Bell Labs
Innovations for Lucent Technologies

# Web Services Composition:

## A Story of Models, Automata and Logics

<< Version of Sept 10 >>

Richard Hull

Bell Labs, Lucent Technologies

September 9-10, 2004

Presented at the EDBT 2004 Summer School

Sardinia, Italy

**Based on the Hull-Su SIGMOD 2004 Tutorial, and some other things**

**Based in part on discussions and collaborations with several people, including Michael Benedikt, Daniela Berardi, Tefvik Bultan, Diego Calvanese, Vassilis Christophides, Guiseppe De Giacomo, Michael Gruninger, Grigorios Karvounarakis, Maurizio Lenzerini, Massimo Marcella, Sheila McIlraith, and Jianwen Su**

# Web Services (a.k.a. E-services)

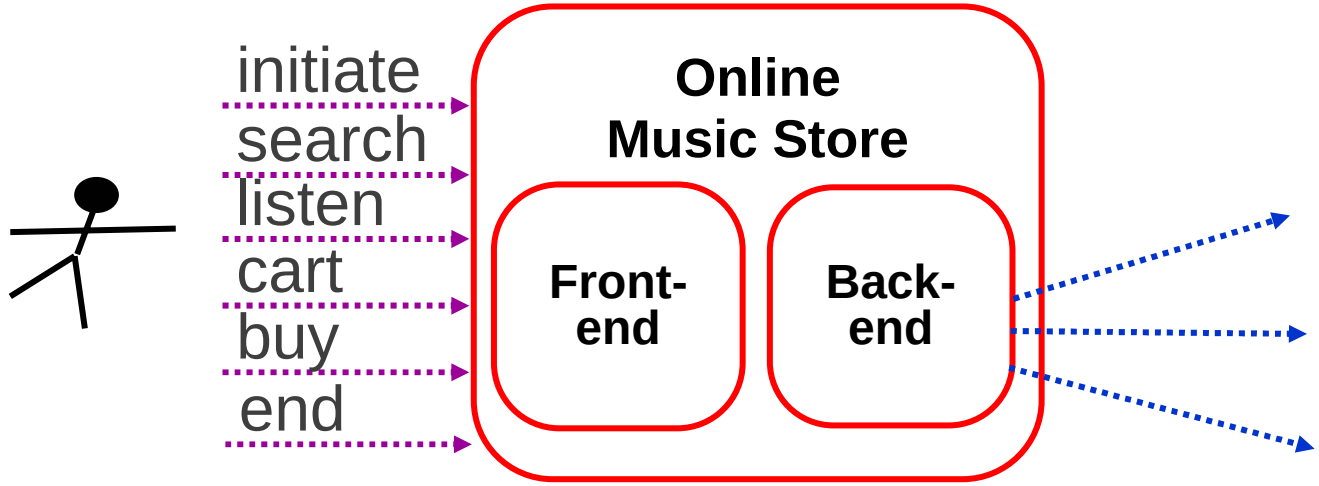
- The Web: Flexible human-machine interaction
- Web Services: Flexible machine-machine interaction
- Working Definition: Network-resident software services accessible via standardized protocols
  - Simple Object Access Protocol (SOAP): very flexible remote procedure call

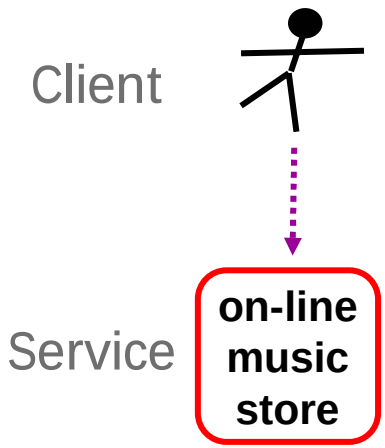
- Lots of interest in trade press, academic community, standards bodies, . . .

- Applications in e-commerce, telecom, science, GRID, government, education, . . .

# E-commerce: Front-end

- Example of customer buying CDs



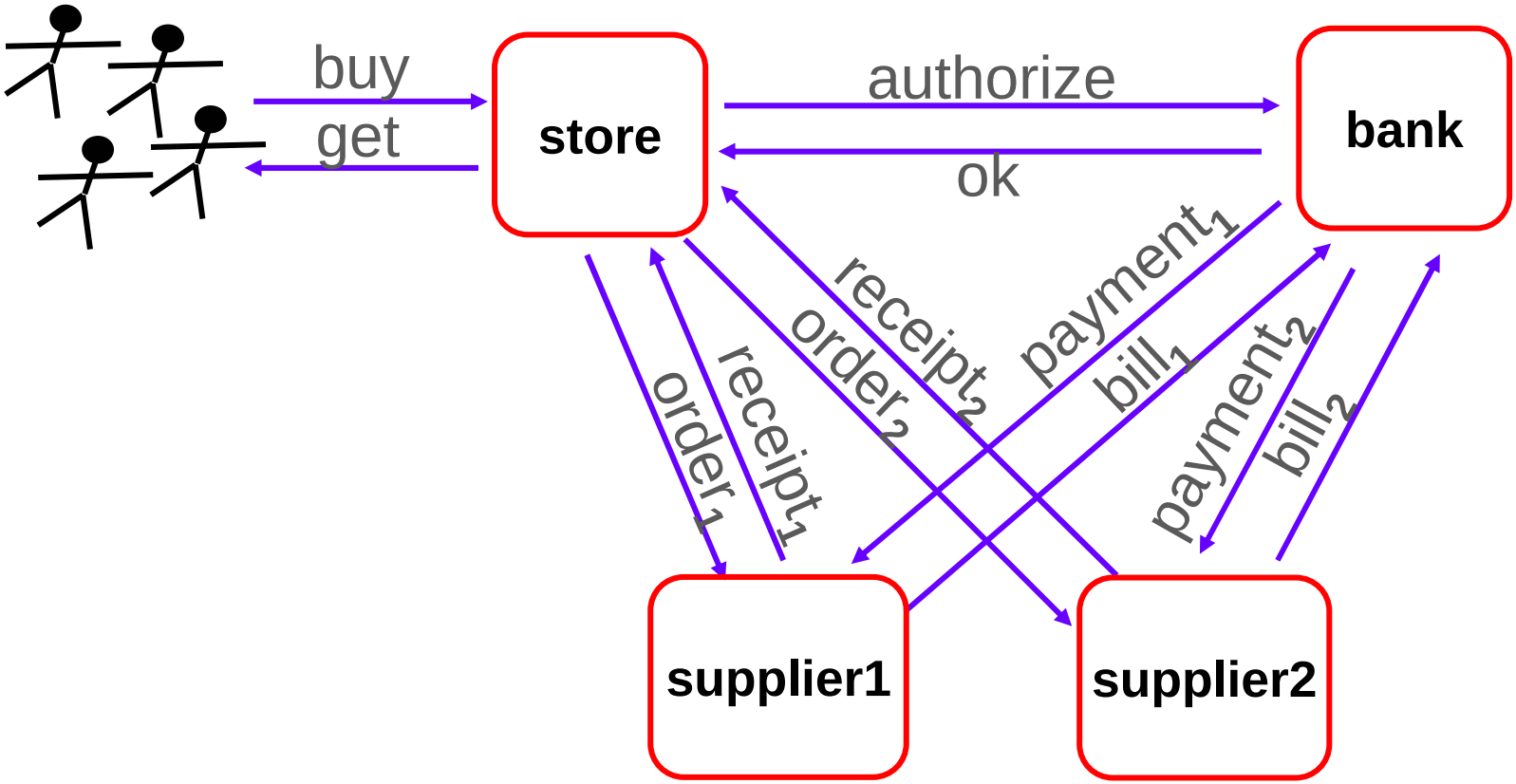




Abstract behavior of the Service:

Do until Client selects “End”

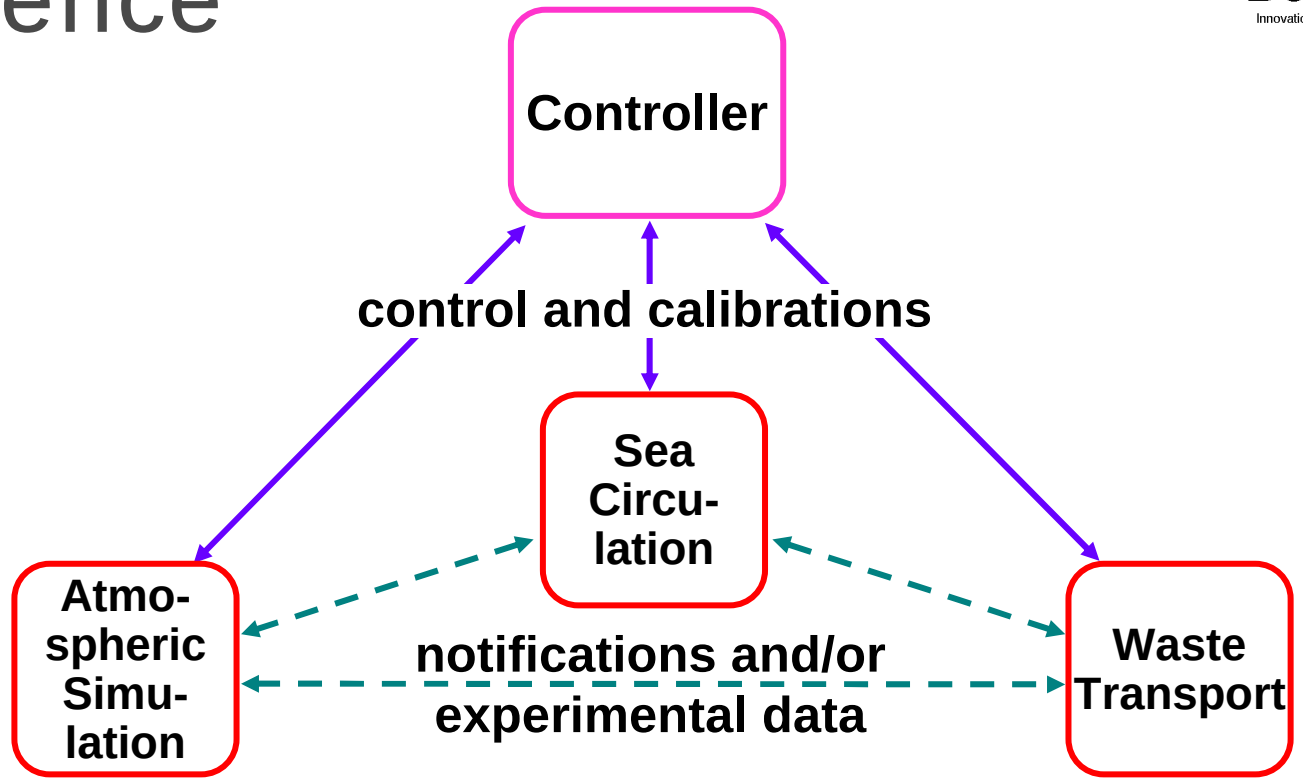
1. Give Client a choice of actions to be performed
2. Wait for Client choice
3. Perform action chosen by Client

# E-Commerce: Back-end



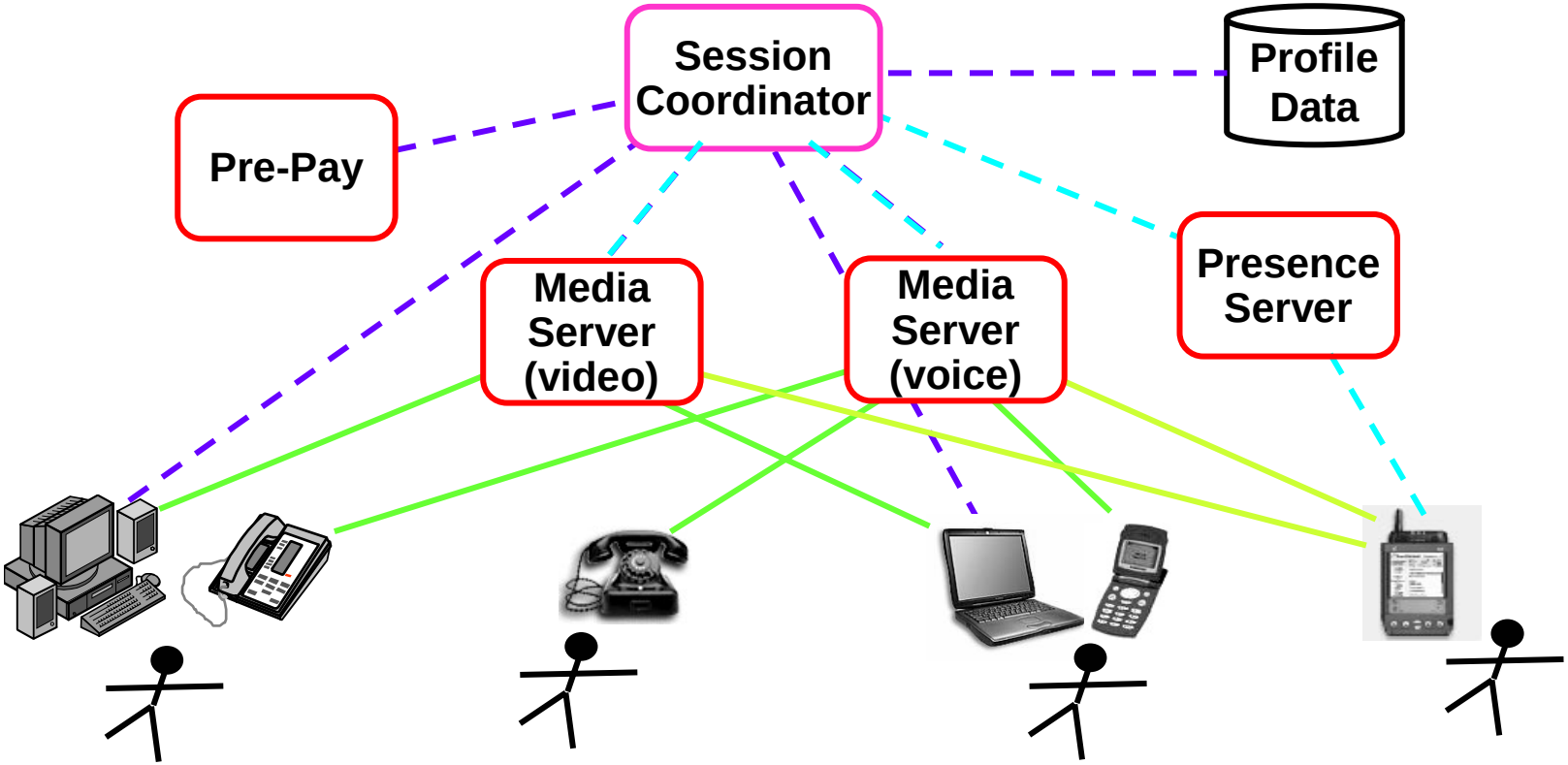

- E.g., for inventory management
- Web services paradigm driven by e-commerce
- Add new services dynamically

# E-Science




- E.g., find best location for waste treatment plant
- Possibly 100s of nodes, and running for weeks
- Data size difference: calibrations (small) and experimental data (large)
- Provenance: need to access derivation history

# Telecom Services




- Future: flexible, dynamic incorporation of features
- Emerging standards enable de-coupling (SIP, 3GPP IMS)
- Bearer traffic vs. signaling/control traffic
- Emphasis on asynchronous events “at any time”
- Convergence of telecom and web services

# Web Services: The Big Questions

Simplify and/or automate web service

- Discovery
  - What properties should be described?
  - How to efficiently query against them?
- Composition
  - Specifying goals of a composition
  - Specifying constraints on a composition
  - Building a composition
  - Analysis of compositions
- Invocation
  - Keeping enactments separated
  - Providing transactional guarantees
- Monitoring
  - How to track enactments
  - Recovering from failed enactments

Primary focus of this talk

# Why automated composition

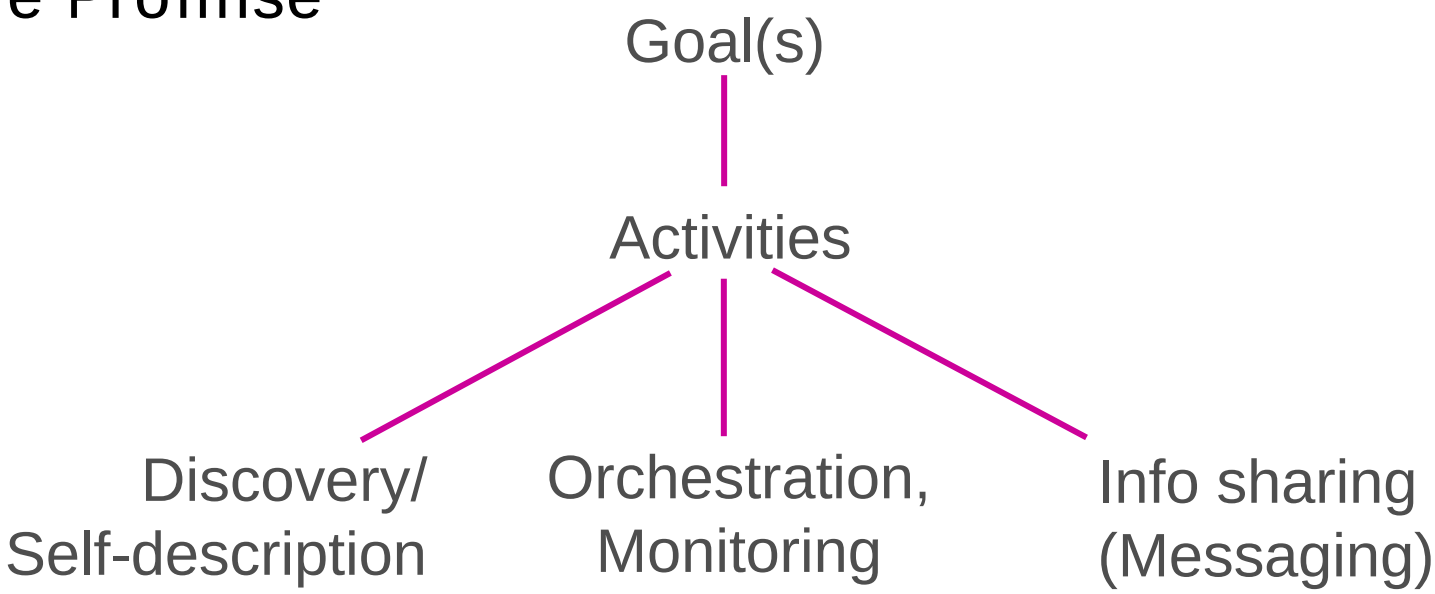
Bell Labs
Innovations for Lucent Technologies

