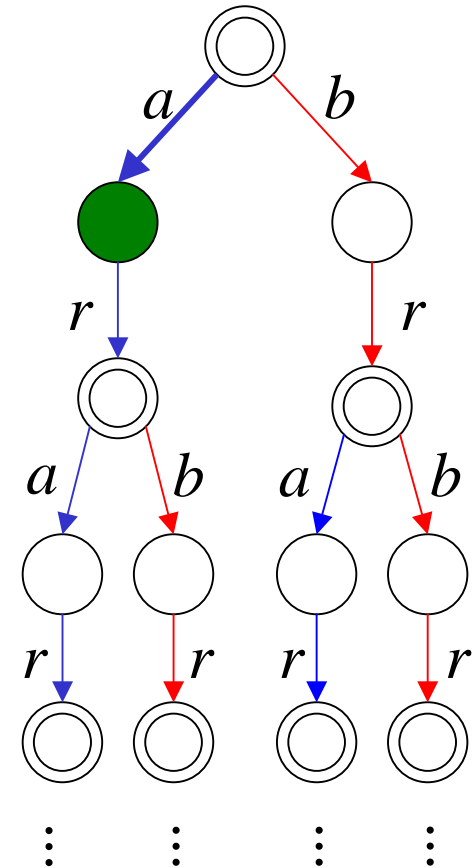


*All e-Services start from their starting state*

# Example of composition



coordinating program (composition)

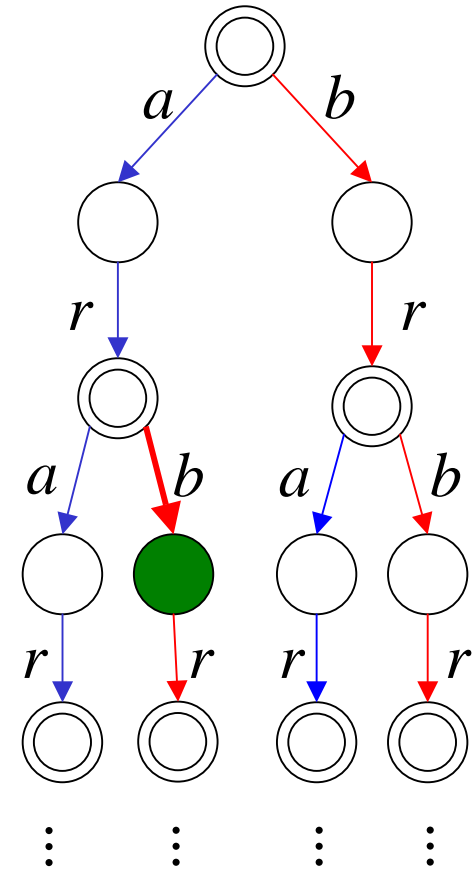
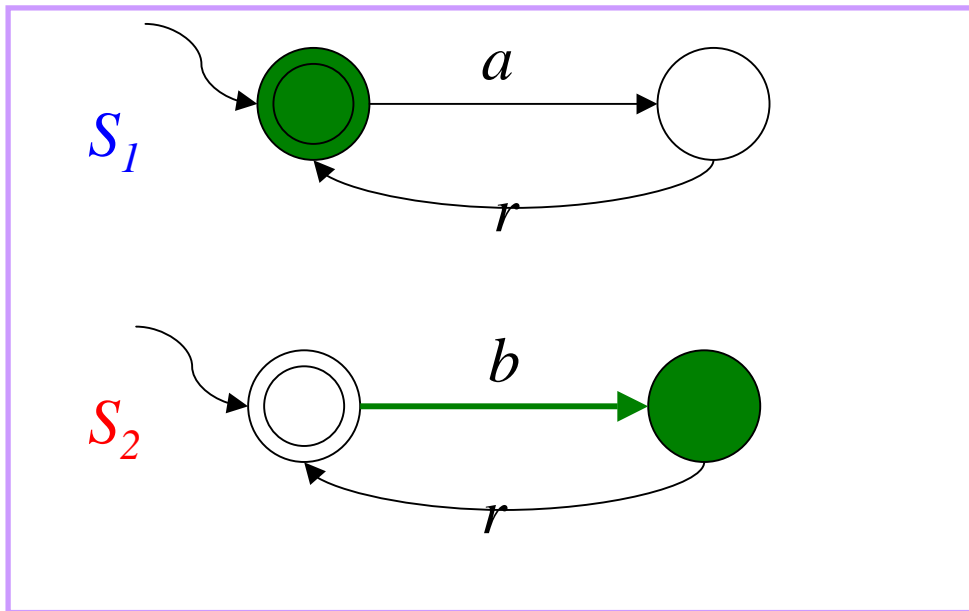
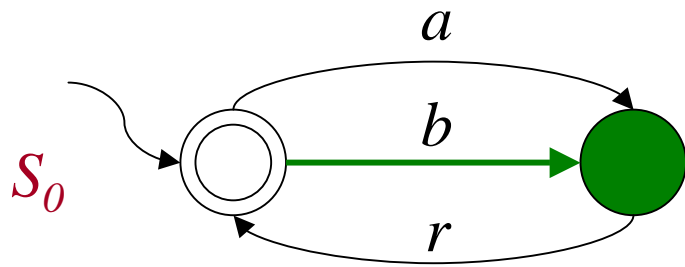


