The FLOWS* Proposal: Presentation to SWSL Committee

May 13, 2004

D.Berardi, M.Gruninger, R.Hull, S.McIlraith

*FLOWS = First-order Logic Ontology for Web Services [name subject to change]

Outline

- Representational Desiderata for a WSC ontology
- Features of our Proposal
  - FLOWS based on FOL Ontology of WS (PSL is the working hypothesis)
  - Identification of subsets of FLOW with attractive computational or representational properties
  - Surface syntax
  - Characterization of reasoning tasks within FOL
  - Reasonable computational strategy is critical
- Short-term Tasks
- Case Studies
  - Amazon example
  - Financial transaction example
  - Travel service scenario
  - WS Discovery (*new)
- Comparing with and bridging to other SWSL proposals

Representational Desiderata:

- Model-theoretic semantics **
- Primitive and complex processes are first-class objects ***
- Taxonomic representation *
- Leverages existing service ontologies (OWL-S) **
- Embraces and integrates with existing and emerging standards and research (BPEL, W3C choreography, etc.) *
- Explicit representation of messages and dataflow (cf. W3C choreography, behavioral message-based signatures, etc.) ***
- Captures activities, process preconditions and effects on world. *
- Captures process execution history. **

Legend

* we believe this feature is in the requirements document

** this feature represents a refinement of the requirements document

*** this feature represents an extensions to document

Features

1. FLOWS based on FOL ontology of WS

Working hypothesis: ontology based on PSL, a dialect of the situation calculus.

Analysis (FOL language):

+ provides a well-understood model-theoretic semantics

+ rich expressive power (e.g., variables, quantifiers, terms, etc.)

overcomes expressiveness issues that have haunted OWL-S

+ enables characterization of reasoning tasks in terms of classical notions of deduction, consistency, etc.

+ enables exploitation of off-the-shelf systems such as existing FOL reasoning engines and DB query engines.

- semi-decidable and intractable for many tasks (worst case) (but note that many intractable tasks often prove easily solved in practice)

Features (cont.)

Analysis (working hypothesis, PSL as a situation calculus dialect):

+ years of development in the business process modeling arena

+ well-established, already proven useful as exchange language

+ extensibility of PSL

+ first-stage characterization of OWL-S semantics

+ specific expressiveness properties:

 actions are first-class objects

 occurrence trees

 complex actions as first-class objects

 histories

 explicit representation of state

- readability and writability

- specific expressive properties:

 Ignores continuous change (though sitcal proposals exist)

- no implementation of associated reasoner

Features (cont.)

2. Identification of subsets of FLOWS with attractive representational or computational properties (e.g., decidability or tractability of certain reasoning tasks, traded-off against expressiveness)

<u>Examples:</u>

- OWL-S
- Situation Calculus and Golog for WSC
- DL for WSC
- Automata-theoretic approaches for WSC
- HTN planning for WSC
- Potential future mappings of other monotonic and nonmonotonic formalisms

Analysis (ID of subsets of FLOWS):

+ provides a theoretical mechanism for preserving semantics and relating different SWS ontologies

+ enables easy mapping to lite versions of ontology

+ provides basis for blending results about SWS origins in different methodologies (e.g., automata-based, DL-based, Petri-net based, sitcalc-based, etc)

Feature (cont.)

3. Surface Syntax (to be developed)

Analysis:

+ Makes FLOWS readable, easy to use and understand by end users

Features (cont.)

4. Characterization of SWS reasoning tasks in FOL.

E.g., WSC as deduction

Query-answering as deduction

WSC, reachability. liveness… as satisfiability

Analysis:

+ enables exploitation of off-the-shelf reasoners, algorithms and techniques

+ facilitates implementation

+ improves understanding of task

Features (cont.)

5. Computational strategy is key. (FOL theorem proving is not considered to be a viable option.) We would like to identify useful subsets of FOL with monotonic/nonmonotonic semantics, leveraging existing tools:

Candidates:
- Model-checking
- DL reasoners
- Prolog
- Answer-set programming, etc.
- Automata-theoretic techniques, verification tools

Analysis:

+ Exploitation of well-tested existing reasoners

Short-Term Tasks

- **Surface Syntax:** Develop a surface syntax
- **Computational Infrastructure:** Develop a (logic programming?) implementation, together with a working demo.
- **Concept Coverage:** Flesh out definition of concept coverage. At present, we envision this including:
  - all concepts in OWL-S (often represented differently to exploit our more expressive language)
  - other structure for individual services (e.g., automata-based) or compositions (e.g., WS-Choreography)
  - messages
  - dataflow
  - negotiation
- **Ontology:** Create a presentation of the entire ontology