- Composition "on-the-fly" will enable flexible use of vast numbers of services
  - E.g., pick the best services for your immediate need
  - E.g., get the job done even if your favorite component service is unavailable
- "Simple" forms of composition
  - Re-use of previous compositions
  - Customization of a generic composition
- Study of composition will give insight and aids for manual composition, e.g.,
  - Richer ways to describe, discover web services
  - Tools to better debug BPEL specs

# Anatomy of Web Services Composition

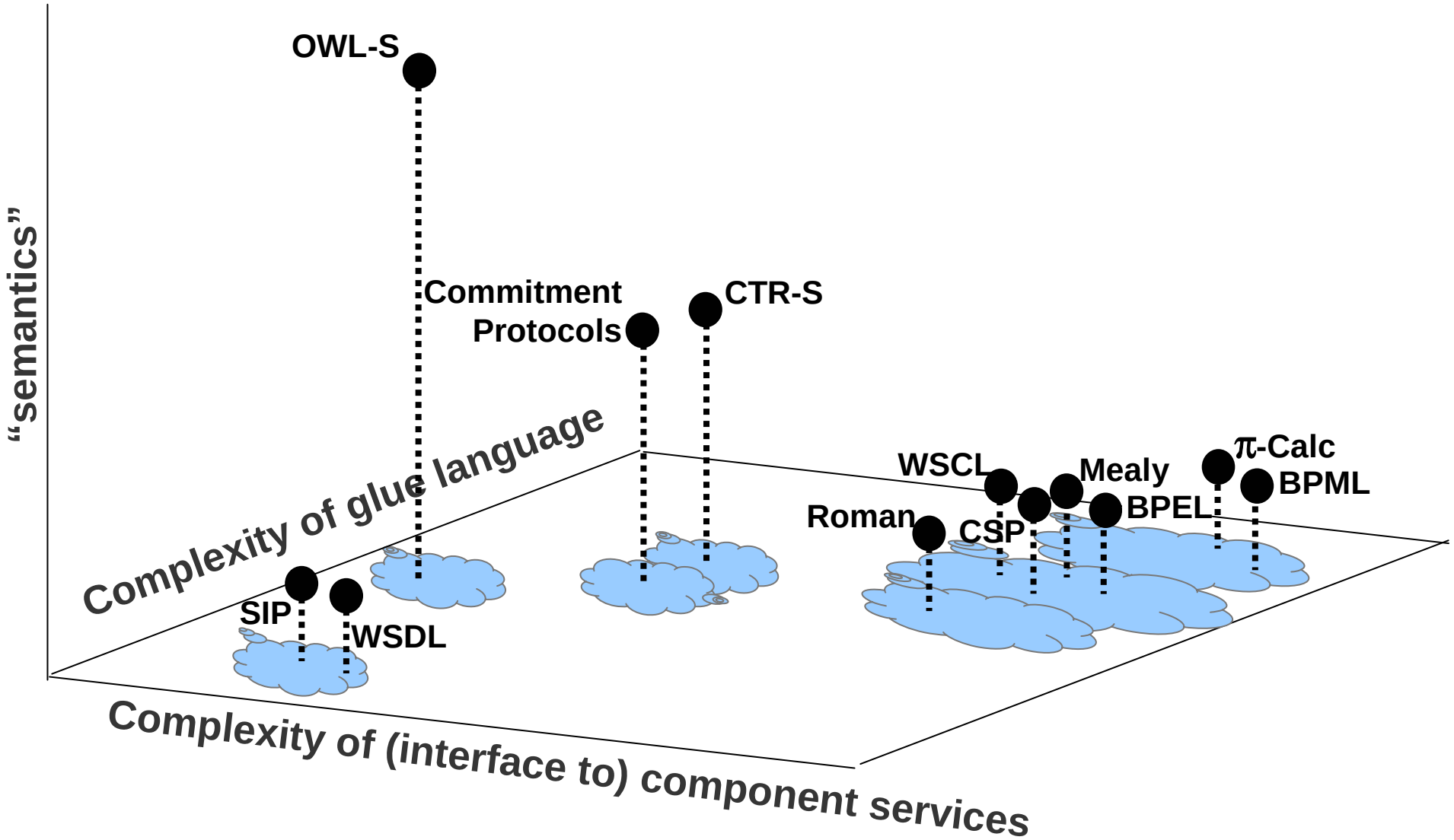
Bell Labs
Innovations for Lucent Technologies

- The Promise




- The reality: No unifying approach has emerged yet
  - BPEL, Mealy: Strong on orchestration, info sharing
  - OWL-S: Strong on goals, activities, discovery
  - “Roman” model: Strong on activities, orchestration

# Key dimensions in web service composition



# Key Disciplines for Web Services Composition

Bell Labs
Innovations for Lucent Technologies

- Science of Design, e.g.,
  - Workflow, Service-oriented computing, Reactive systems, Process algebras, Automata, Verification
- AI, e.g.,
  - Knowledge Representation, Reasoning about actions, Agents, Planning
- Logics and Finite Model Theory, e.g.,
  - Temporal Logics, Propositional Dynamic Logic, Description Logics, Situation Calculii, Verification
- Databases, e.g.,
  - XML, Domain-specific query languages, Indexing, Transaction Management

# Goals of this Talk: Tools for Composition

Bell Labs
Innovations for Lucent Technologies

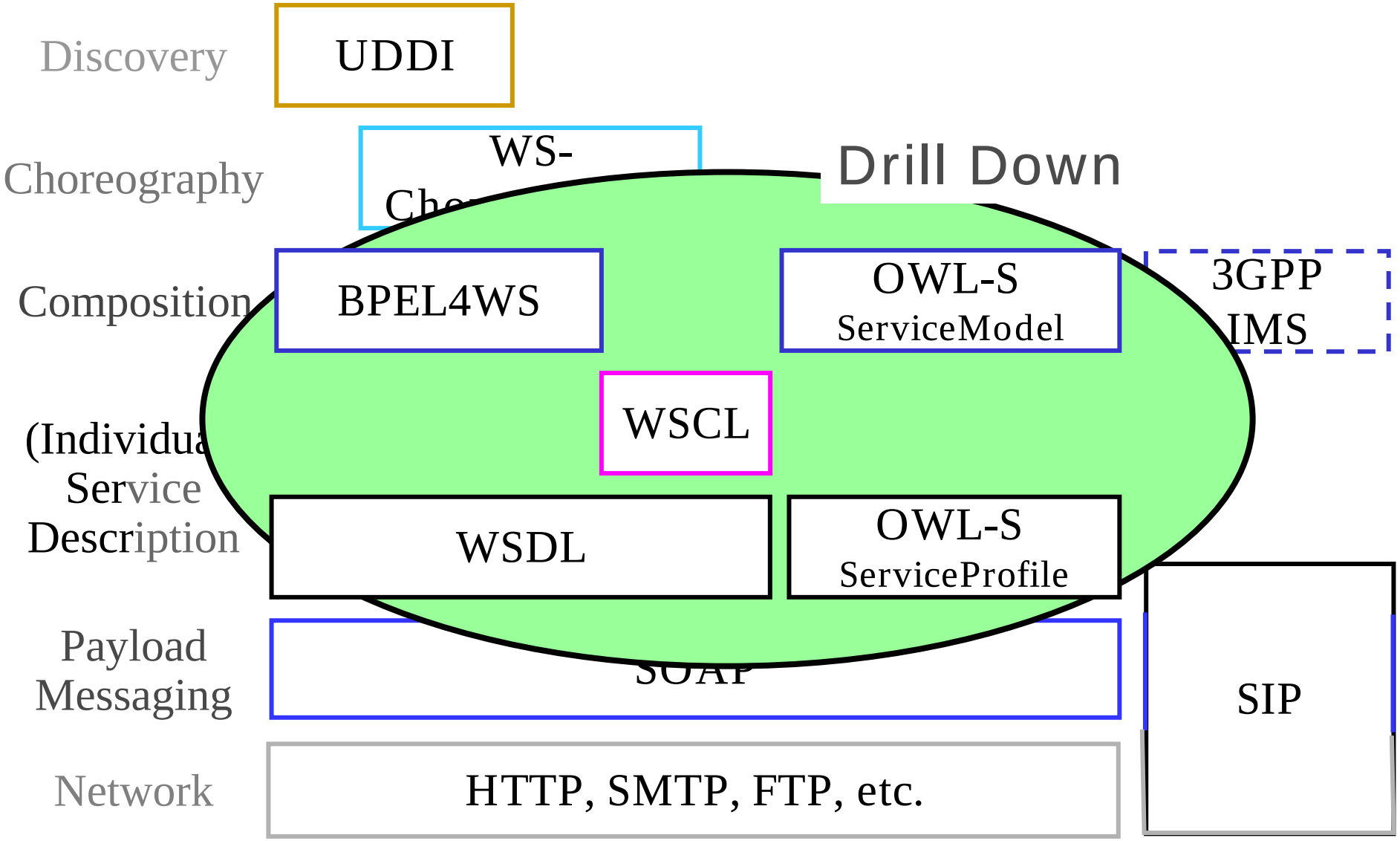
- Describe the playing field
  - Key standards that we'll all build upon
  - Promising models and formalisms
- (Selected) Perspectives and Results on Composition
  - Automata-theoretic approaches
    - Digression into targeted logics
  - Incorporating "semantics": The OWL-S community, PSL
  - Representative results from verification (if there's time)
  - Towards a model for coordinating telecom services
- The key challenge going forward
  - Some foundations are laid, but how to unify them

# Outline

Bell Labs
Innovations for Lucent Technologies

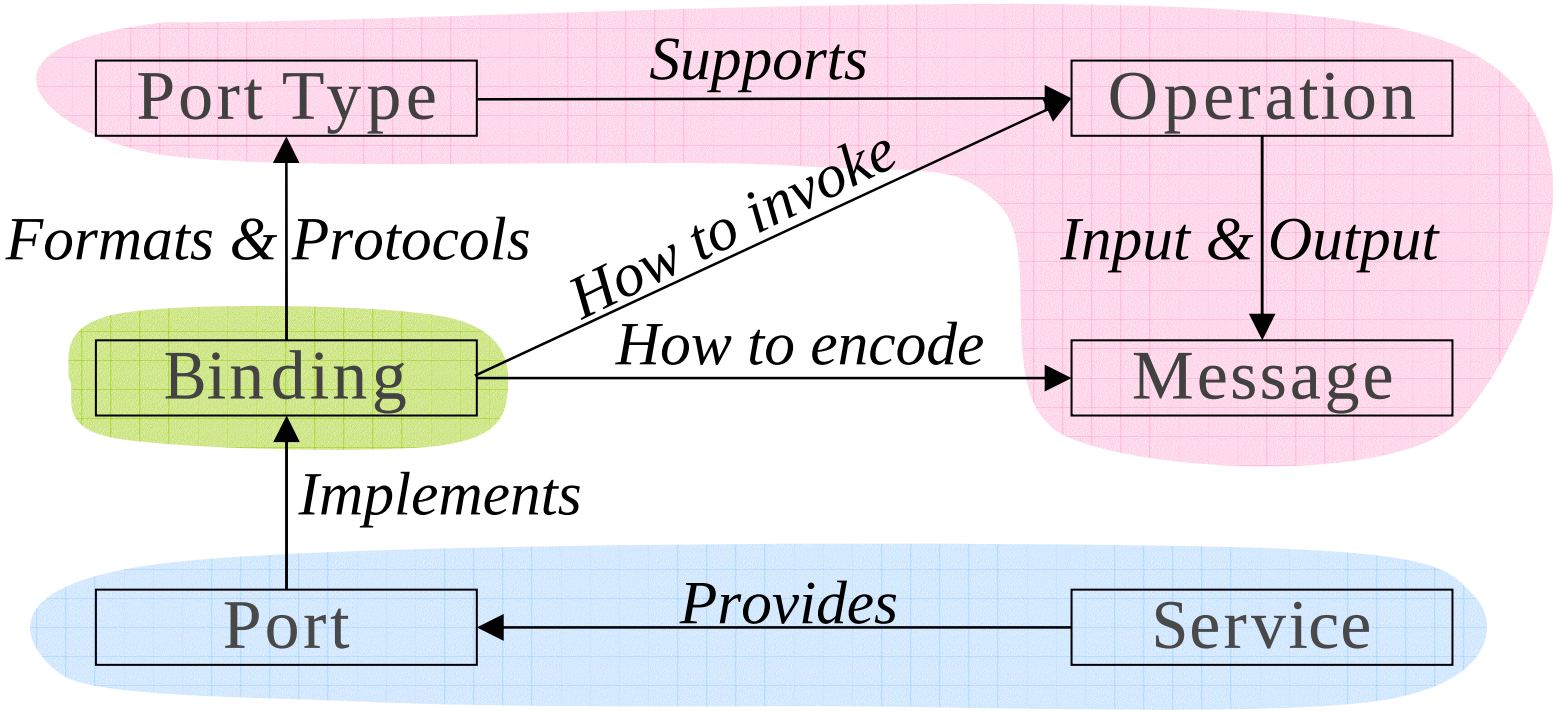
- Introduction
- Standards
- Perspectives on Composition
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# Web Services Standards Stack: Key Elements



# Web Service Definition Language (WSDL)

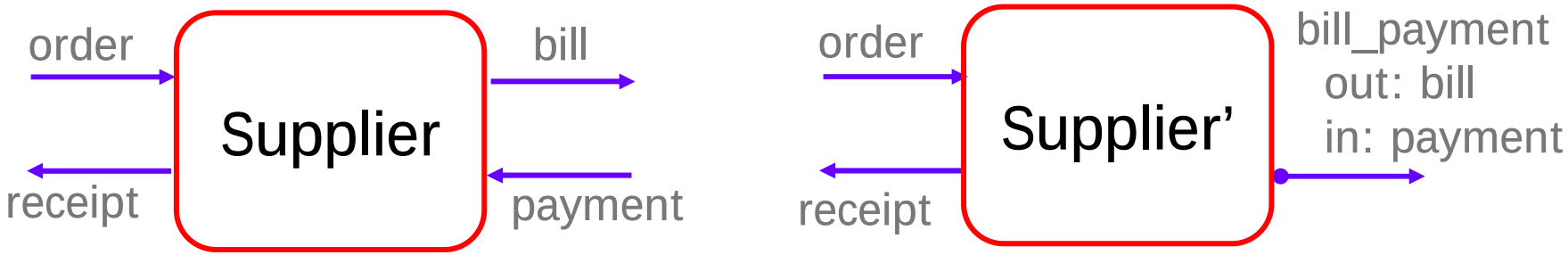
- WSDL provides a framework for defining
  - Interface: operations and input/output
  - Access specification: SOAP bindings (e.g., RPC)
  - Endpoint: the location of service




[from Leymann BTW 2003 talk]

# WSDL 1.0 Operations

- Traditional I/O signatures (using XML Schema)
- Four operation types
  - Proactive : send request
    send request, block till response
  - Reactive : receive request
    receive request, send response




- Port: mechanism to cluster operations
  - Port as unit of interoperation between services
- WSDL 2.0: several more operation types

# Business Process Execution Language (BPEL)

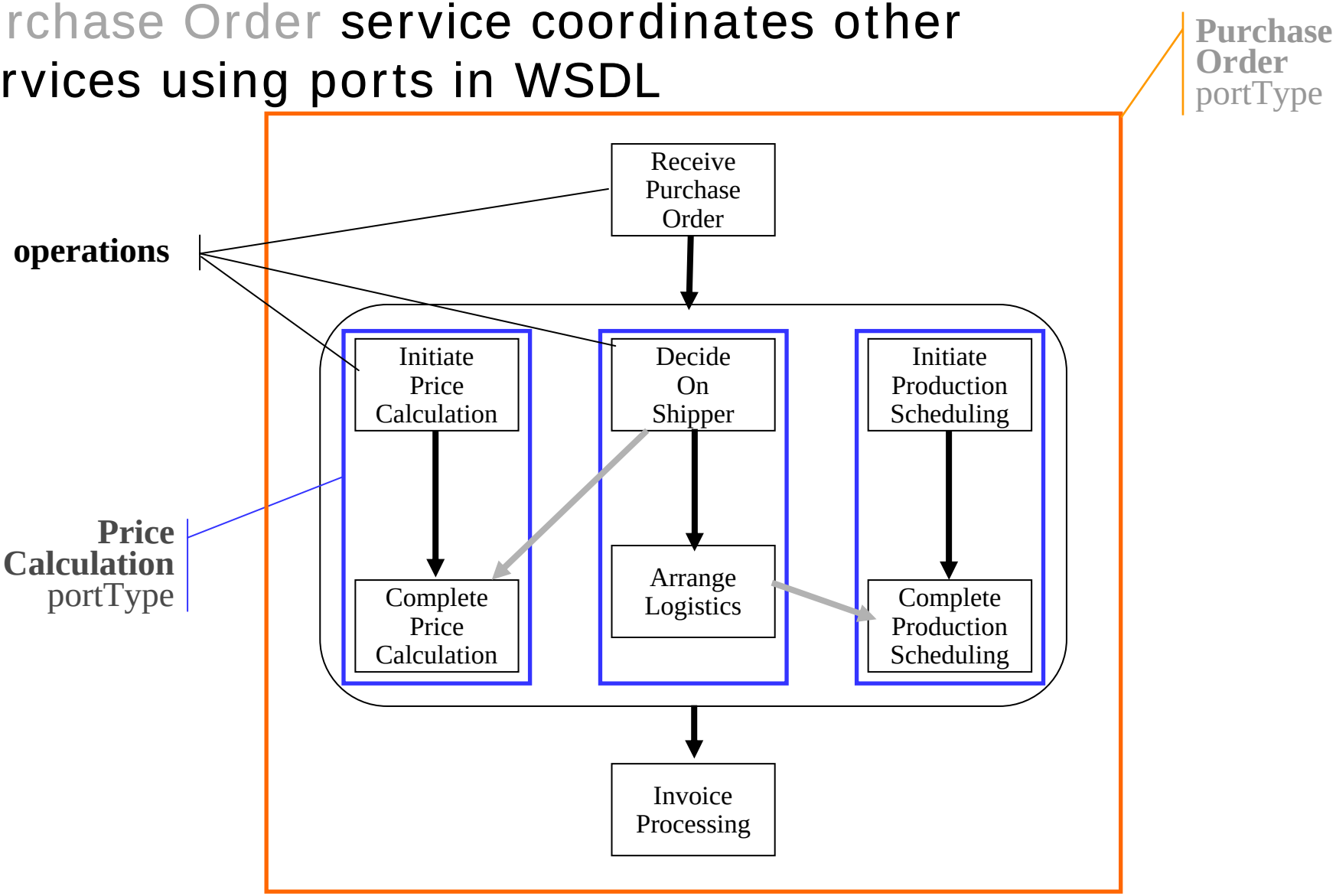
Bell Labs
Innovations for Lucent Technologies