*Each action of the target e-Service is executed by at least one of the component e-Services*

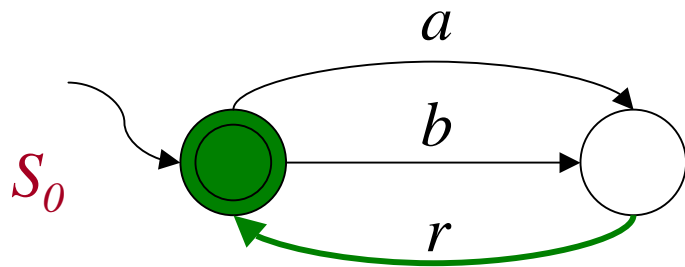


# Example of composition

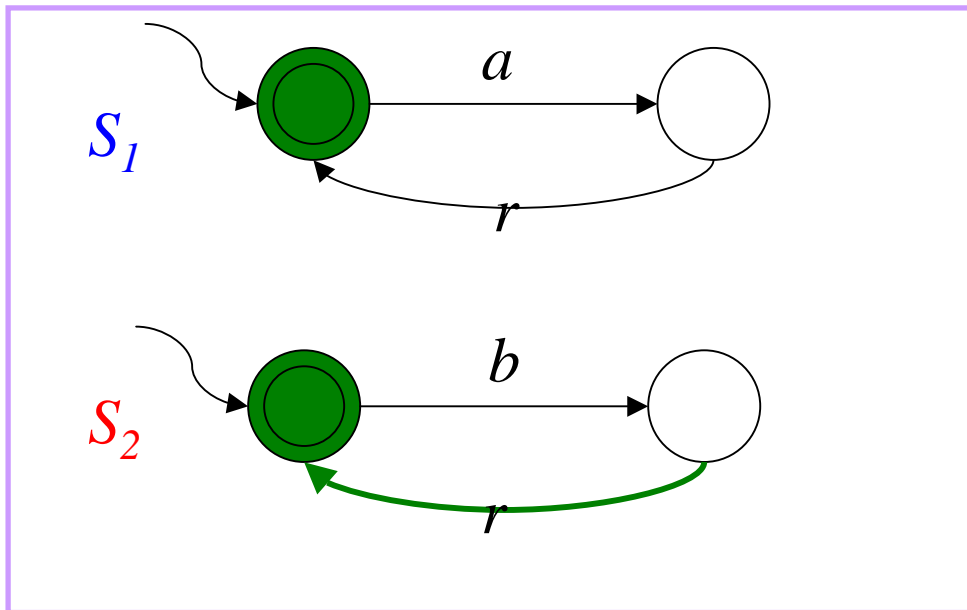
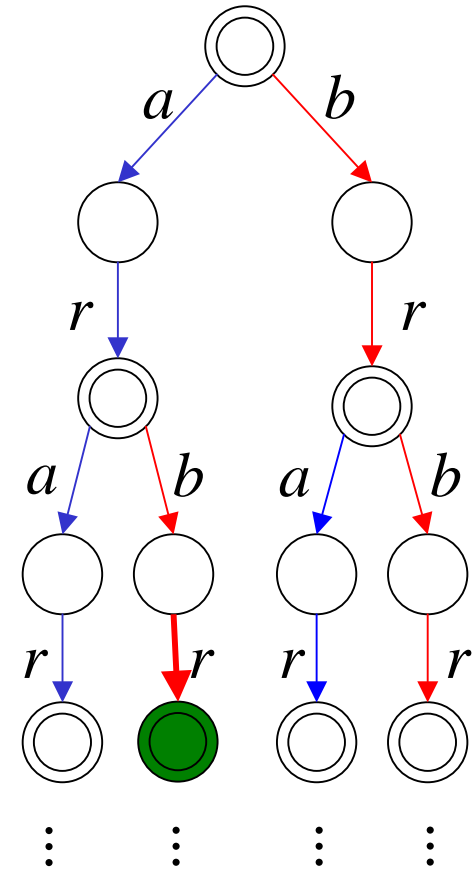
coordinating program (composition)



# Example of composition



coordinating program (composition)



# $\mathcal{ALC}$ encoding

- Target  $e$ -Service  $S_0 = (\Sigma, S_0, s_0^0, \delta_0, F_0)$ 
  - $s \sqsubseteq \neg s'$  for all pairs of distinct states in  $S_0$   
 *$e$ -Service states are pair-wise disjoint*
  - $s \sqsubseteq \exists a. \top \sqcap \forall a. s'$  for each  $s' = \delta_0(s, a)$   
*target  $e$ -Service can do an  $a$ -transition going to state  $s'$*
  - $s \sqsubseteq \forall a. \perp$  for each  $\delta_0(s, a)$  undef.  
*target  $e$ -Service cannot do an  $a$ -transition*
  - $F_0 \equiv \sqcup_{s \in F_0} s$   
*denotes target  $e$ -Service final states*
- ...