Case Studies

- Amazon example
- Financial transaction example ✓
- Travel service scenario ✓
- WS Discovery (proposed)

Financial Transactions Use Case

- Embedding in PSL involves the following:
  - Subactivities
  - Partially ordered deterministic complex activities
  - Precondition axioms
    - Conditions on fluents that must hold before an activity can occur
  - Context-sensitive effect axioms
    - Effects of an activity occurrence can vary depending on fluents
  - Classes of activities denoted by terms (with parameters)
    - This capability not in OWL
- We illustrate how selected use-case assertions can be expressed in PSL
  - We rely on quantification over complex activities

Financial Transactions: Key Building Blocks

- Activities as terms

  $\forall$x activity(buy_products(x))
  $\forall$x,y,z activity(transfer(x,y,z))
  $\forall$x,y activity(withdraw(x,y))
  $\forall$x,y activity(deposit(x,y))

- Composition relationships

  $\forall$a,y (a = buy_product(y) $\supset$ $\exists$ x,z subactivity(transfer(x,y,z) , a))
  $\forall$x,y,z subactivity(withdraw(x,y), transfer(x,y,z))
  $\forall$x,y,z subactivity(deposit(x,z)), transfer(x,y,z))

- Process description for *buy_product*

  $\forall$o,x occurrance_of(o, buy_product(x)) $\supset$
  $\exists$o1,o2,y,z,w,v occurrence_of(o1, transfer(y,x,z)
  $\wedge$ occurrence_of(o2, transfer(w,x,v))
  $\wedge$ subactivity_occurrence(o1, o)
  $\wedge$ subactivity_occurrence(o2, o)

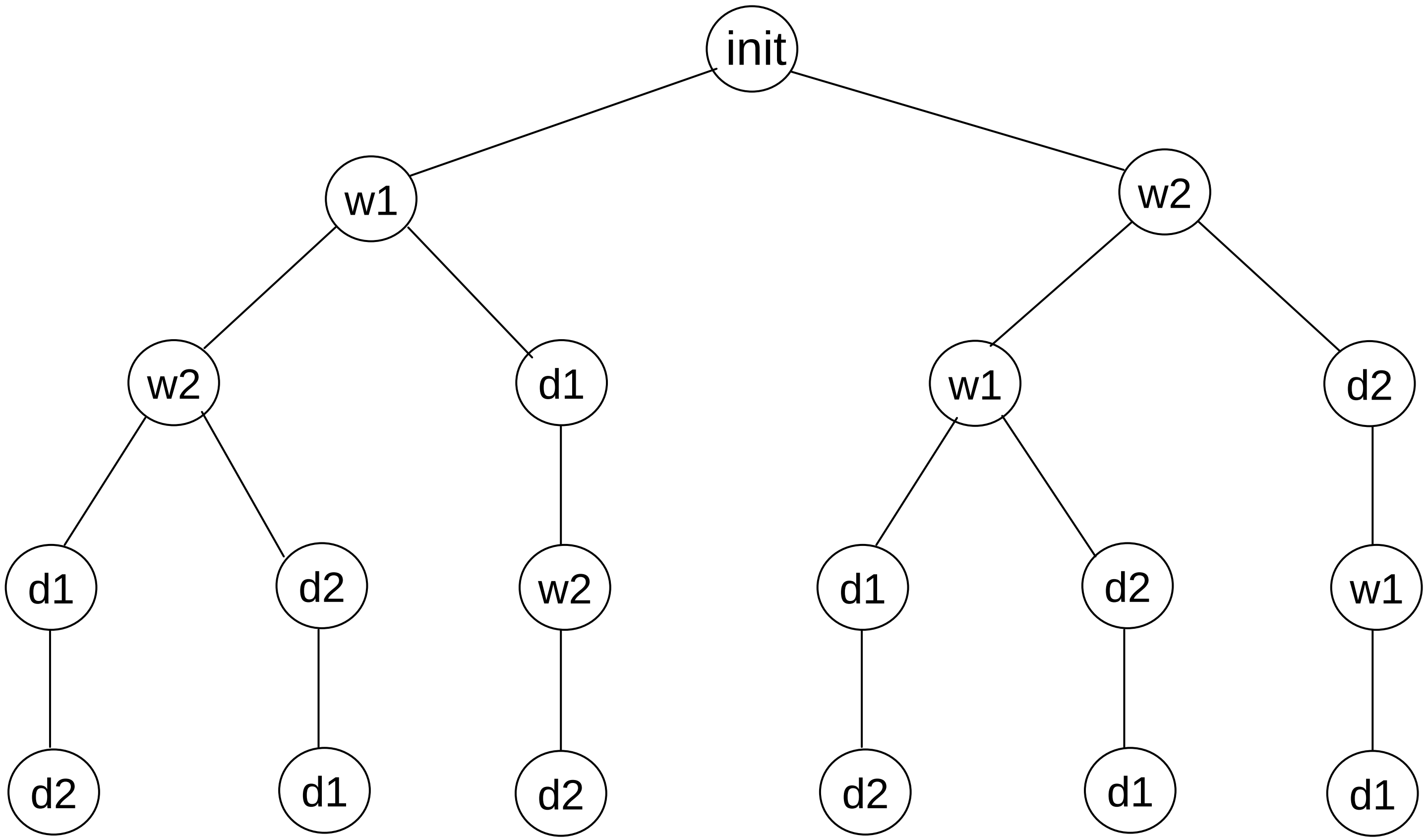
- Can represent
  – Other composite activities
  – Pre-conditions (e.g., transfers only if sufficient funds)
  – Effects (e.g., of a transfer)