- Allow specification of compositions of Web services
  - business processes as coordinated interactions of Web services
- Allow “abstract” and “executable” processes
- Influences from
  - Traditional flow models
  - Structured programming
  - Successor of WSFL and XLANG
- Assumes WSDL ports

- Standardization through OASIS

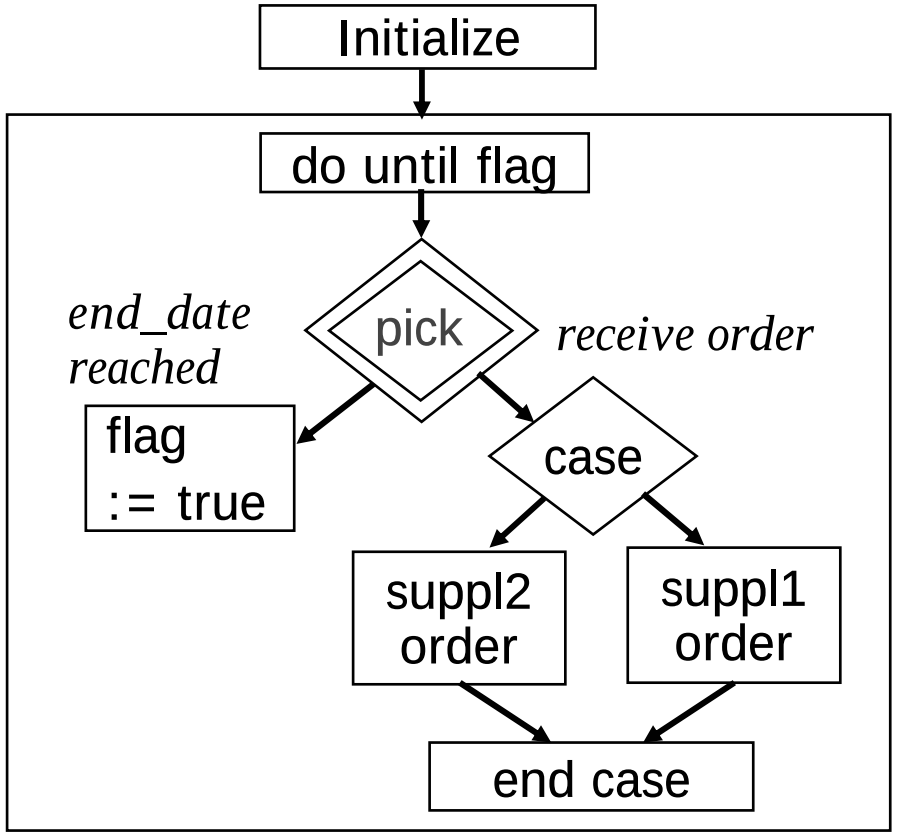
# BPEL in Action

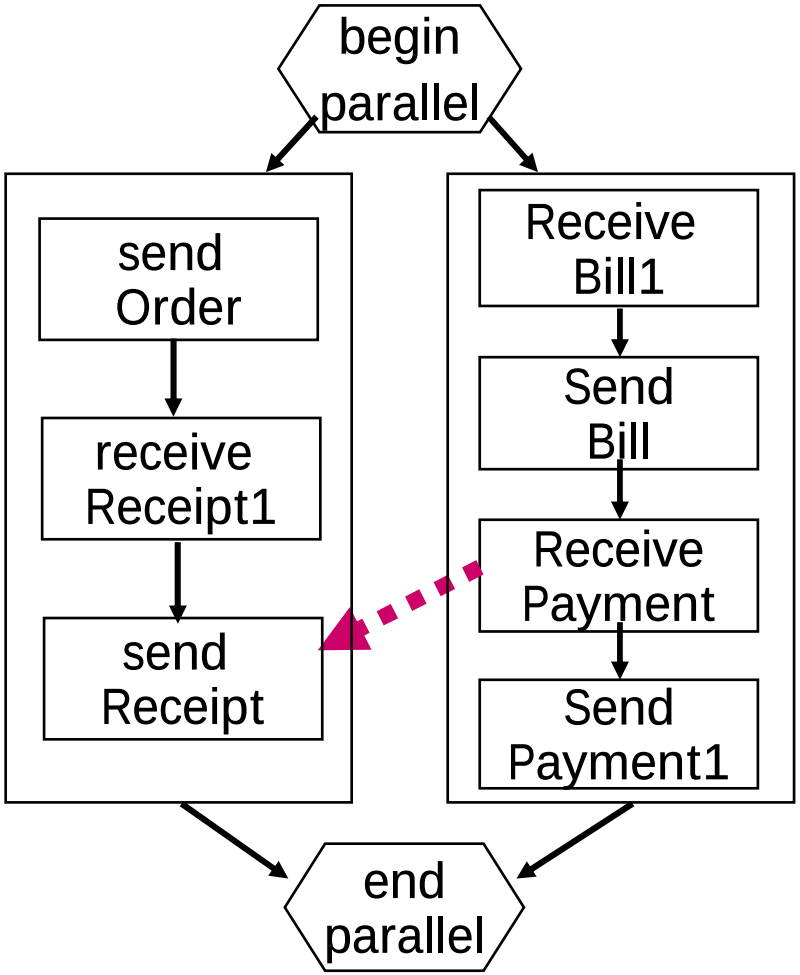
Purchase Order service coordinates other services using ports in WSDL




[from BPEL 1.1 standard]

# BPEL Executable: Procedural Constructs




- Flowcharts "with parallelism"
- "Pick" construct to enable waiting for input (or time out)
- Synchronization within parallel threads
- Comparison of supported constructs: see [van der Aalst '03]

# BPEL Activities

Bell Labs
Innovations for Lucent Technologies

- Invokes an operation on a partner service
  - Send to WSDL port, wait for a response
- Receives invocation from a partner
  - Wait for a message
- Sends a reply message in partner invocation
  - Send a message (corresponding to some earlier message)
- Data assignment between containers
  - Copy local data
- Control structures: sequence, flow (possibly with links), pick, loops, etc.
- Scoping, exceptions, compensation

BACKUP

# Structure of a BPEL Process

Bell Labs
Innovations for Lucent Technologies

```
<process ...>

  <partners> ...
  </partners>
  <containers> ...
  </containers>
  <correlationSets> ...
  </correlationSets>
  <faultHandlers> ...
  </faultHandlers>
  <compensationHandlers> ...
  </compensationHandlers>

  (activities)*

</process>
```

- Web services the process interacts with
- Data used by the process
- Used to support asynchronous interactions
- Alternate execution path to deal with faulty conditions
- Code to execute when "undoing" an action
- What the process actually does

BACKUP

# BPEL Structured Activities

Bell Labs
Innovations for Lucent Technologies

<sequence>

execute activities sequentially

<flow>

execute activities in parallel

<while>

iterate execution of activities until condition is violated

<pick>

several event activities (receive message, timer event) scheduled for execution in parallel; first one is selected and corresponding code executed

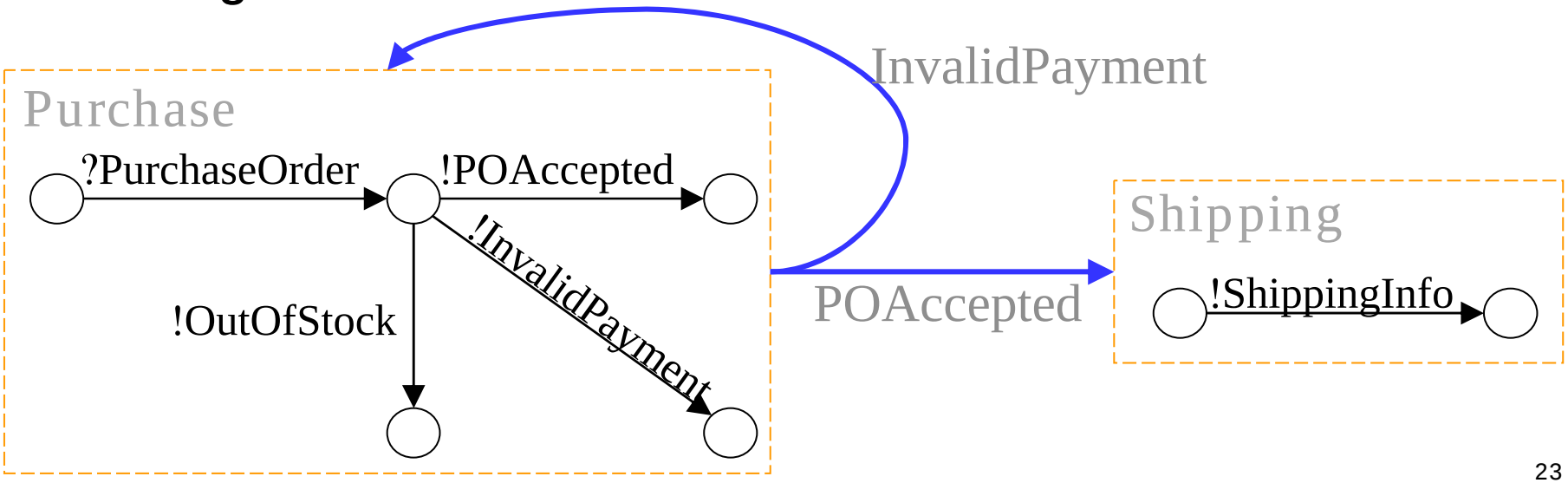
<link ...>

defines a control dependency between a source activity and a target

BACKUP

# Web Service Conversation Language (WSCL)

- A key to web service composition:
  - Interactions between services
- WSCL specifies a conversation (behavior signature) as a labeled graph:
  - Nodes: interactions, individual units of responses
  - Edges: transitions, sequencing of interaction
  - Edge labels: conditions on transitions



# Conversation Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="simpleConversation"
    version="1.01"
    xmlns="http://www.w3.org/2002/02/wscl10"
    initialInteraction="Start"
    finalInteraction="End"
    description="Conversation for a Store Front Service" >

    <ConversationInteractions>
        …list of all interactions
    </ConversationInteractions>

    <ConversationTransitions>
        …list of all transitions
    </ConversationTransitions>
</Conversation>
```

BACKUP

# WS Choreography

Bell Labs
Innovations for Lucent Technologies

- An emerging standard from W3C
  - Drawing inspiration from the $\pi$-calculus
- Global view of composite service interactions
  - Global model: interactions and choreography
  - Choreography Definition Language (WS-CDL)
- Key technical elements
  - Participants and roles: what services are involved
  - Channels: where and how the messages are sent
  - Interactions: message exchange patterns
  - Activities and control structures: e.g., sequencing
  - Choreography: a global description of a composition
    - Interactions, exceptions, finalizer
    - Composable

# WS-CDL Example (Incomplete)

Bell Labs
Innovations for Lucent Technologies

```
<role name="Consumer">
   <behavior name="consumerForRetailer" interface="cns:ConsumerRetailerPT"/>
   <behavior name="consumerForWarehouse" interface="cns:ConsumerWarehousePT"/>
</role> …

<relationship name="ConsumerRetailerRelationship">
   <role type="tns:Consumer" behavior="consumerForRetailer"/>
   <role type="tns:Retailer" behavior="retailerForConsumer"/>
   </relationship> …