## $\mathcal{ALC}$ encoding (cont.d)

- Community  $e$ -Services  $\mathcal{S}_i = (\Sigma, \mathcal{S}_i, s^0_i, \delta_i, F_i)$ 
  - $s \sqsubseteq \neg s'$  for all pairs of distinct states in  $\mathcal{S}_i$   
 *$e$ -Service states are pair-wise disjoint*
  - $s \sqsubseteq \forall a. (\text{moved}_i \sqcap s' \sqcup \neg \text{moved}_i \sqcap s)$  for each  $s' = \delta_i(s, a)$   
*if  $e$ -Service moved then new state, otherwise old state*
  - $s \sqsubseteq \forall a. (\neg \text{moved}_i \sqcap s)$  for each  $\delta_i(s, a)$  undef.  
*if  $e$ -Service cannot do  $a$ , and  $a$  is performed then it did not move*
  - $F_i \equiv \sqcup_{s \in F_i} s$   
*denotes community  $e$ -Service final states*
- ...

# $\mathcal{ALC}$ encoding (cont.d)

- Additional assertions

- $\exists a. \top \sqsubseteq \forall a. \sqcup_{i=1, \dots, n} \text{moved}_i$  for each action  $a$   
*at least one of the community e-Services must move at each step*

- $F_0 \sqsubseteq \prod_{i=1, \dots, n} F_i$   
*when target e-Service is final all comm. e-Services are final*

- $\text{Init} \equiv s_0^0 \sqcap \prod_{i=1, \dots, n} s_i^0$   
*Initially all e-Services are in their initial state*

# DPDL encoding

$$\Phi = \mathbf{Init} \wedge ([\mathbf{u}]\Phi_0 \wedge_{i=1,\dots,n} [\mathbf{u}]\Phi_i \wedge [\mathbf{u}]\Phi_{aux})$$

Initial states of all *e*-Services

DPDL encoding of target *e*-Service

DPDL encoding of *i*-th component *e*-Service

DPDL additional domain-independent conditions

DPDL encoding is polynomial in the size of the *e*-Service FSMs

## DPDL encoding

- Target  $e$ -Service  $S_0 = (\Sigma, S_0, s_0^0, \delta_0, F_0)$

in DPDL we define  $\Phi_0$  as the conjunction of:

- $s \rightarrow \neg s'$  for all pairs of distinct states in  $S_0$   
 *$e$ -Service states are pair-wise disjoint*
- $s \rightarrow \langle a \rangle \top \wedge [a] s'$  for each  $s' = \delta_0(s, a)$   
*target  $e$ -Service can do an  $a$ -transition going to state  $s'$*
- $s \rightarrow [a] \perp$  for each  $\delta_0(s, a)$  undef.  
*target  $e$ -Service cannot do an  $a$ -transition*
- $F_0 \equiv \bigvee_{s \in F_0} S$   
*denotes target  $e$ -Service final states*

- ...

## DPDL encoding (cont.d)

- Community *e*-Services  $\mathcal{S}_i = (\Sigma, S_i, s_i^0, \delta_i, F_i)$

in DPDL we define  $\Phi_i$  as the conjunction of:

- $s \rightarrow \neg s'$  for all pairs of distinct states in  $S_i$   
*e-Service states are pair-wise disjoint*
- $s \rightarrow [a](\text{moved}_i \wedge s' \vee \neg \text{moved}_i \wedge s)$  for each  $s' = \delta_i(s, a)$   
*if e-Service moved then new state, otherwise old state*
- $s \rightarrow [a](\neg \text{moved}_i \wedge s)$  for each  $\delta_i(s, a)$  undef.  
*if e-Service cannot do  $a$ , and  $a$  is performed then it did not move*
- $F_i \equiv \bigvee_{s \in F_i} S$   
*denotes community e-Service final states*

## DPDL encoding (cont.d)

- Additional assertions  $\Phi_{aux}$ 
  - $\langle a \rangle T \rightarrow [a] \bigvee_{i=1, \dots, n} moved_i$  for each action  $a$   
*at least one of the community e-Services must move at each step*
  - $F_0 \rightarrow \bigwedge_{i=1, \dots, n} F_i$   
*when target e-Service is final all comm. e-Services are final*
  - $Init \equiv s_0^0 \wedge_{i=1 \dots n} s_i^0$   
*Initially all e-Services are in their initial state*