Minimal activity tree

- Assume four atomic activity types

w1 = Withdraw (100, Account1)
w2 = withdraw (5, Account1)

d1 = deposit (100, Account2)
d2 = deposit (5, Account3)



Example assertion from Use Case

- Very preliminary sketch, to give basic idea
- Two transfers of X and Y are equivalent to one transfer of X+Y (between same accounts). But the fee is double.

∀ o1,o2 (
  equivalent(o1,o2) iff
  ∀ o3, o4, buyer, seller, broker, amount1, amount2, amount3, fee1, fee2, fee3
   (if occurrence_of (o1, double_transfer (buyer, seller, broker, amount1, fee1, amount2, fee2)
        ∧ subactivity_occurrence (o3, o1)
        ∧ subactivity_occurrence (o4, o1)
        ∧ subactivity (transfer(buyer, seller, amount1), o3)
        ∧ subactivity (transfer(buyer, broker, fee1), o3)
        ∧ subactivity (transfer(buyer, seller, amount2), o4)
        ∧ subactivity (transfer(buyer, broker, fee2), o4)

        ∧

      occurrence_of (o2, merged_transfer(buyer, seller, broker, amount3, fee3)
      ∧ subactivity(transfer(buyer, seller, amount3), o2) and
      ∧ subactivity(transfer(buyer, broker, fee3)), o2)

    then amount3 = plus(amount1, amount2) ∧ fee3 = plus(fee1, fee2)
)

Another assertion from Use Case

- Very preliminary sketch, to give basic idea
- Multiple international money transfers on the same account are not executed in parallel by bank B unless the costumer has a long-lasting relationship with bank B

∀ o1, o2, account, account1, account2, amount1, amount2 (
  if occurrence_of (o1, transfer(account, account1, amount1))
    ∧ occurrence_of (o2, transfer(account, account2, amount2))
    ∧ "o1 is international"
    ∧ "o2 is international"

  then precedes(o1, o2) or precedes(o2, o1)

Travel Use Case

An example of rich services and rich composition

- Atomic and non-atomic (fsa-based) “base” services
- Sequential and interleaved composition
- Activities and messages in one framework

Three services

- Different kinds of users want the services called in different orders
  - E.g., tourist wants hotel; plane; event

We illustrate how PSL can express 3 perspectives:

1. Atomic / SingleUse (cf OWL-S)
   - View each service as atomic
   - Create composite service for one use only
2. Interactive / generic re-usable (cf Roman model)
   - View each service as activity-based fsa
   - Create re-usable composite service targeted to any user
3. Blending of activity-based and message-based
   - View message send/receive as activities
   - Record message contents in predicate-based fluents
   - Can describe data flow, track history

book_hotel

Prec: hotel_booked = false

Input: hotel_city, date_arrive, date_back

Output: name_hotel, hotel_booking_id

Eff: hotel_booked = true

book_plane

Prec: plane_booked = false

Input: depature_city, date_leave, arrival_airport, date_back

Output: ticket_plane_id

Eff: plane_booked = true

register_event

Prec: event_booked = false

Input: event_name,

Output: start_attend_date, end_attend_date, registration_id, city_nearby_hotel, nearby_airport

Eff: event_booked = true

1. Atomic eService/SingleUse composition (sketch)

- Building composite activity “Maria_serv” for tourist Maria
- Specify that the three atomic services are in sequence; include simple exception handling
- (Selected) fluents:*booked_xxx*, *Success_xxx_booking*, *Fail_xxx_booking*

// establish sub-activity structure for Maria_serv

subactivity(launch, Maria_serv) ∧ subactivity(book_hotel, Maria_serv) ∧
subactivity(book_plane, Maria_serv) ∧ subactivity(register_event, Maria_serv)

// characterize all possible occurrances of Maria_serv (i.e., all paths in activity tree for Maria_serv)