<choreography name="ConsumerRetailerChoreo" root="true">
   …
   <interaction channelVariable="tns:retailer-channel "
         operation="handlePurchaseOrder" align="true" initiateChoreography="true">
     <participate relationship="tns:ConsumerRetailerRelationship“
           fromRole="tns:Consumer" toRole="tns:Retailer"/>
     <exchange messageContentType="tns:purchaseOrderType" action="request">
           <use variable="cdl:getVariable(tns:purchaseOrder, tns:Consumer)"/>
           <populate variable="cdl:getVariable(tns:purchaseOrder, tns:Retailer)"/>
     </exchange>
     <exchange messageContentType="purchaseOrderAckType" action="respond">
           <use variable="cdl:getVariable(tns:purchaseOrderAck, tns:Retailer)"/>
           <populate variable="cdl:getVariable(tns:purchaseOrderAck, tns:Consumer)"/>
     </exchange>
     …
   </interaction>
</choreography>
```

BACKUP

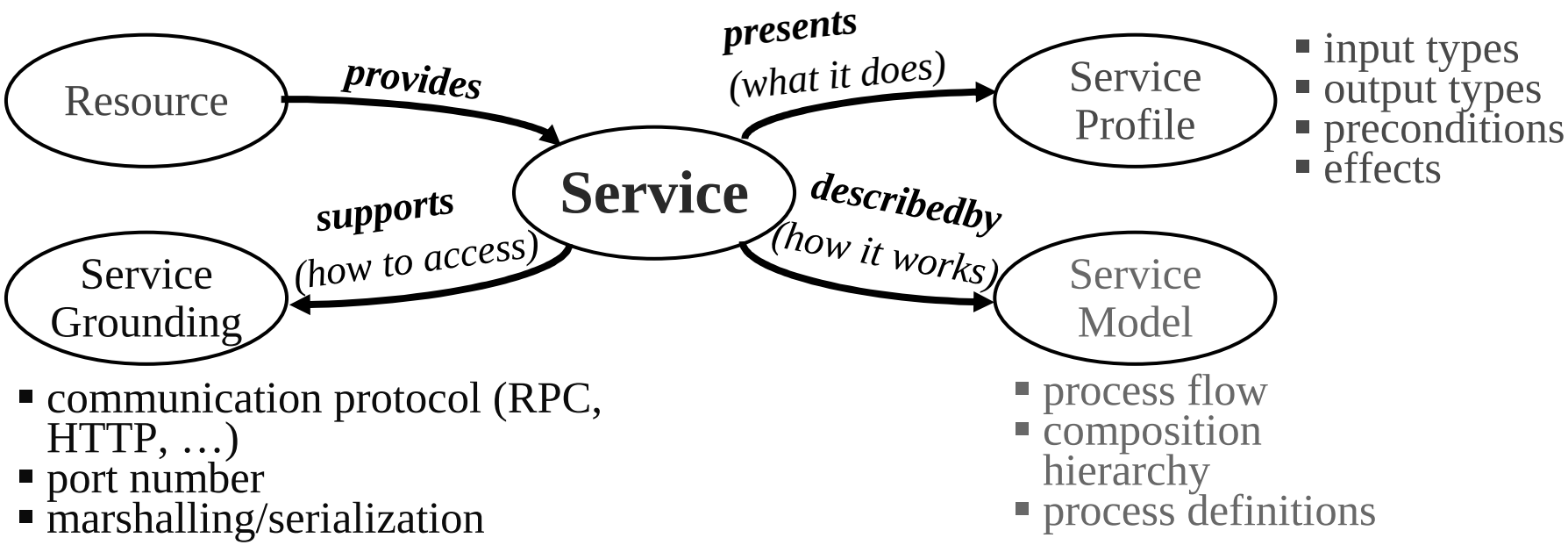
# BPEL meets WS-Choreography

- A scripted composition using
  - WSDL messages
  - Control structures with constrained parallelism
- More procedural
- Executable or abstract
- Favor centralized composition

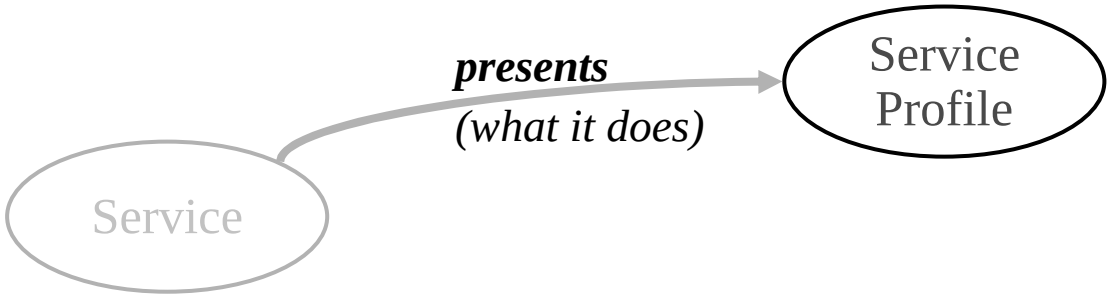
- A global description of what and how WDSL messages are exchanged
- Declarative flavor
- Abstract and not executable (yet)
- Neutral re composition topology
- Channels can be passed around (c.f., $\pi$-calculus)
  - Supports a form of delegation

# OWL-S (Formerly DAML-S)

- An emerging standard to add semantics:
  - An upper ontology for describing properties & capabilities of web services using OWL
- Enable automation: service discovery & selection, invocation, composition & interoperation, execution monitoring



# OWL-S Service Profiles

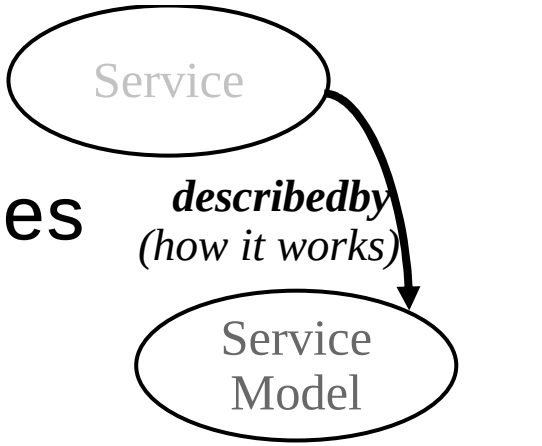



- Input types
- Output types
- Preconditions
- Effects

- Service profile defines what the service provides:
  - Functional descriptions: In/Output, Preconditions, Effects
  - Non functional descriptions: name, category, QoS, …
- Can use situation calculi (or PSL) as formal basis for pre-conditions, effects
  - See below
- Reasoning with pre-conditions and effects

However: OWL-S assumes pre-conditions/effects are in OWL-DL – cannot work with variables in rich ways

# OWL-S Process Model

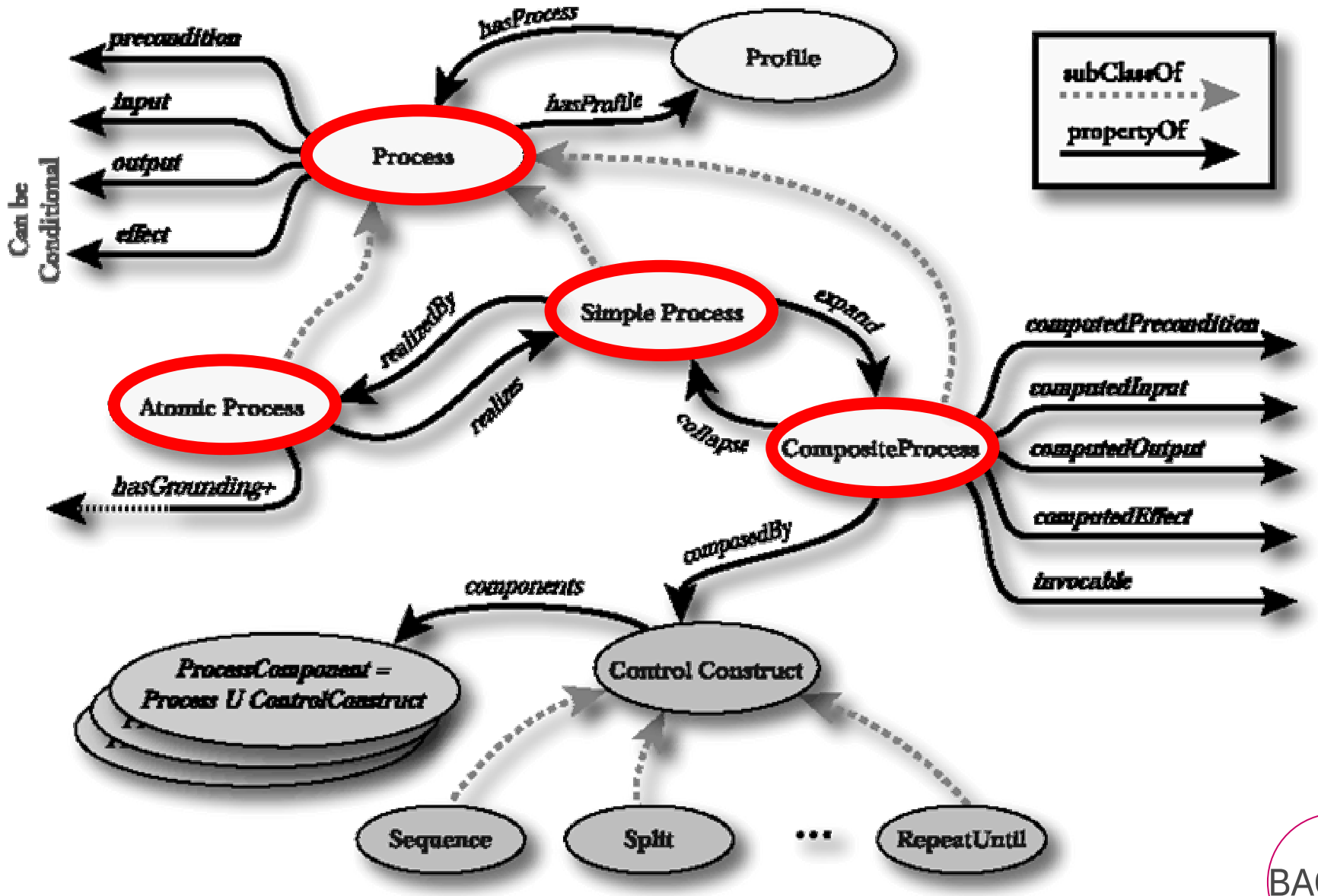



- process flow
- composition hierarchy
- process definitions

- Constructs for composite processes
  - Sequence
  - Concurrency: Split; Split+Join; Unordered
  - Choice
  - If-Then-Else
  - Looping: Repeat-Until; Iterate (non-deterministic)
  - Note: In spirit of Golog, these can be viewed as constraints
- Data Flow
  - No representation of explicit variables, no internal data store
  - Predicate “sameValues” to match input of composite service and input of subordinate service
- Less refined than, e.g., BPEL
- Message behavior of composed OWL-S services not well-understood

# OWL-S Service Model

Bell Labs
Innovations for Lucent Technologies




BACKUP

# Functionality Description

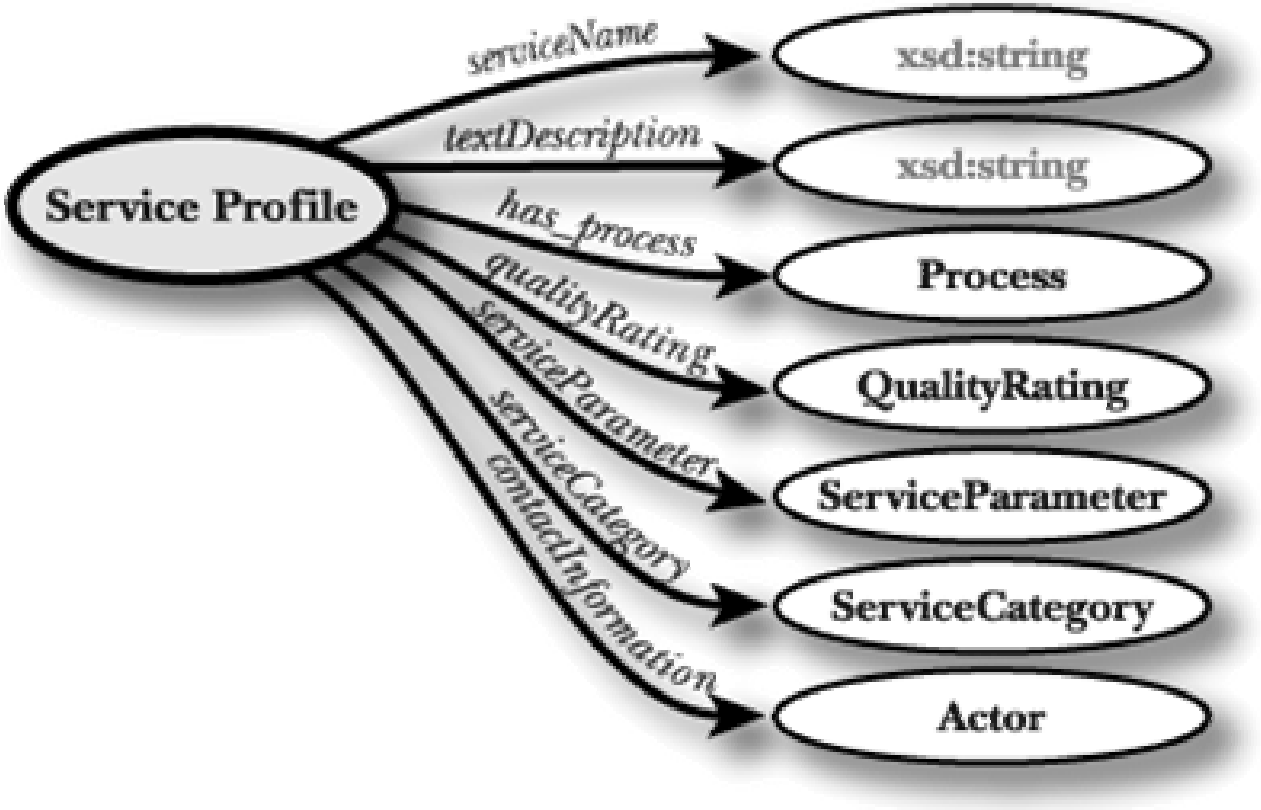
Bell Labs
Innovations for Lucent Technologies

- Preconditions
  - Set of conditions that should hold prior to service invocation
- Inputs
  - Set of necessary inputs that the requester should provide to invoke the service
- Outputs
  - Results that the requester should expect after interaction with the service provider is completed
- Effects
  - Set of statements that should hold true if the service is invoked successfully