**DPDL encoding:**  $\Phi = Init \wedge [u](\Phi_0 \wedge_{i=1, \dots, n} \Phi_i \wedge \Phi_{aux})$

# Results

**Thm:** Composition exists iff DPDL formula  $\Phi$  SAT

*From composition labeling of the target e-Service one can build a tree model of the DPDL formula and viceversa*

*Information on the labeling is encoded in predicates moved<sub>i</sub>*

$\Rightarrow$  Composition existence of e-Services expressible as FSMs is decidable in EXPTIME

# Results on Finite State Composition

**Thm:** If composition exists then Mealy composition exists.

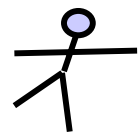
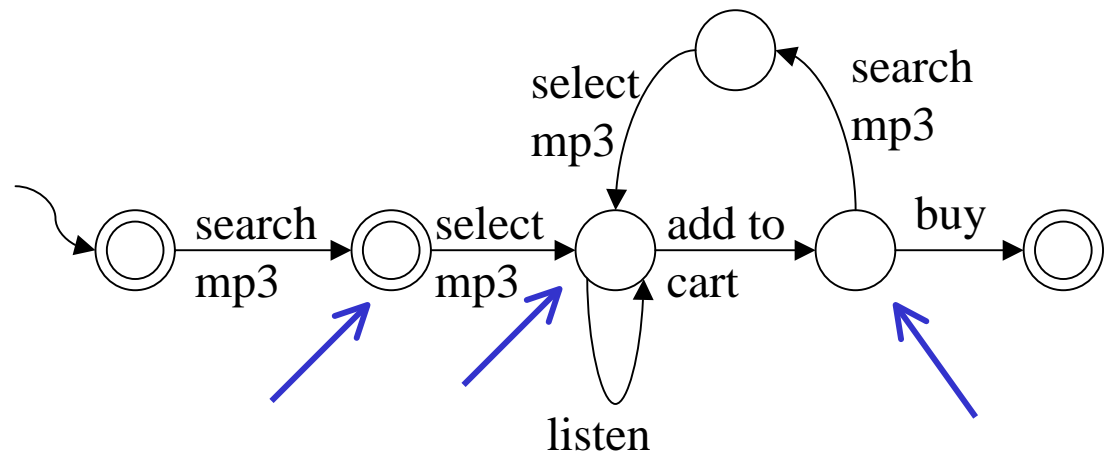
*From a small model of the DPDL formula  $\Phi$ ,  
one can build a Mealy machine*

*Information on the output function of the machine is encoded in  
predicates  $moved_i$*

$\Rightarrow$  Finite state composition existence of  $e$ -Services  
expressible as FSMs is decidable in EXPTIME

# Summary: The “Roman” Activity Based Model for e-Services

Service: on-line music store



Client



choice points: the *e-Service* makes **always** the **client decide** what to do next (in principle, all states can be choice points).



states at which client can stop



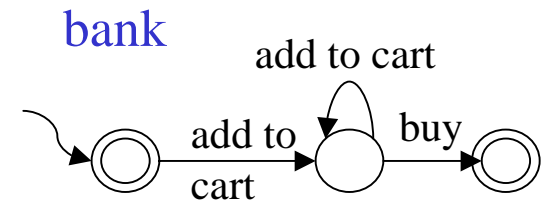
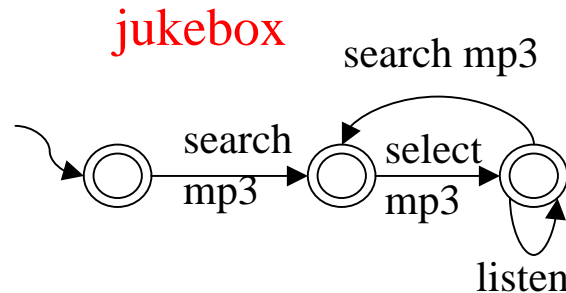
states at which client cannot stop

# Summary: Automatic *e*-Service composition in the “Roman” Framework

**But:** what if

- there does not exist an *e*-Service **on-line music store** ?
- the only available *e*-Services are **jukebox** and **bank**?

Community  
of *e*-Services:



# Summary: Automatic *e*-Service composition in the "Roman" Framework (cont.d)

Target *e*-Service (client request):  
on-line music store

Community of *e*-Services  
(available *e*-Services):

jukebox, bank

based on  
tableau  
techniques  
for DLs

***e*-Service Automatic  
Composition Engine**

Domain indep.  
constraints

Delegator (delegates  
each action of target *e*-  
Service to *e*-Service(s)  
in the community):