∀x. occurrence_of (x, Maria_serv) ⇔

// exists a root atomic occurrance and atomic occurrance of book_hotel activity

(∃o1 occurrence_of(o1,book_hotel) ∧ subactivity_occ(o1, x) ∧ root(o0,x) ∧

(if ¬ (prior(*Precond_hotel*, o1) ∧ prior(*Input_hotel*, o1))
then (holds(*Failure_hotel_booking* , o1) ∧ leaf_occurrence(o1, x))
else (holds(*Eff_hotel*, o1) ∧ holds(*success_hotel_booking*,01) ∧

// if the book_hotel occurrance succeeded, then there is also an occurrance of book_plane

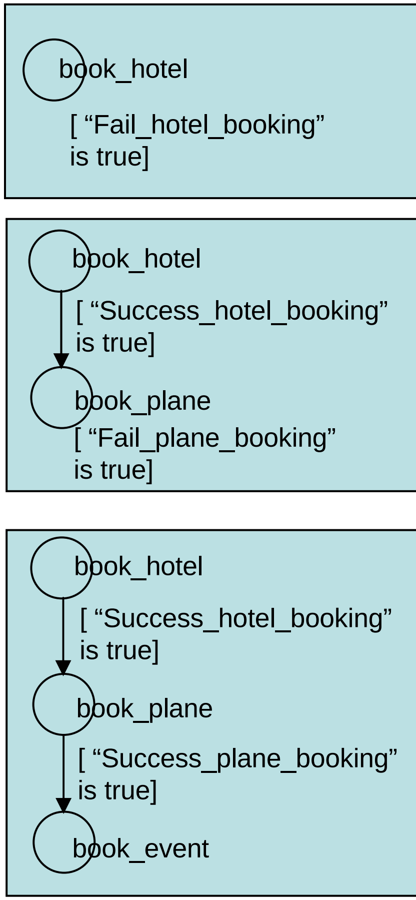
∃ o2. ocurrence_of(o2, book_plane) ∧ subactivity_occ(o2, x) ∧ next_subocc(o1, o2, x)
(if ¬ (prior(*Precond_plane*, o2) ∧ prior(*Input_hotel*, o2))
then (holds(*Failure_plane_booking*, o2) ∧ leaf_occurrence(o2, x))
else (holds(*Eff_plane*, o2) ∧ holds(*Success_plane_booking*, o2) ∧

// if the book_plane occurrance succeeded, then there is also an occurrance of register_event

∃ o3. occurrence_of(o3, register_event) ∧ subactivity_occ(o3, x) ∧ next_subocc(o2, o3, x) ∧
(if ¬ (prior(*Precond_event*, o3) ∧ prior(*Input_event*, o3))
then (holds(*Failure_event_booking*, o3) ∧ leaf_occurrence(o3, x)
else (holds(*Eff_event*, o3) ∧ holds(*Success_event_booking*, o3) ∧ leaf_occurrence(o3, x))))))))

// some notational short-hand

Precond_hotel ⇔ ¬ booked_hotel; *Eff_hotel* ⇔ booked_hotel; ...similar for plane and event

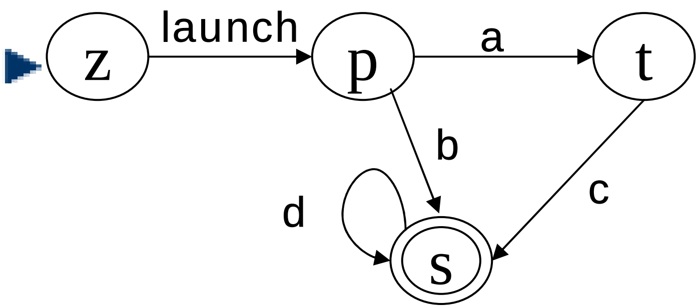


The three activity trees (up to isomorphism) corresponding to composite activity Maria_serv as defined in green box. Maria_serv can be defined in a variety of ways, leading to different (sets of) activity trees

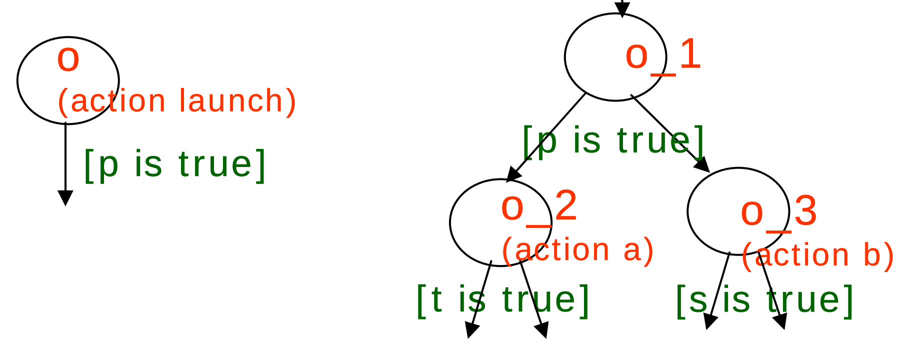
2a. Representing in PSL a complex process, whose internal structure corresponds to an activity-based FSA (sketch)