  - Often refer to real-world effects, e.g., Package being delivered, or Credit card being debited

BACKUP

# Non Functional Properties

- Provides supporting information about the service




BACKUP

# Profile Hierarchy

Bell Labs
Innovations for Lucent Technologies

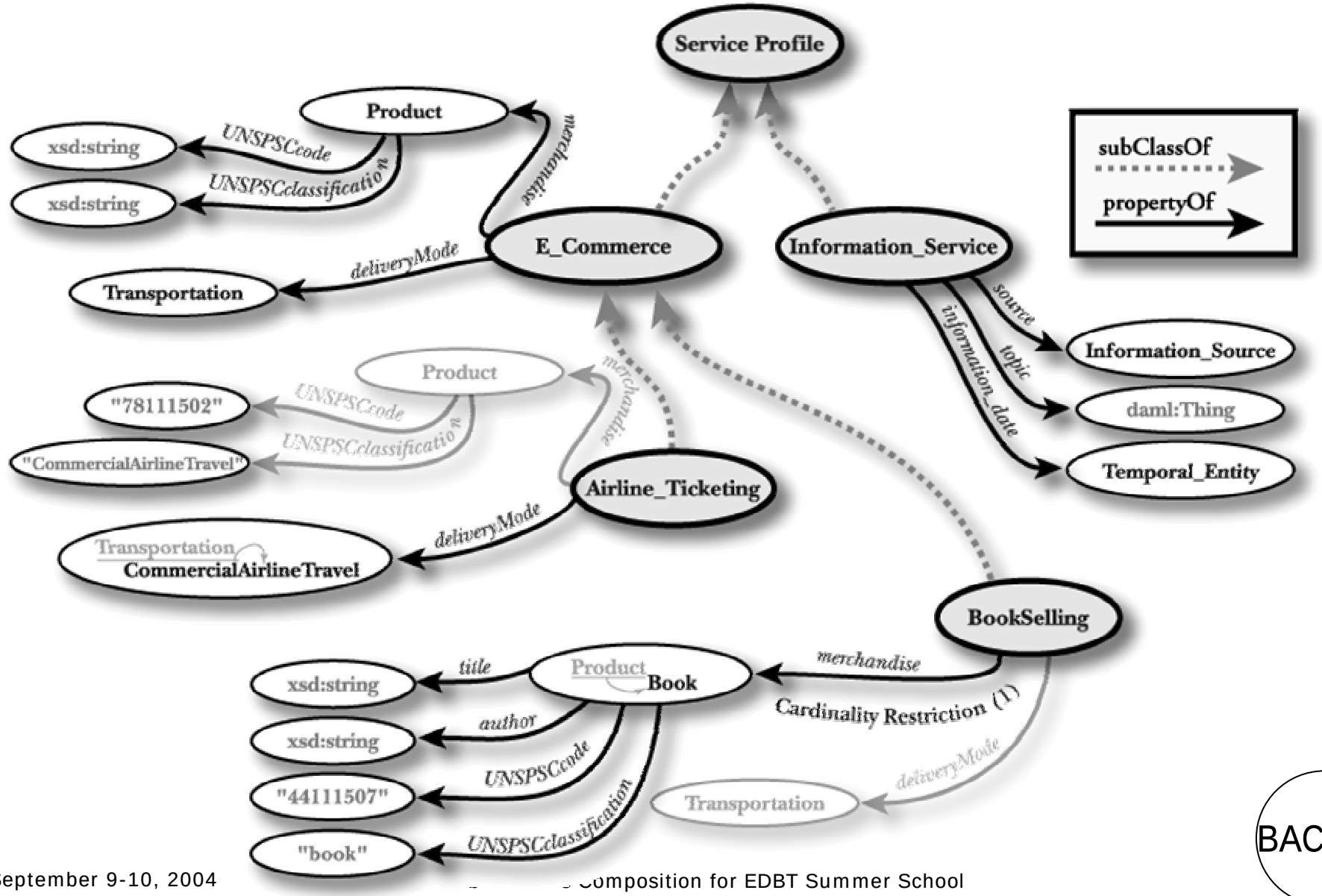
- Sub-classing the Profile model facilitates the creation and specialisation of service categories
- Each subclass can:
  - Introduce new properties
  - Place restrictions on existing properties
- Sub-classing can also be used to specialise requests for service
- An example Profile Hierarchy is provided, but others could just as easily be defined

BACKUP

# Universal Description, Discovery and Integration (UDDI)

- Directory for web services
  - Communicate via SOAP
  - Includes descriptions of services, in terms of:
    - Business, services, binding, “technical fingerprints”
- tModels
  - “Schemas” for describing service templates (PortTypes)
    - There are tModel’s for WSDL descriptions of a service, for ebXML, …
    - When a service registers with UDDI, the technical fingerprint includes listing of tModels that it uses
  - tModel’s can be registered, and incorporated into taxonomies
- Allows queries over services, tModels, implementations, and other information
- UDDI expected to expand over time, enabling richer service descriptions

# OWL-S Profile Ontology is Analogous to the Concept of UDDI Taxonomy




BACKUP

# Outline

Bell Labs
Innovations for Lucent Technologies

- Introduction
- Standards
- Perspectives on “Composition”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# Models of Interoperation

Bell Labs
Innovations for Lucent Technologies

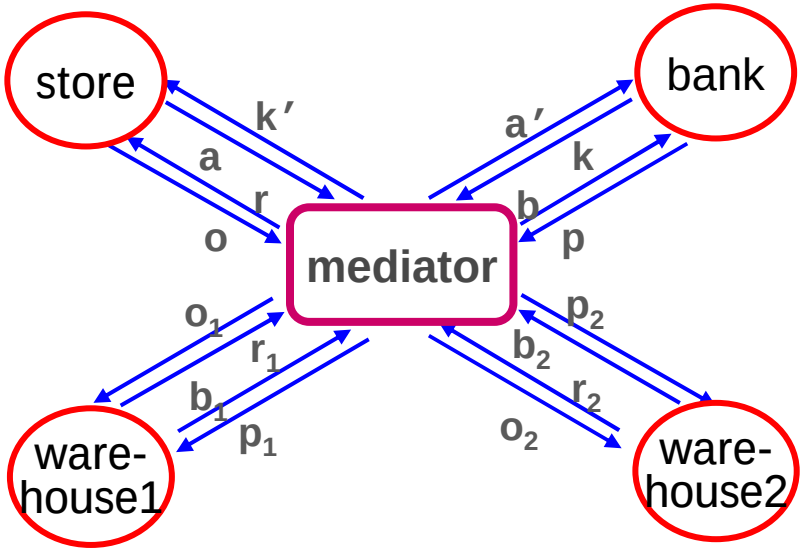
Different models focus on different aspects

- Automata-based
  - Intricate structure for atomic services
  - Rich interleaving between atomic components
  - Activity-based or message-based
- Logic-based perspectives
  - Frameworks supporting proof and model theories
  - Possible to model “effect on the world”
  - Different ways of modeling complex services
- Constraint-based
  - Support partial specification of desired behaviors
  - Non-determinism OK when building composition

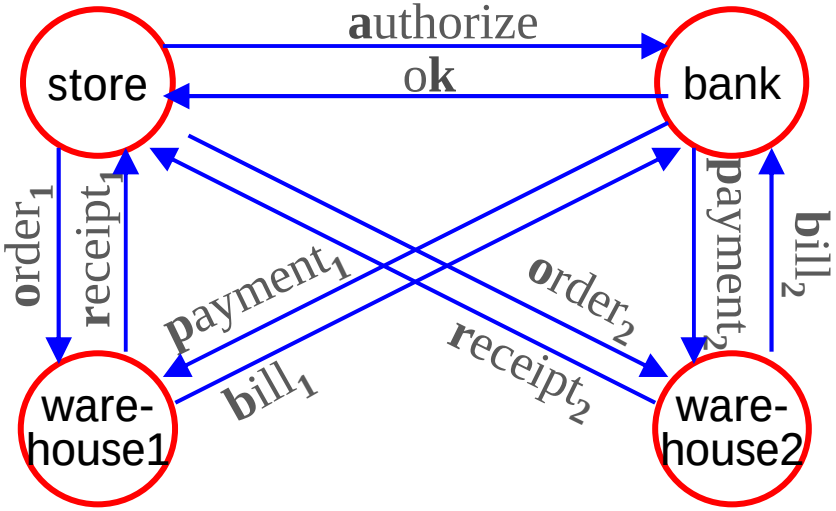
# First Impressions: Topology

Two common approaches:

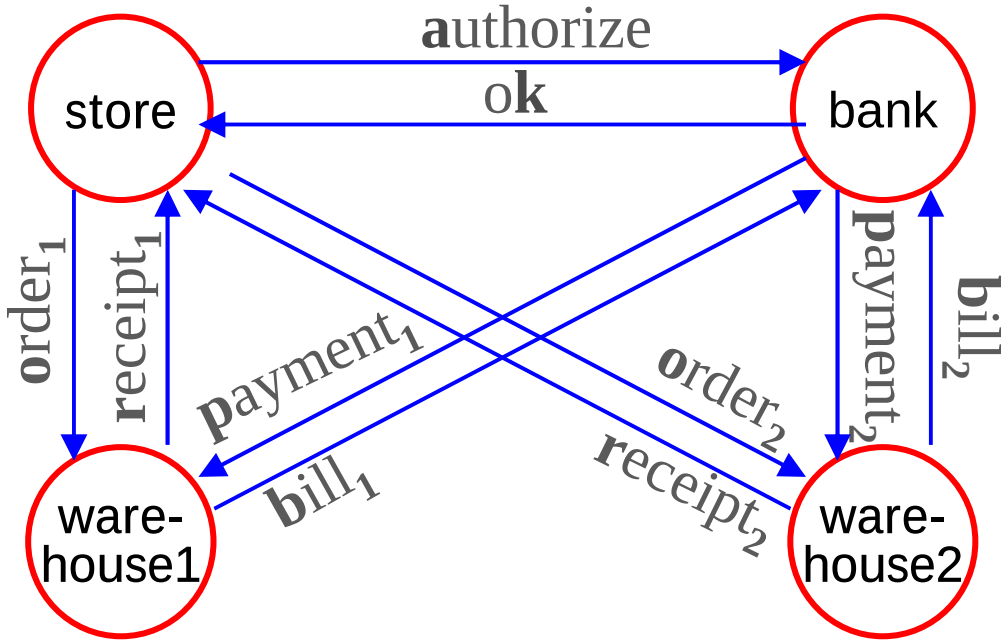
- Mediated, or "hub and spoke" (typical of BPEL)




- Peer-to-peer



# First Impressions: Enactments



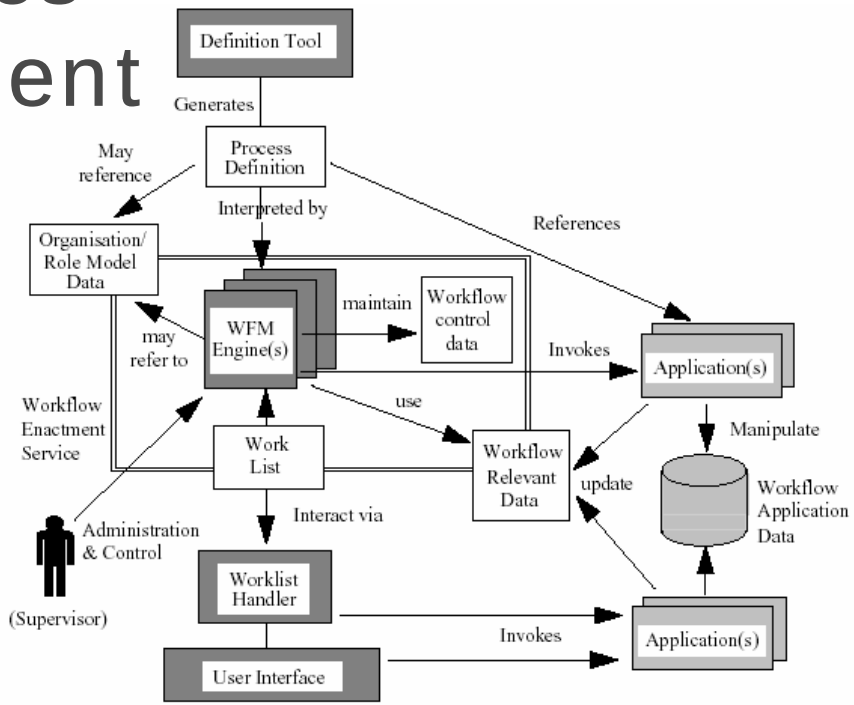

- "Enactment" = the execution of multiple steps in a (composite) service, corresponding to a single instance of a (possibly complex) business process
- Nested enactments: one **authorize**, several **order**s
- Typical to focus modeling and reasoning on individual enactments

# Compositions vs. Complex Individual Services

Bell Labs
Innovations for Lucent Technologies

- ***Re-usability of component parts***
  - For individual services, this is a design goal, but not enforced
  - For compositions, this is foundational assumption
- ***World view***
  - Components of individual service can “see” the rest of the service, modulo scoping, etc.
  - Services in a composition have a limited interface to “see” other services (typically via messages only)
    - Implications on transactional aspects
- ***Management of different enactments***
  - Individual service: Details of enactment management are hidden
  - Composite service: Need mechanism for associating activities of component services with appropriate global enactment