We illustrate the encoding using an abstract example

- Assume 1 fluent per state, assert that only one state-fluent can be true at a time
- We transform the fsa by adding a new start-state with "launch" activity



FSA M



Parts of (representative) "activity tree" for M

[This tree might be embedded into an "occurrence tree" which represents a family of concurrent activity occurrences]

$\varphi_M(x) =$ (

// initial situation $\exists$ o. occurrence_of(o, launch) $\wedge$ root(o,x) $\wedge$ holds(p, launch)

// for all transitions in FSA M include the following (the following example is for $\delta(p,a) = t$)

$\forall o_1, o_2$ if (subactivity_occurrence(o_1, x) $\wedge$ subactivity_occurrence(o_2, x) $\wedge$
next_subocc(o_1, o_2, x) then (holds(p, o_1) $\wedge$ occurrence_of(o_2, a) $\rightarrow$ holds(t, o_2))

// from a given atomic occurrance, there is at least one child for each transition out of the corresponding state, and no illegal transitions (the following is for atomic occurrance o_1 that corresponds to being in state p)

$\forall o_1$ if (subactivity_occurrence(o_1, x) $\wedge$ holds(p, o_1))
then $\exists o_2$ (subactivity_occurrence(o_2, x) $\wedge$ next_subocc(o_1, o_2, x) $\wedge$ occurrence_of(o_2, a)
$\wedge$ $\exists o_2$ (subactivity_occurrence(o_2, x) $\wedge$ next_subocc(o_1, o_2, x) $\wedge$ occurrence_of(o_2, b)
$\wedge$ $\neg \exists o_2$ (subactivity_occurrence(o_2, x) $\wedge$ next_subocc(o_1, o_2, x) $\wedge$ occurrence_of(o_2, c)

// for all final states include the following (the following example is for s in final states)
$\forall$o (if leaf_occurrence(o, x) $\rightarrow$ holds(s, o))

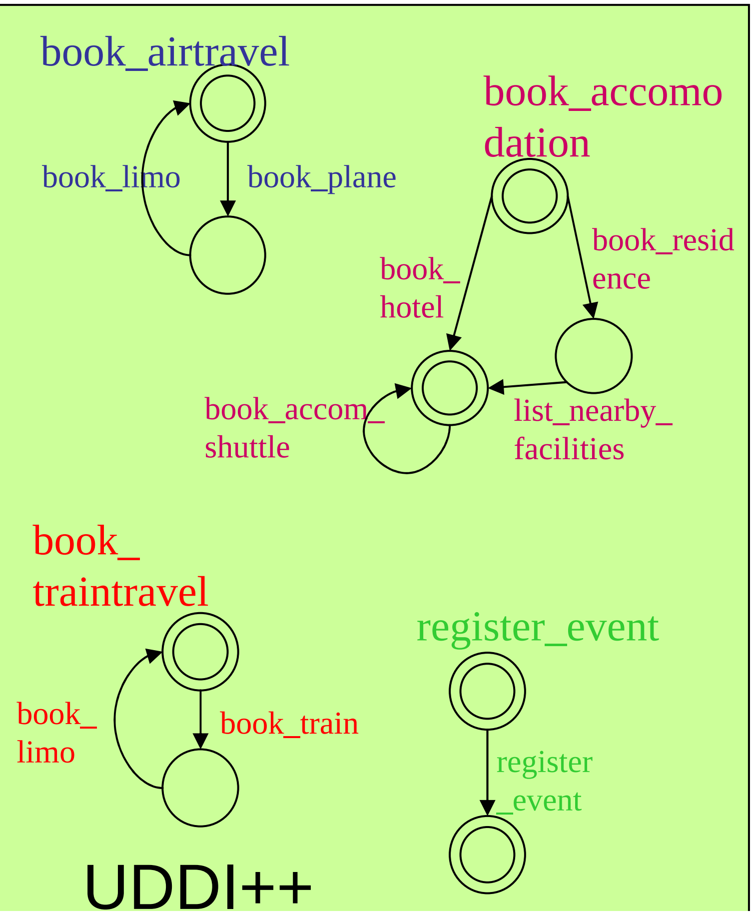
2b. Comments re embedding of FSA descrips into PSL