    - BPEL uses the phrase “correlation sets”

# Service "Glue" Languages and Workflow Management




- "Computerised facilitation or automation of a business process, in whole or part" [WfMC]
- Centralized control
  - State of conversation maintained by WF manager
- Delegation of "almost everything" to the app.s
  - E.g., application data is not accessible to WF manager
- Workflow standardization has mixed success
  - Web services must interoperate ⇒ standards likely
  - Should focus on interfaces, not internals

Web services world is much more "open-ended"

# Selected results on Automated Composition

Bell Labs
Innovations for Lucent Technologies

Three families of results, each based on a different model

- Mealy model: Emphasis on message passing
  - Primarily useful for characterizations
- Roman model: Elegant approach in restricted framework
  - Using automata with emphasis on activities
  - Automata-based and logic-based proofs
- "Semantic": Simple compositions driven by achieving goals "in the world"
  - Two approaches for combining atomic OWL-S services in a systematic fashion

# Selected Models

Bell Labs
Innovations for Lucent Technologies

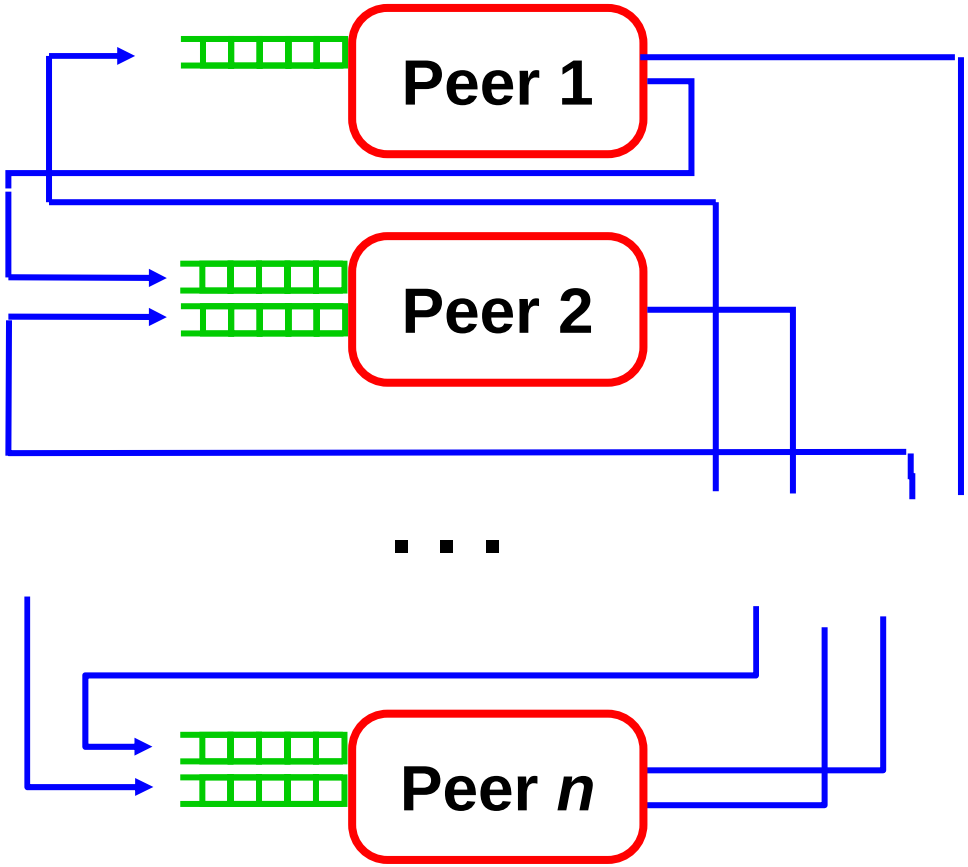
| Model | Emphasize Individual service | Emphasize Composite service | Style |
|---|---|---|---|
| Mealy/Conversation | | yes | Automata, message-based |
| Roman | | yes | Automata, activity-based |
| Situation Calculus/PSL | yes | yes | First-order logic, activity-based |
| Data-Driven | yes | | Automata (specified by rules) |
| CTR/CTR-S | yes | | Stylized logic |
| Commitment | | yes | Constraints, message-based |
| AZTEC, (Minoan) | | yes | Emphasizes asynchronous events, telecom apps |

BACKUP

# Outline

- Introduction
- Standards
- Perspectives on “Composition”
  - Mealy machine/conversations
  - Roman
  - “Semantic”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# A composition framework based on messaging

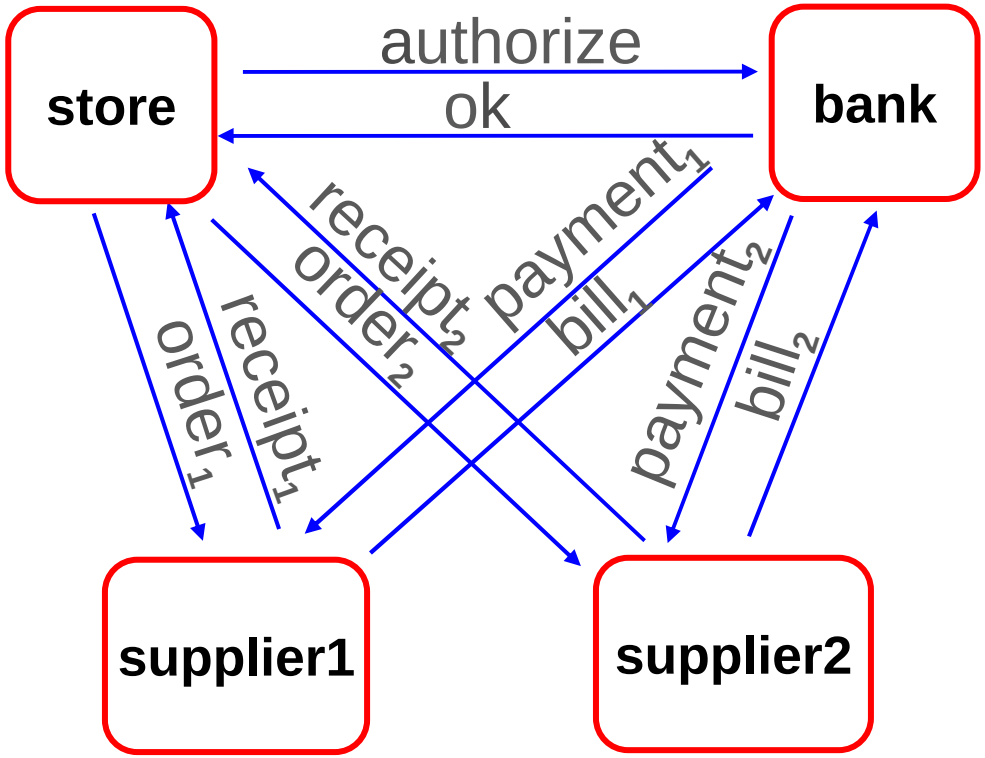



- A peer: autonomous process executing an e-service
- Assume reliable communication

# E-Composition Schema

- An ec-schema is a triple $(P, C, M)$
  Specifies the infrastructure of composition

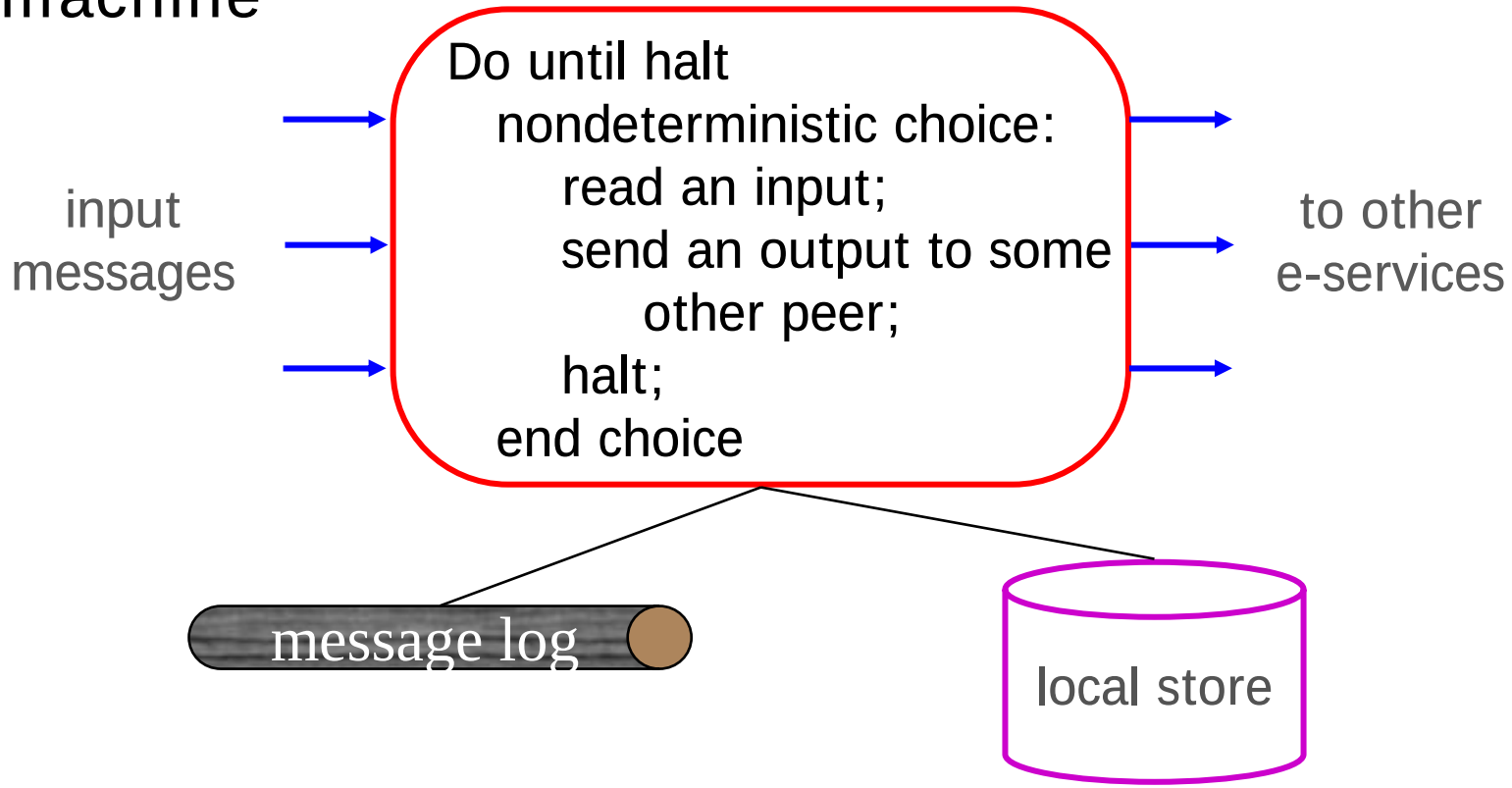
- $P$ : finite set of peers (e-services)
- $C$ : finite set of peer-to-peer channels
- $M$ : (finite) set of message classes




- Fixing an ec-schema is strong restriction
  - It would be useful to study a relaxation of this

# What about inside the web service?

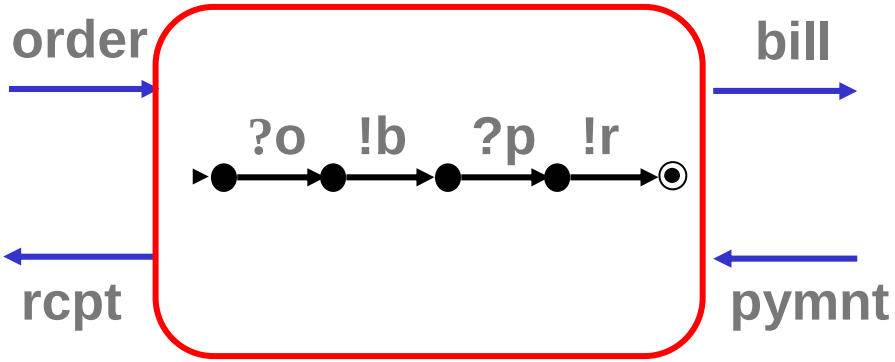
- In the most general case, a peer can be a Turing machine



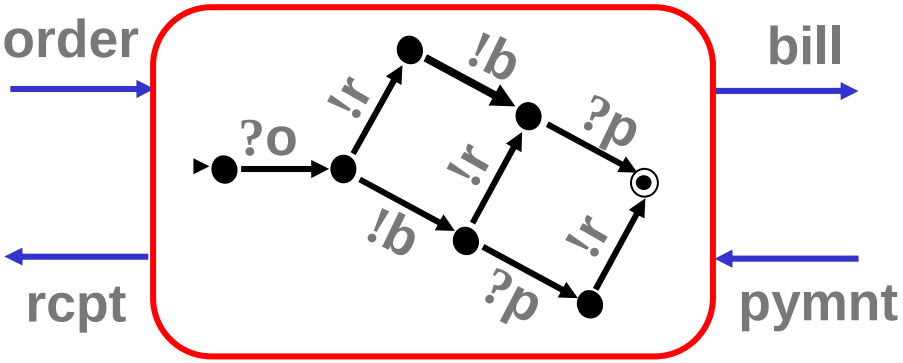

- For analysis and optimization, useful to study more focused languages

# Behavioral signatures via Mealy peers

- Finite State Automata with input/output
  - Follows spirit of process algebras, communicating processes



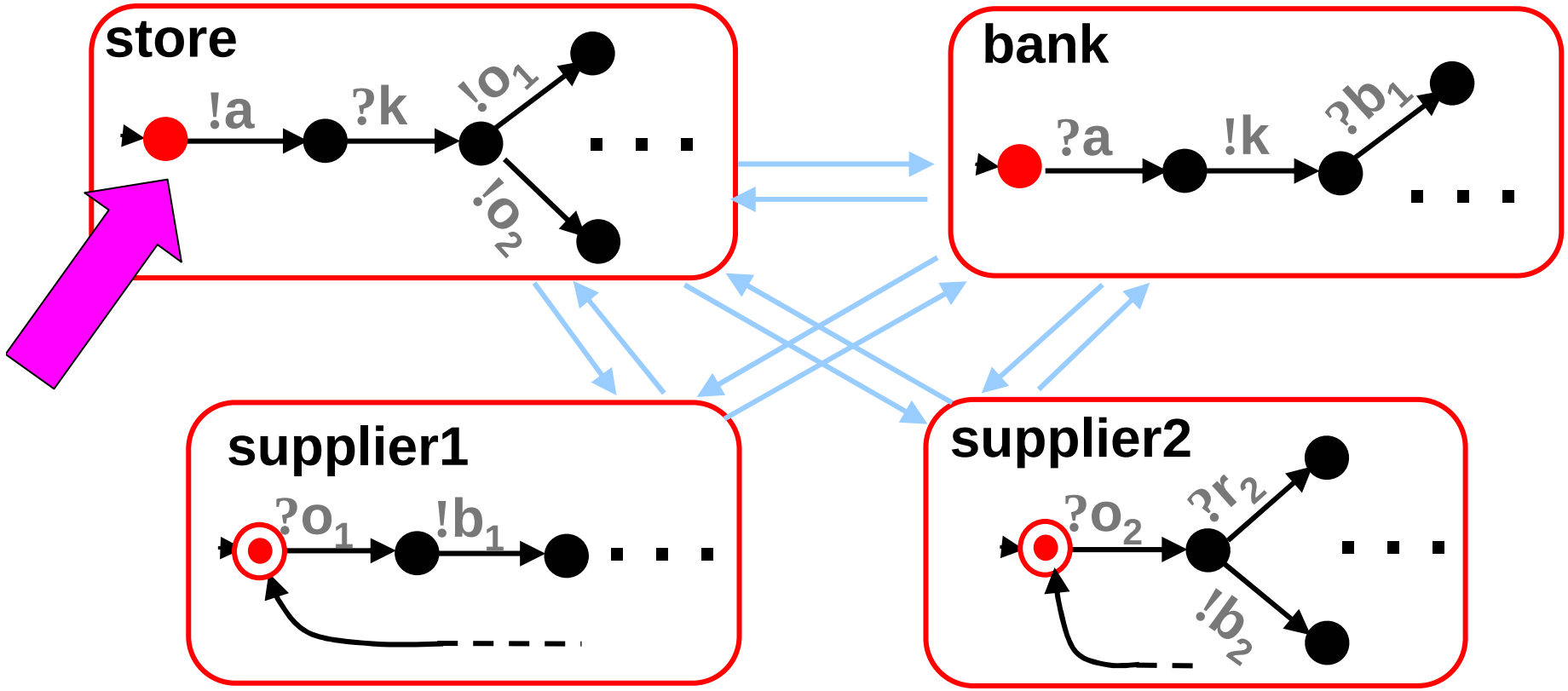

(a) "cautious" supplier




(b) "trusting" supplier

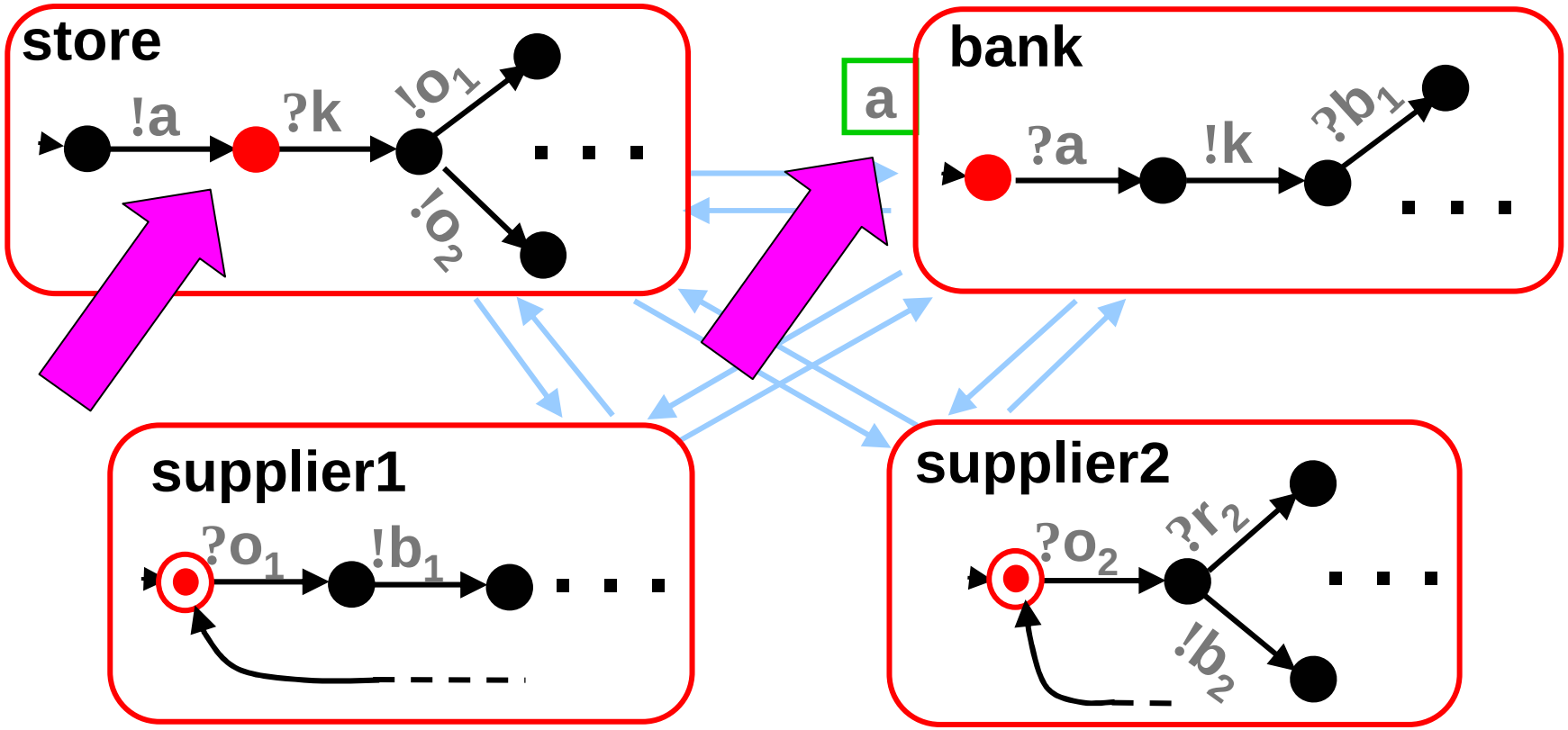
- Can model single or sequenced enactments
- Advantages for analysis, e.g., verifying temporal properties, characterizing global behavior

# Combining Peer and Composition Models



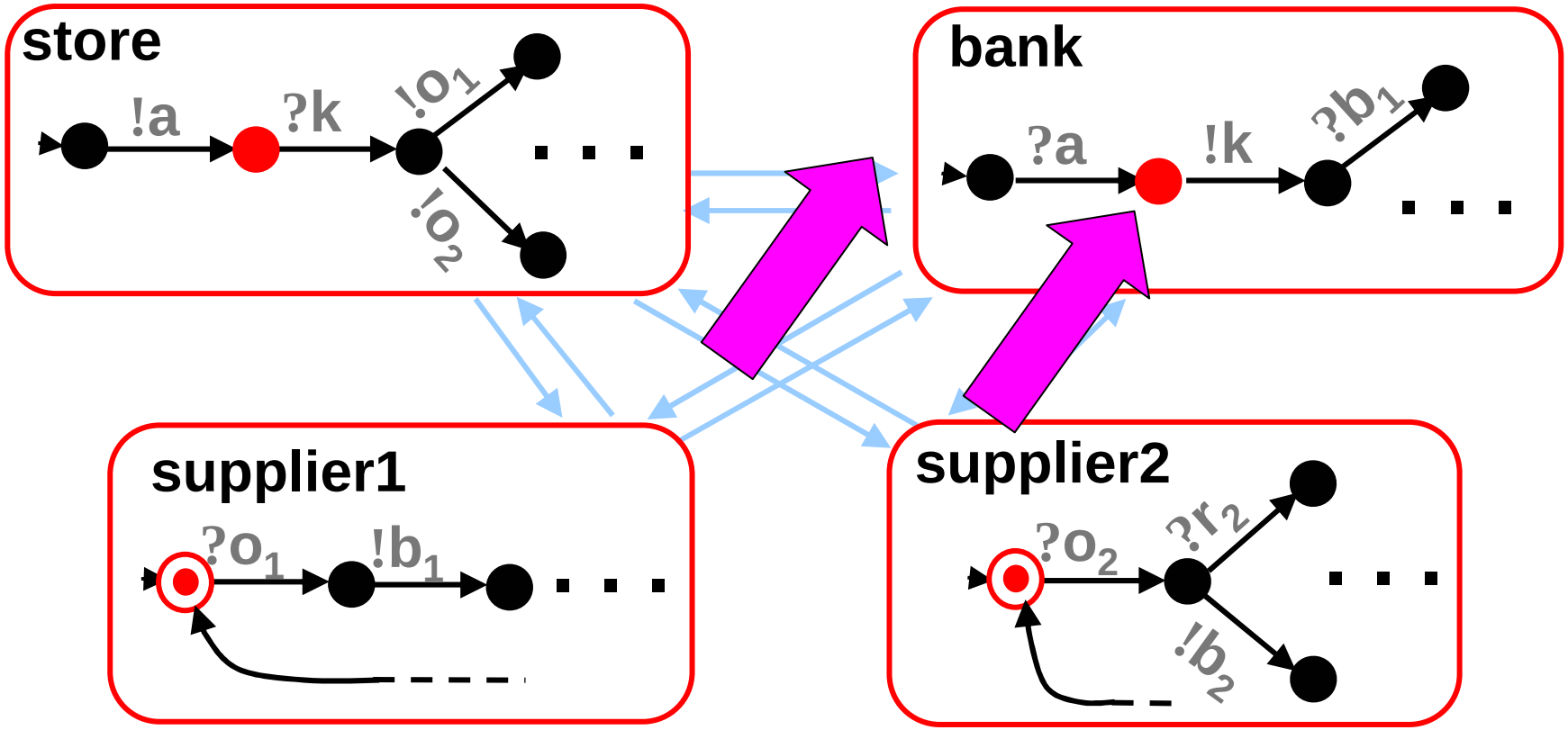

- Peer fsa's begin in their start states

# Executing a Mealy Composition (cont.)



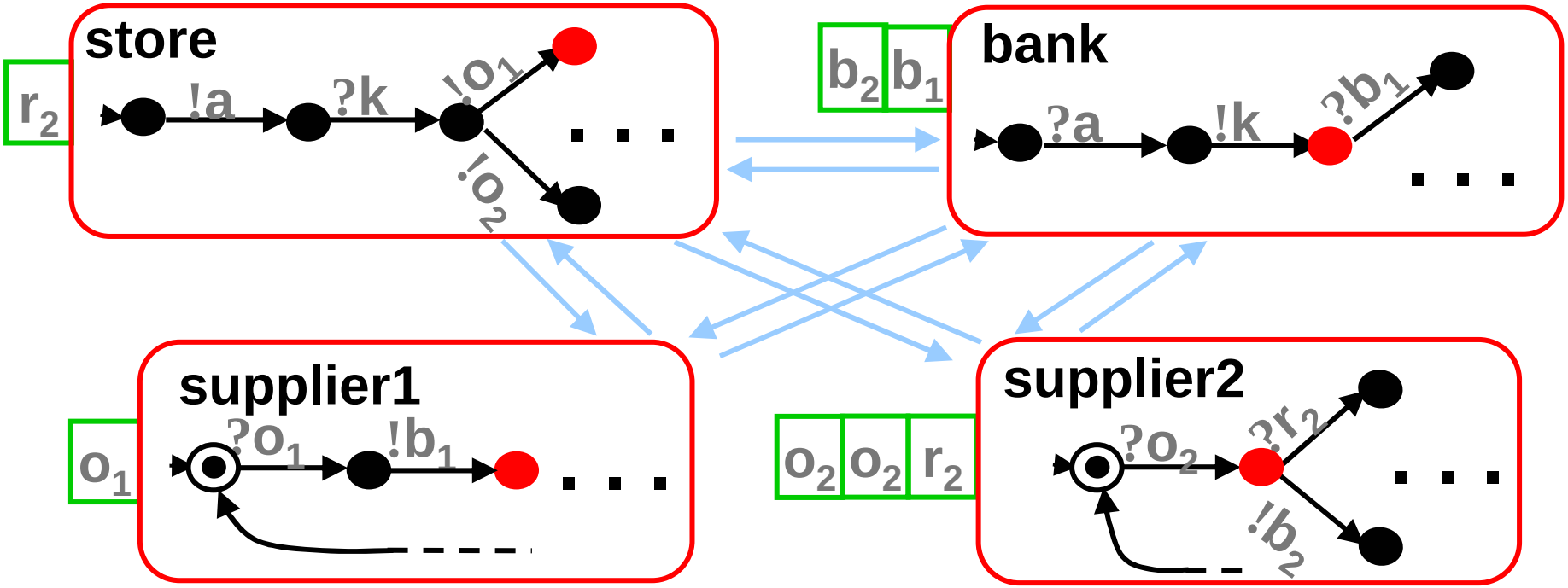

- STORE produces letter **a** and sends to BANK

# Executing a Mealy Composition (cont.)




- BANK consumes letter a
- Execution successful if all queues are empty and fsa's in final state

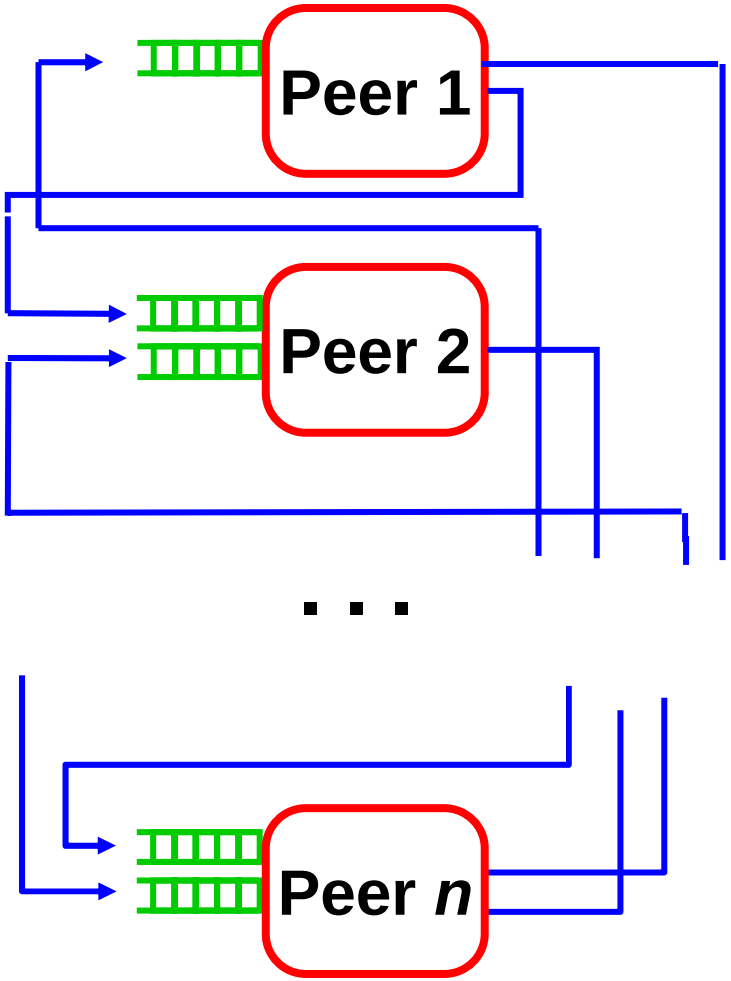
# “State” and “Conversation”




- The ***state of the composition*** is based on
  - state of each peer
  - contents of the queues
- ***Conversation*** : one enactment of global process
  - Can have “sub-conversations” of a conversation
  - Little known about formal properties of conversations

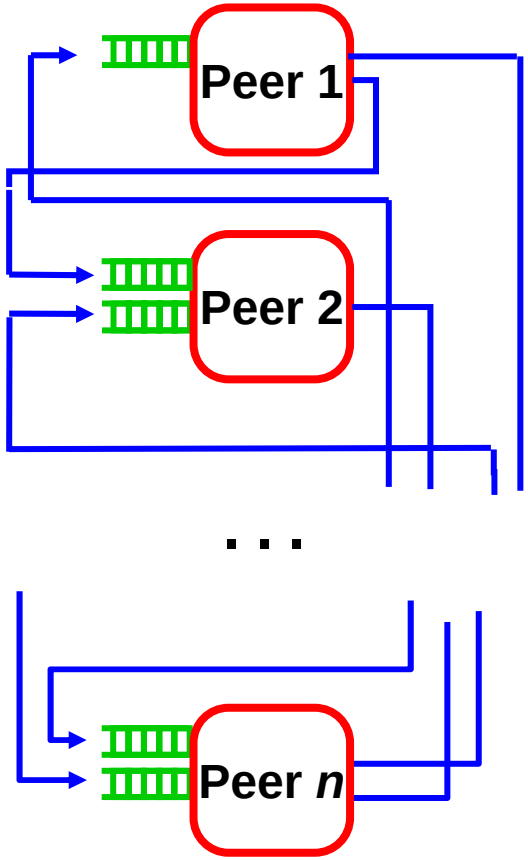
# Important Choices for Composition Model

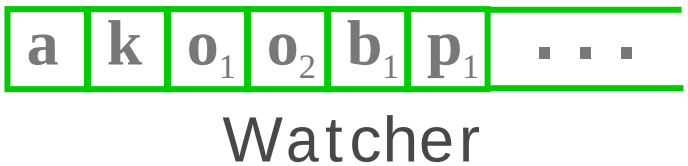
- Representation formalism for peer implementations
- Expressive power of peer implementations
- Bounded vs unbounded queues
- Several queues vs one queue vs heap vs …
- Open vs closed
- Restricted topologies/control: peer-to-peer, hub-and-spoke, hierarchical, …



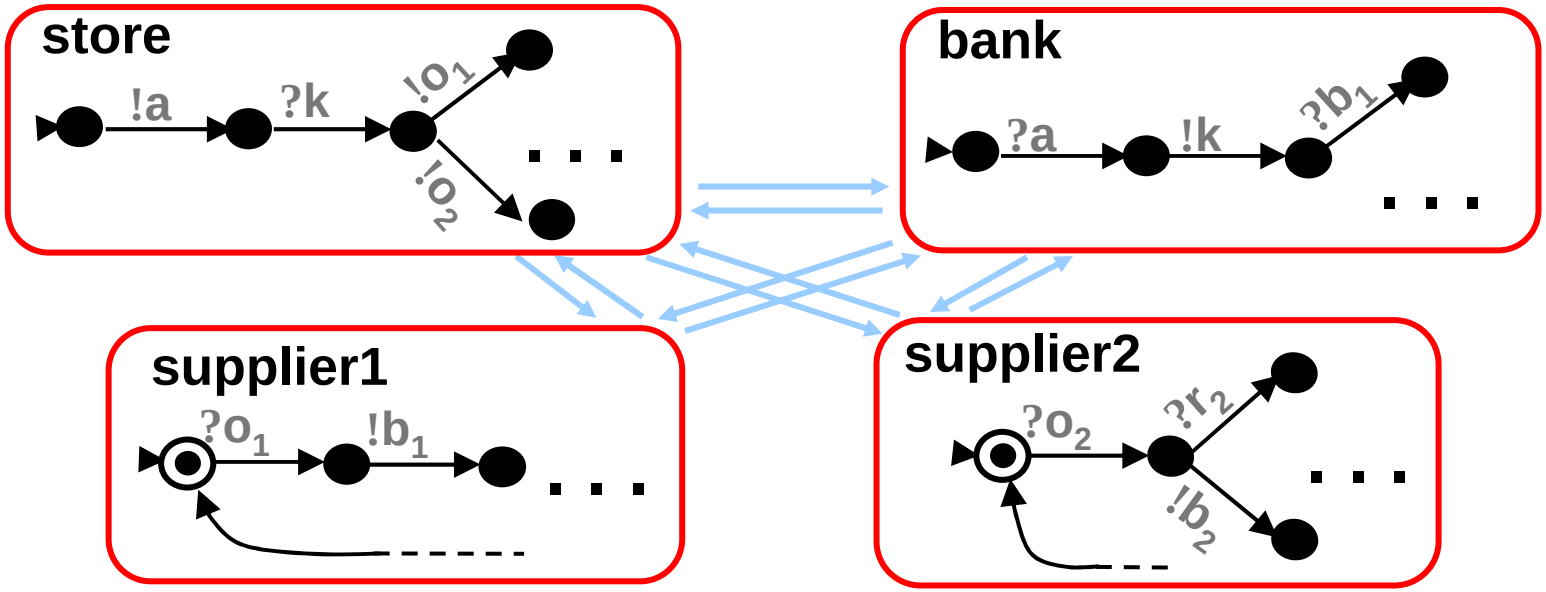
# Qualitative Analysis of Compositions for Message-based Models

- "Conversation Languages" [Bultan et.al. WWW'03]
- Assume a "watcher" that observes all messages sent
  - So, the "observables" here are simply the messages sent
- Language of peer implementation is set of words formed by successful executions of the implementation



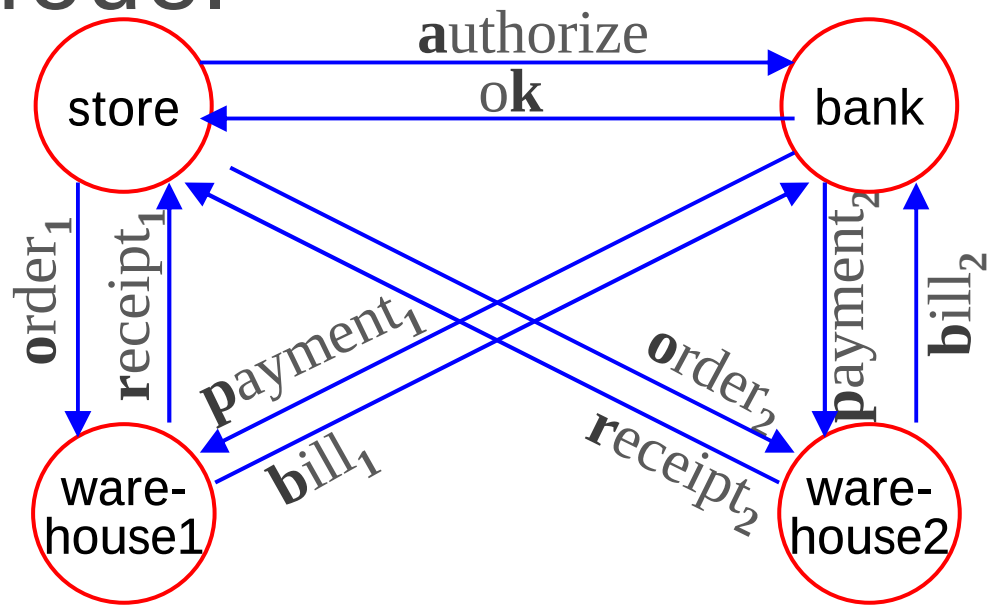



# Example Conversation Language




- Language: ak SH( ($o_1$ $r_1$ $b_1$ $p_1$)*, ($o_2$ SH($r_2$,$b_2$$p_2$))* )
  - First ak, then a shuffle of orders against Supplier1 and orders against Supplier2
  - Supplier1 is “cautious” and Supplier2 is “trusting”
- This language is regular
- Same language for bounded or unbounded queues

# Conversation Realizability in Mealy model



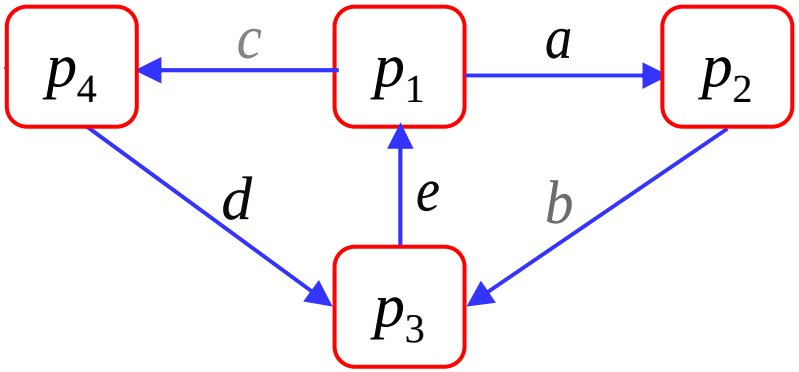

- In the example, the system realizes a regular language:

$$\mathbf{a}\ \mathbf{k}\ \mathit{shuffle}\,(\mathbf{o_1}\ \mathbf{r_1},\ \mathbf{b_1}\mathbf{p_1})^*,\ (\mathbf{o_2}(\mathit{shuffle}\,(\mathbf{r_2},\ \mathbf{b_2}\mathbf{p_2}))^*)$$

- Design question:

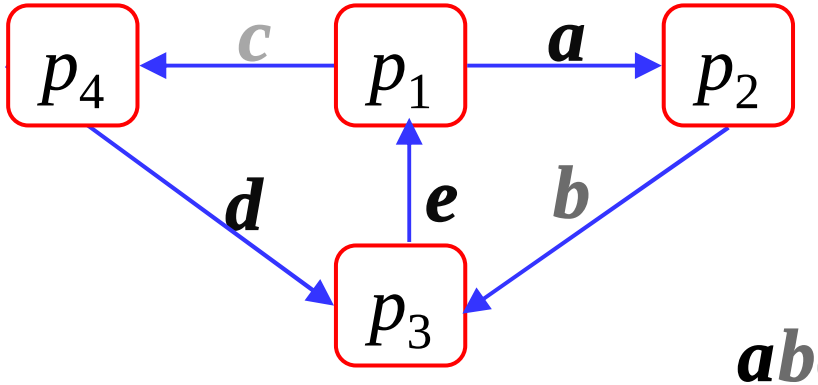
Given a (regular) language $L$, can we design Mealy services so that their conversation language is $L$ ?

# A Quick Answer: Some Regular Languages are not Realizable




- Very simple language $\{ abcde \}$
  - Every Mealy composition allowing conversation $abcde$ will also allow $acbde$
- Three reasons why a language may not be realizable:
  - Impact of local views — Projection-Join
  - Impact of send delays — Preponing
  - Impact of unbounded queues — can yield context-sensitive languages

# Local View and Join




Local views

| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | |
|---|---|---|---|---|---|
| ***abcde*** | ***ace*** | ***ab*** | ***bde*** | ***cd*** | ***acbde*** |

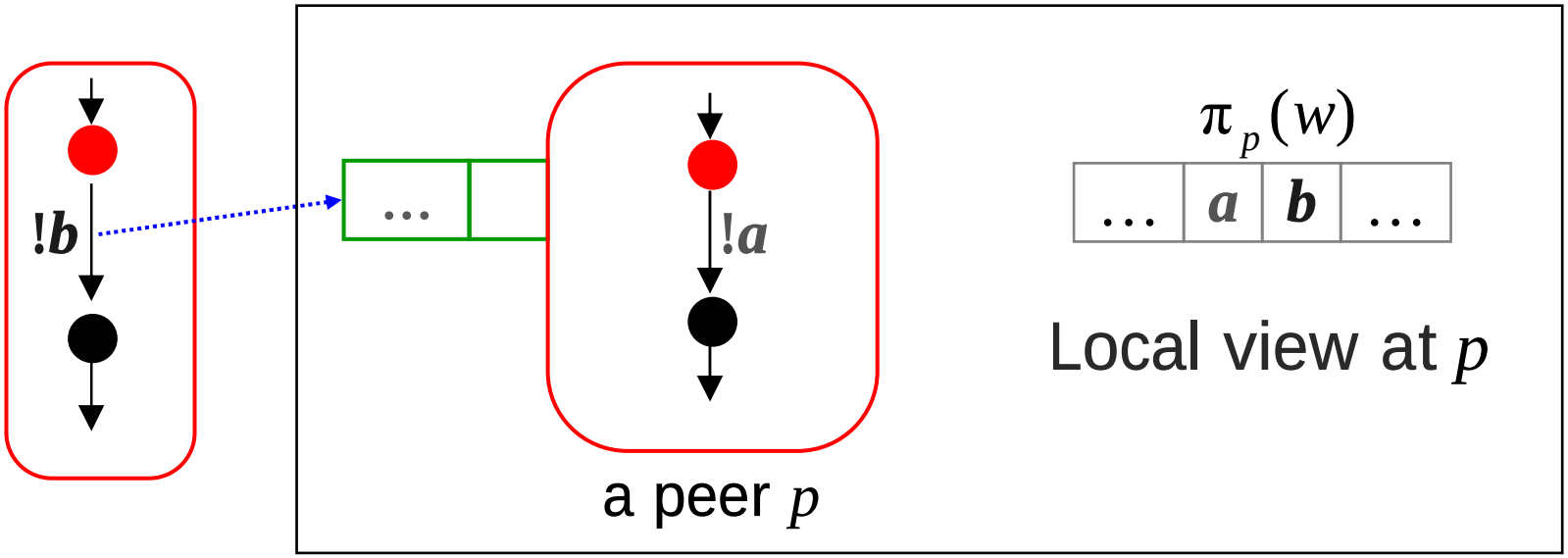
- Local view of a peer $\pi_{\text{peer}}(L)$: the part of conversation the peer participates (receives or sends)
- Given languages $L_i$ over $\Sigma_i$, $1 \le i \le n$

$$\bowtie_i L_i = \left\{ w \,\middle|\, \forall 1 \le i \le n, \pi_{\Sigma_i}(w) \in L_i \right\}$$

- Mealy conversation languages $L$ are closed under "projection-join": $\bowtie_{\text{peers}} \pi_{\text{peer}}(L) \subseteq L$

# Delayed Send and Prepone

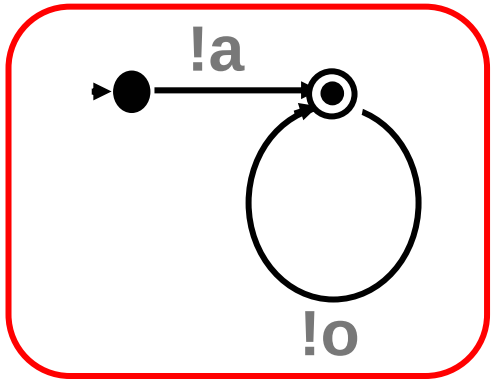
- If the global watcher sees $w =$ | … | $a$ | $b$ | … |



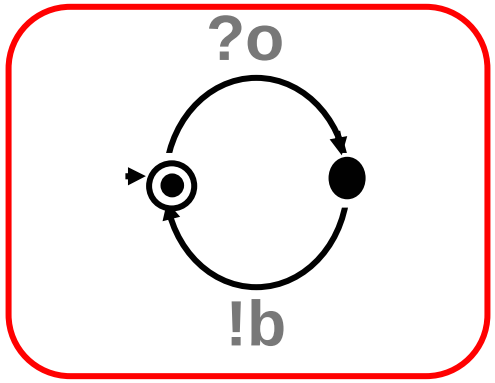

- $\pi_p(w)$ should also allow $w' =$ | … | $b$ | $a$ | … |

- $w'$ is an element of the prepone of $w$
- Mealy conversation languages are closed under prepone

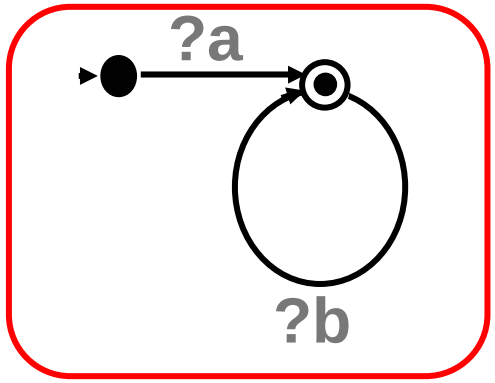
# Unbounded Queues ⇒ Unexpected Behaviors




(a) Store'