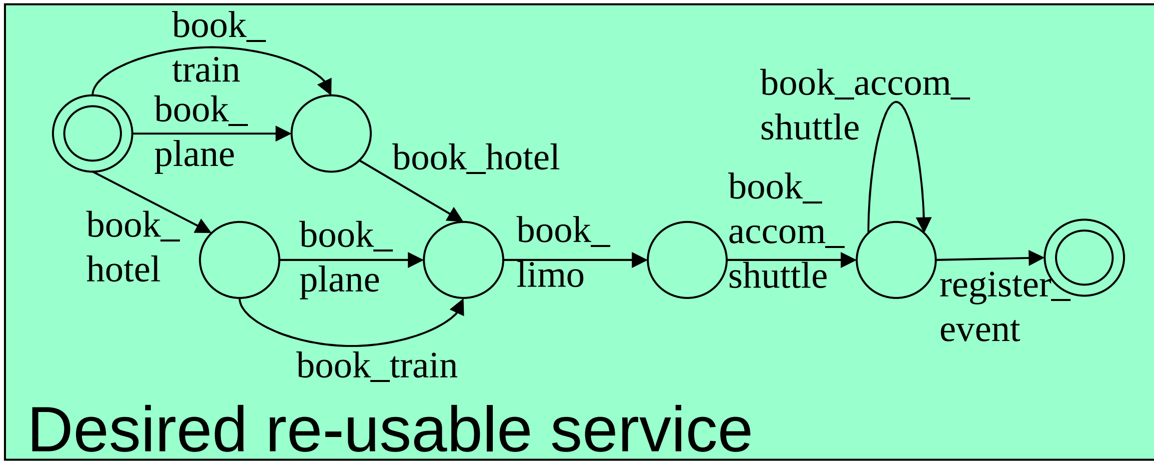
We have sketched a specific way to build up a formula $\varphi_M(.)$ as described informally on prevoius slide

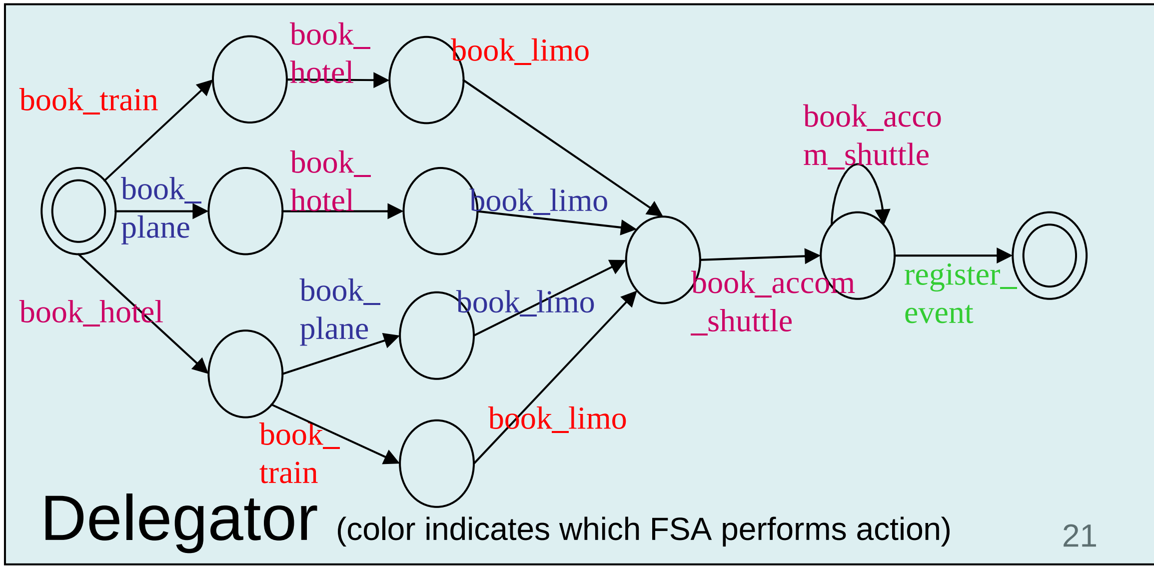
- *Conjecture ("Faithfulness"): x* satisfies formula $\varphi_M(x)$ iff *x* is an activity tree and there is a mapping between accepted words of M and finite branches of *x*.
  - For each word *w* in L(M) at least one finite branch with actions corresponding to *w*
  - For each finite branch β satisfying appropriate fluents at the end, there is a word in L(M) corresponding to β
- Can build similar formula $\chi(x)$ characterizing a single path through the activity tree for M, i.e., (finite branch) *x* satisfies $\chi(x)$ iff *x* corresponds to an accepted word of M
- Can build similar formula $\Psi_M(x,z)$ stating that *x* is the activity tree of M embedded into the occurrence tree *z*
- Given a UDDI+, can build a $\varphi_M(.)$ for each M in the UDDI+
  - Open problem: Can we reify the UDDI+ directory, and talk about member_of(*x*,*U*) ??
- Open problem (informal statement): Is there a "generic" first-order formula $\Gamma(\varphi_M(.), \varphi_N(.))$, such that for arbitrary fsa's M and N and associated formulas $\varphi_M(.)$ and $\varphi_N(.)$, we have $\Gamma(\varphi_M(x), \varphi_N(y))$ iff L(M) = L(N)
  - At a minimum, given fsa's M and N, you can by hand build a formula stating that M and N accept equiv languages

2c. Using automated composition to create re-usable, generic composition of interactive (fsa-based) services

- The base services for this example are richer than for previous example
- (We think that) we can encode multiple FSA's, and describe requirements for a composition (via delegator) to exist (in spirit of "Roman" results)

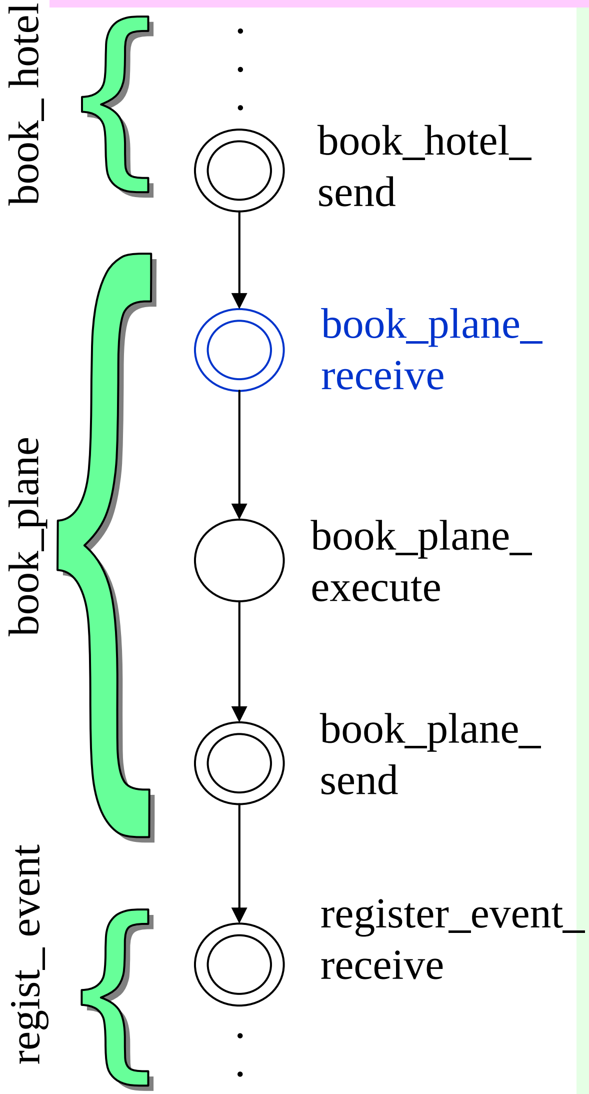






3a. Message Passing between atomic services (illustration in very simple context)

- book_plane assumed to have 3 sub-activities: _receive, _execute, _send
- Use predicate-based fluent “mess_repos(service_name, message_variable)” to hold messages being passed to a service



μ(x) ⇔

// basic structure of book_plane

occ_of(x, book_plane) ∧

∃ o1, o2, o3 (sub_act(o1, x) ∧ sub_act(o2, x) ∧ sub_act(o3, x) ∧
 occ_of(o1, book_plane_rec) ∧
 occ_of(o2, book_plane_exec) ∧
 occ_of(o3, book_plane_send) ∧

// “glue” between book_hotel and book_plane

(∃o4 o5 occ_of(o5, reg_event) ∧ sub_act(o4, o5) ∧
occ_of(o4, reg_event_send) ∧ leaf_occ(o4, o5) ∧
next_subocc(o1, o4)) ∧

// reading from message repository

(∃m’, v’, m’’, v’’, m’’’, v’’’, m’’’’, v’’’’
 (prior (mess_repos(book_plane, m’), o1) ∧
 mess_type(m’, departure_city) ∧ mess_value(m’, v’) ∧
 ¬holds(mess_repos(book_plane, m’), o1) ∧
 ... /* similar for m’’, m’’’, m’’’’ */) ∧

// execution of book_plane_execute ...

// sending messages to regist_event ...

// “glue” between book_plane and register_event

3b. Expressing Constraints on Data Flow

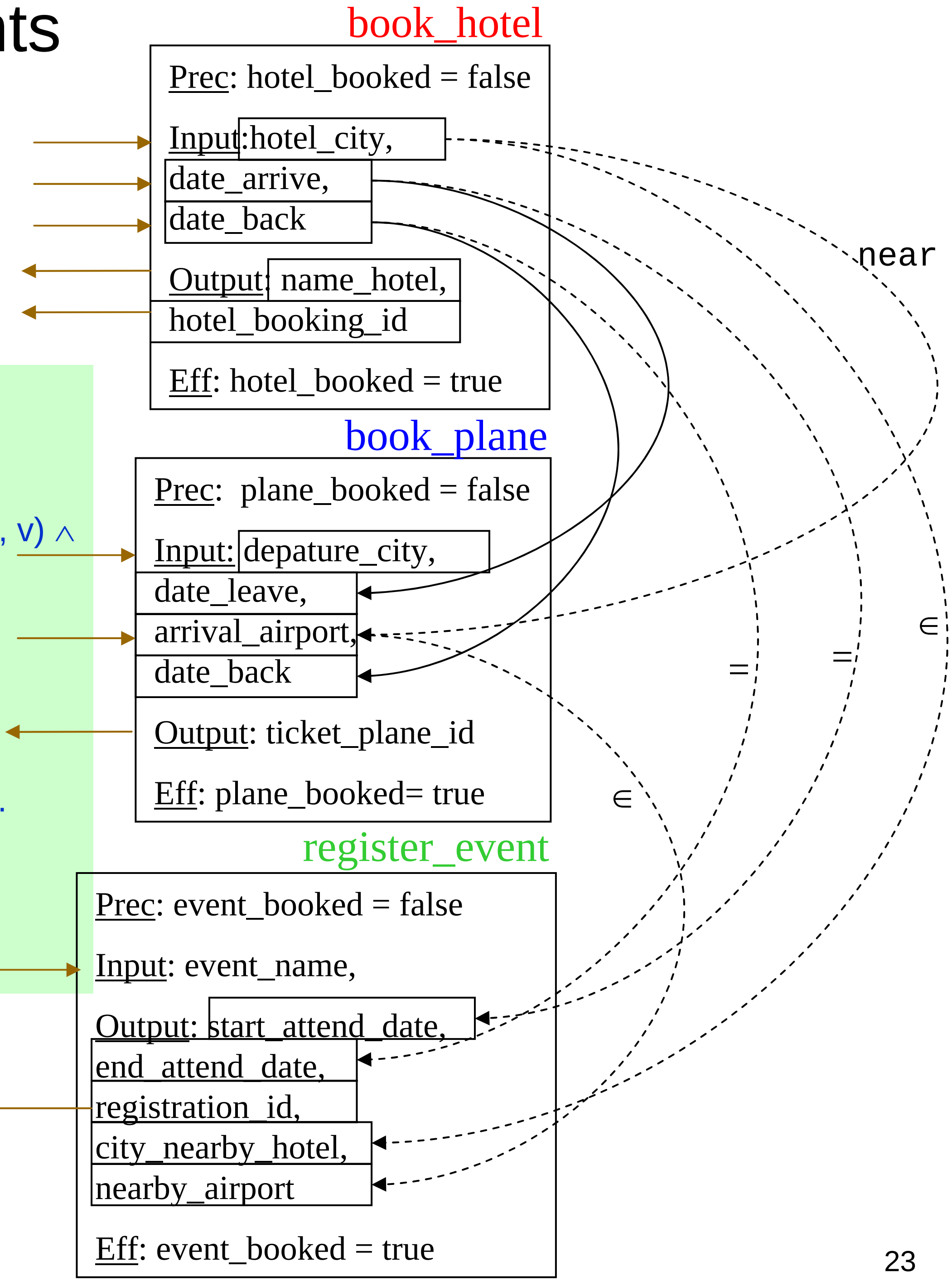
- Can express variety of data flow constraints
- Assume the 3 atomic services as on previous slide

// Values passed from book_hotel to book_plane

o is occ of composite service
o1 is occ of book_plane_receive …
$\exists$i, m, v (input_type(i, date_arrive) $\wedge$ input_value(i, v) $\wedge$
mess_type(m, date_leave) $\wedge$ mess_value(m, v) $\wedge$
prior(mess_repos(comp_service, i), o) $\wedge$
prior (mess_repos(book_plane, m), o1)

// Constraint between input values

o is occ of composite service
o1 is occ of book_hotel; o2 is occ of book_plane …
$\exists$i, i', v,v' (
input_type(i, date_arrive) $\wedge$ input_value(i, v) $\wedge$
input_type(i', date_leave) $\wedge$ input_value(i', v') $\wedge$
element_of(v, v')



Legend
- data in/out of composite service
- data flow within composite service
- constraint on data flowing within composite service