(b) Supplier'




(c) Bank'

- Abstract versions of previous e-services
  - But, no "handshakes" for messages
- Conversation language L:
  - $L \cap ao^*b^* = \{ ao^nb^n \mid n \geq 0 \}$, i.e., L is not regular

# How bad is it ?

Bell Labs
Innovations for Lucent Technologies

- In general, conversation language with Mealy peers and unbounded queues is context-sensitive
  - Accepted by a quasi-realtime automaton with 3 queues

- Take away: "Bottom up" design of compositions may lead to undesirable global behaviors

- For hierarchical ec-schemas:

Each peer is a Mealy implementation $\Rightarrow$ Conversation language is join-prepone closure of a regular language

# Realizing the Closure of a Regular Language

- Given a regular language $L$, we can always find a Mealy composition that realizes the following closure:

$$\bowtie_{\text{peers}} \text{LocalPrepone}^{*}(\pi_{\text{peer}}(L))$$

  – Based on a projection construction

- Provides an approach to “top-down” design

- Reminiscent of “synthesis” results from automata

# A Sufficient Condition for "true" Realizability [Fu et al CIAA '03]

- Let $L$ be a regular language, accepted by Mealy machine $A$
  - Let $A_1, ..., A_n$ be the "projections" of $A$ to peers $1, ..., n$
- Three key conditions:
  1. Lossless join: $\mathrm{JOIN}(\pi_1(L), ..., \pi_n(L)) = L$
  2. Queues are optional: can construct a product machine from the $A_1, ..., A_n$ in which every message sent is ready to be read immediately
  3. Each $A_i$ is autonomous: in each state, $A_i$ can do only sends, or only receive, or terminate
- The conjunction of these conditions implies that $L$ is realizable

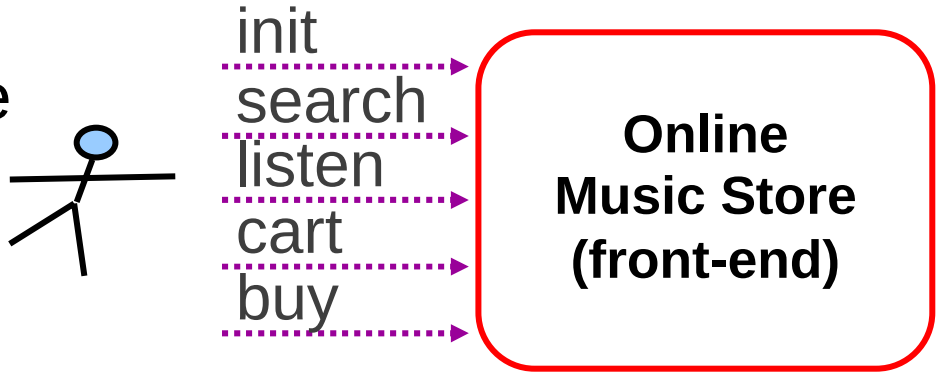
# Outline

- Introduction
- Standards
- Perspectives on “Composition”
  - Mealy machine/conversations
  - Roman
  - “Semantic”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# “Roman” model: A Focus on human-machine

[Berardi et. al. ICSOC 03]

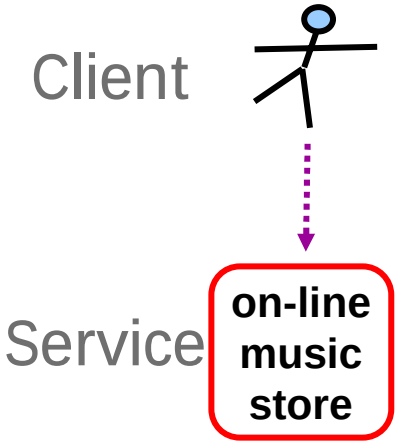
- Model of human-machine web services (e.g., Amazon)
- Focus on activities



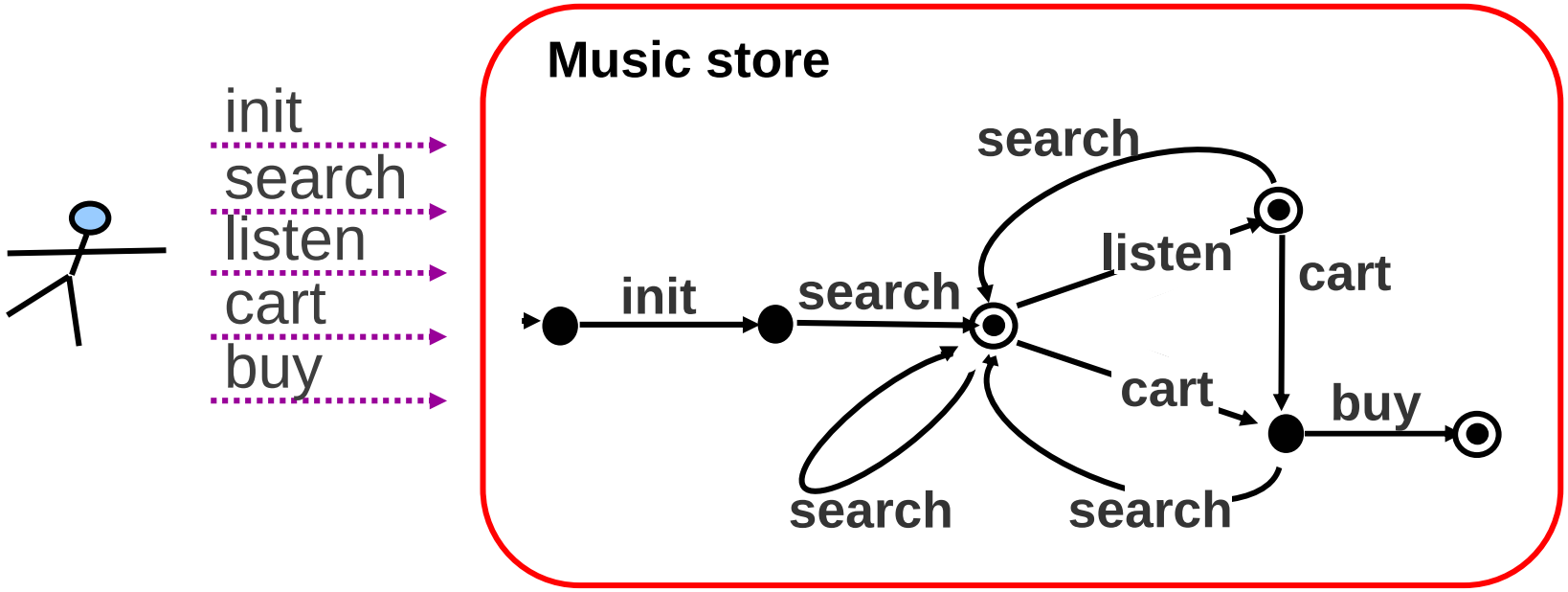

- Abstract behavior of the Service: Client selects next activity

Do until Client selects “End”

1. Give Client a choice of actions to be performed
2. Wait for Client choice
3. Perform action chosen by Client

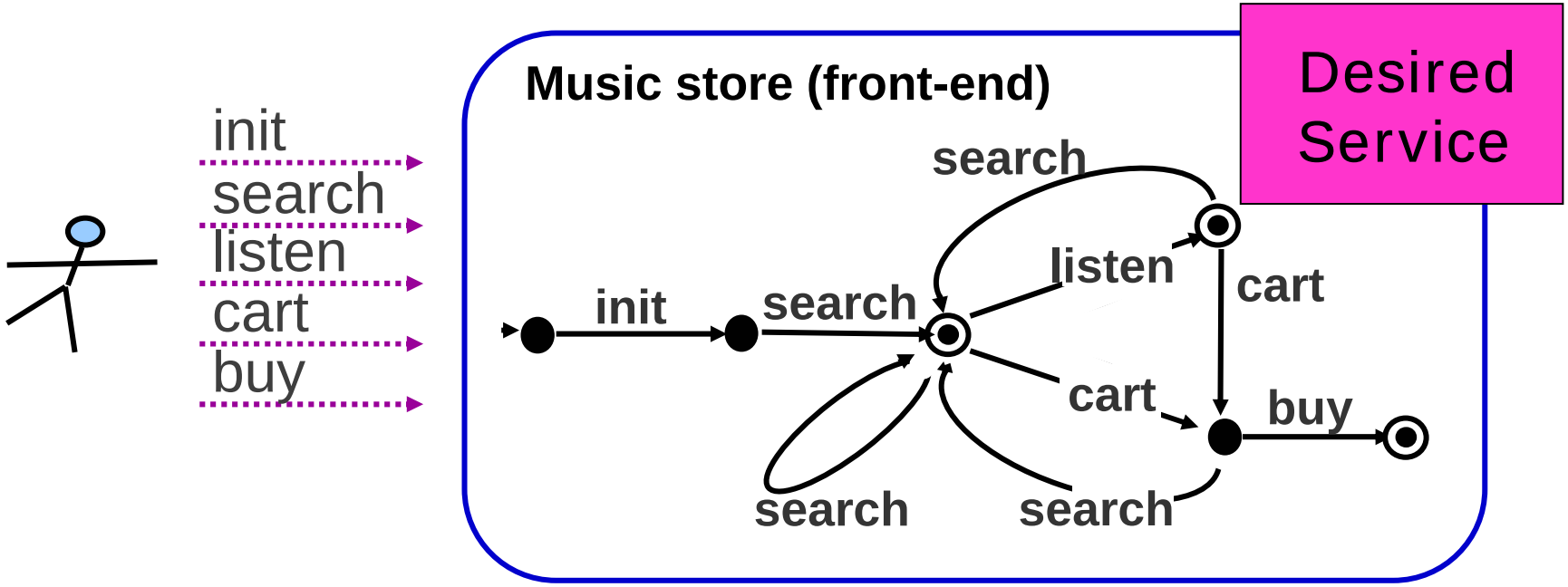


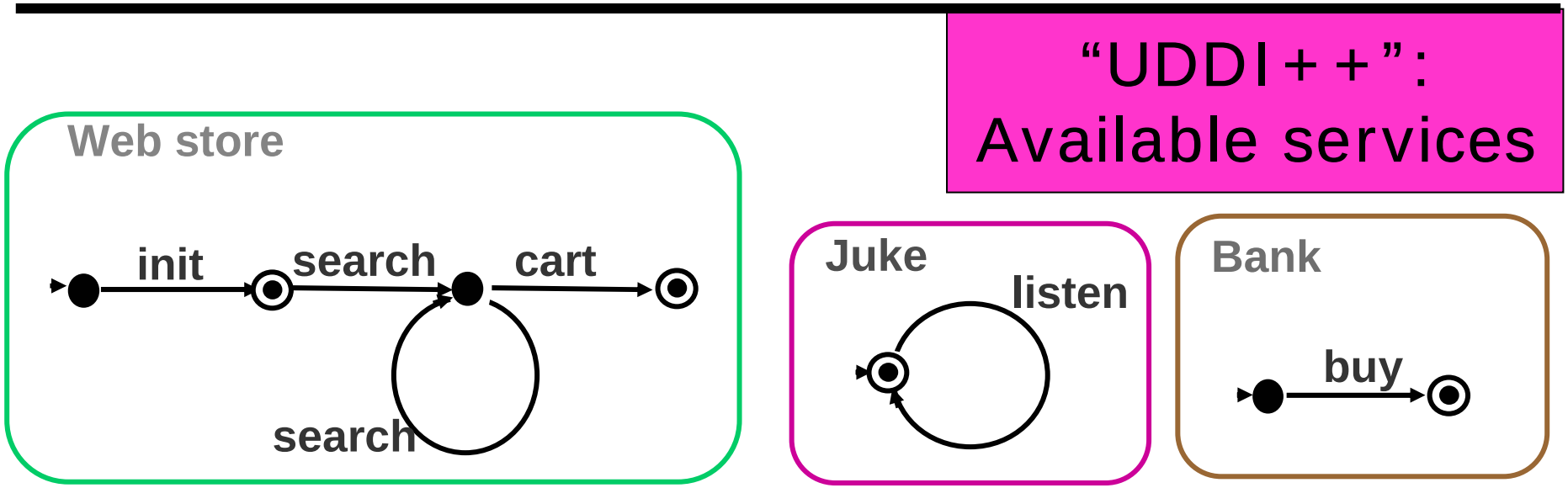
# Roman Model: Automata representation




- Transitions labeled by activities
- More abstract than message-based approach
- For a given state, the out-edges represent the set of options that will be presented to the user

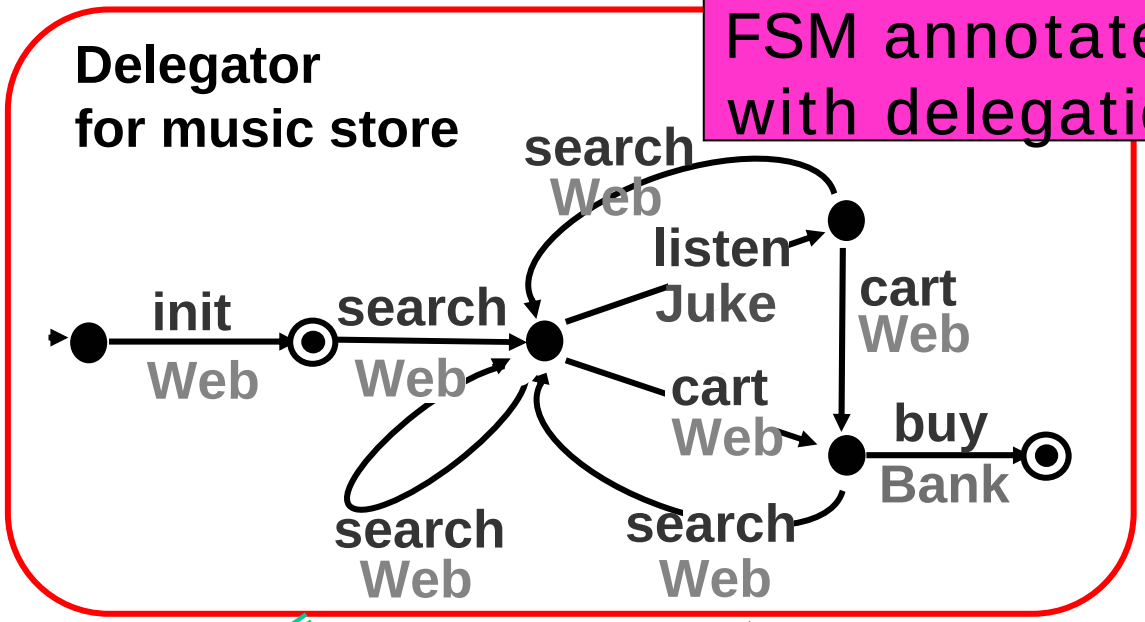
# Roman Model: Composition Problem



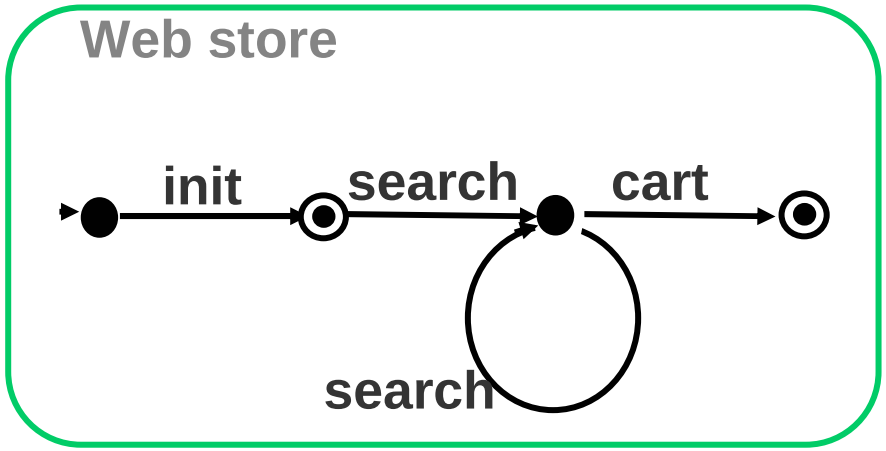



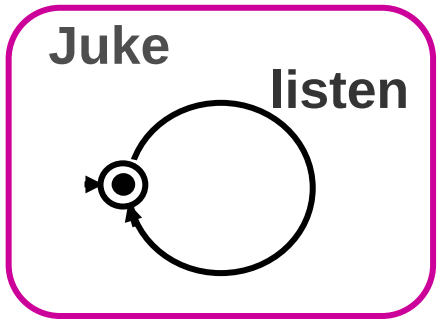
# A Roman Composition

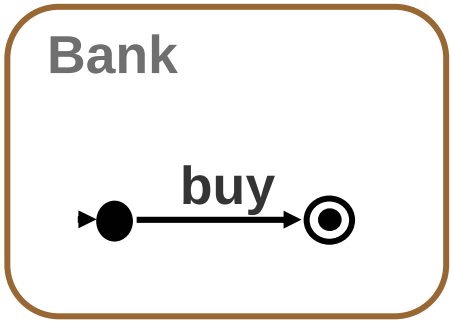
Delegator: Activity-based FSM annotated with delegations


"UDDI++": Available services

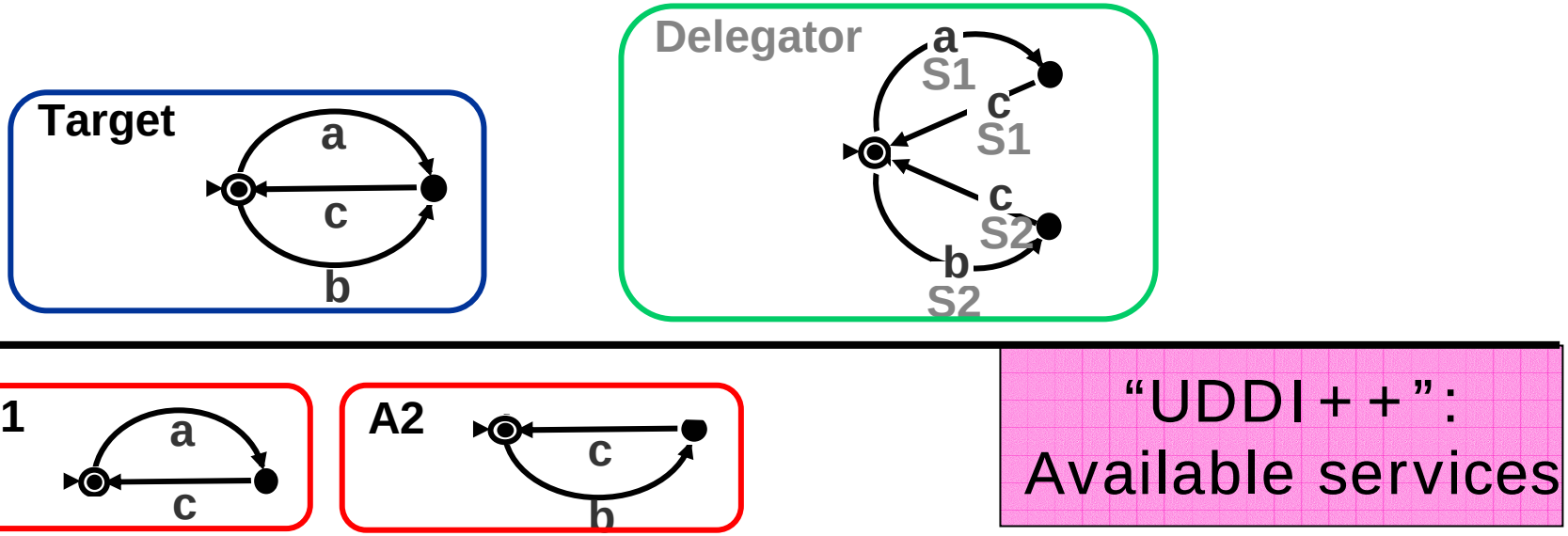


# Main result on Roman Composition

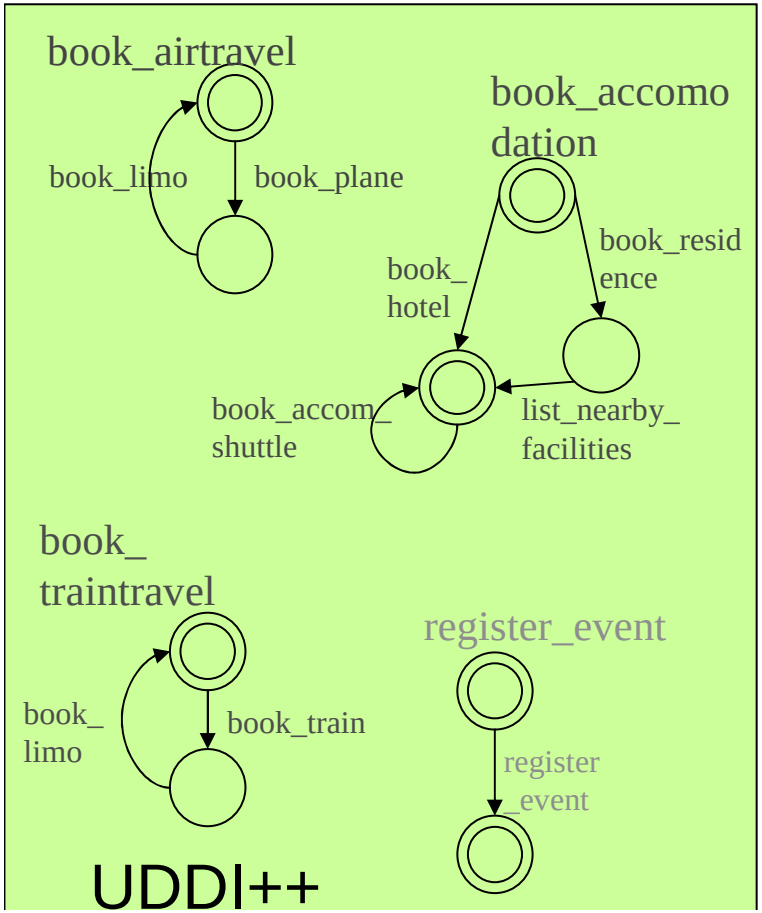
- Note: In some cases, delegator is not simply a labeling of target machine

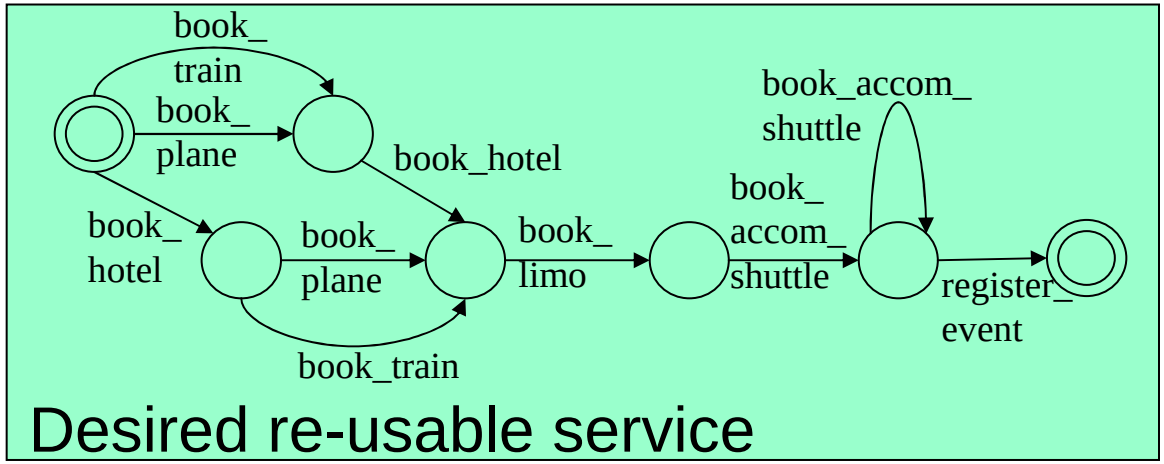


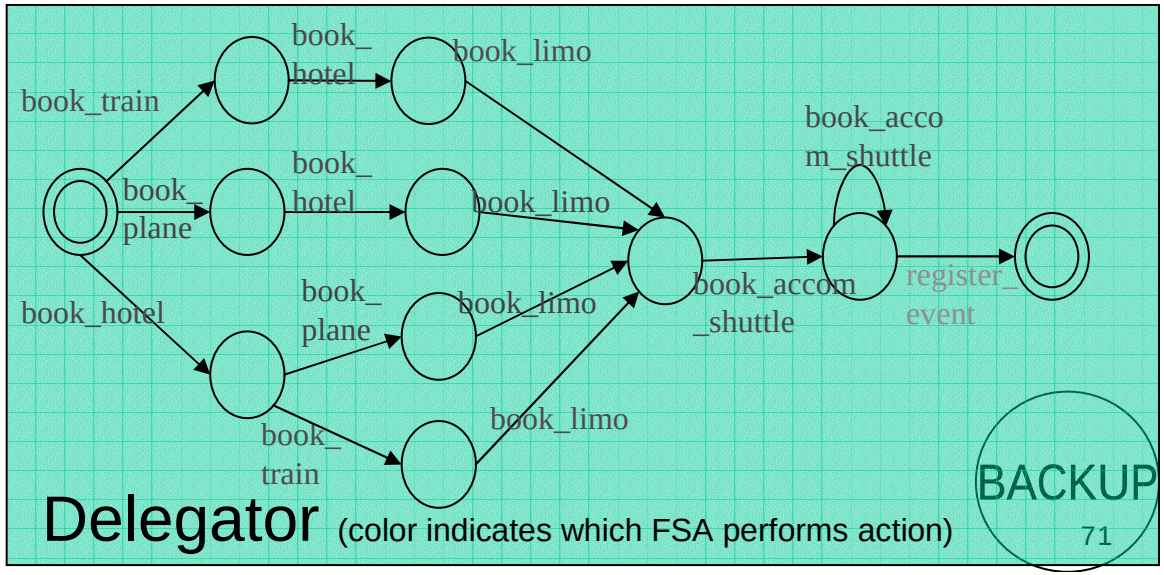

- Thm: Can determine if a delegator exists, and build it, in EXPTIME [Berardi et al ICSOC'03]
  - We'll examine 2 proof techniques

# Second example of Roman Delegator

- Delegator is not simply a labeling of desired service



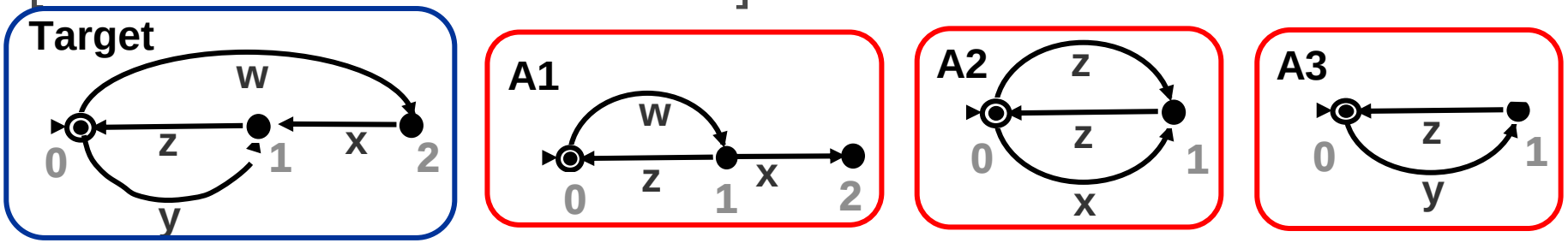



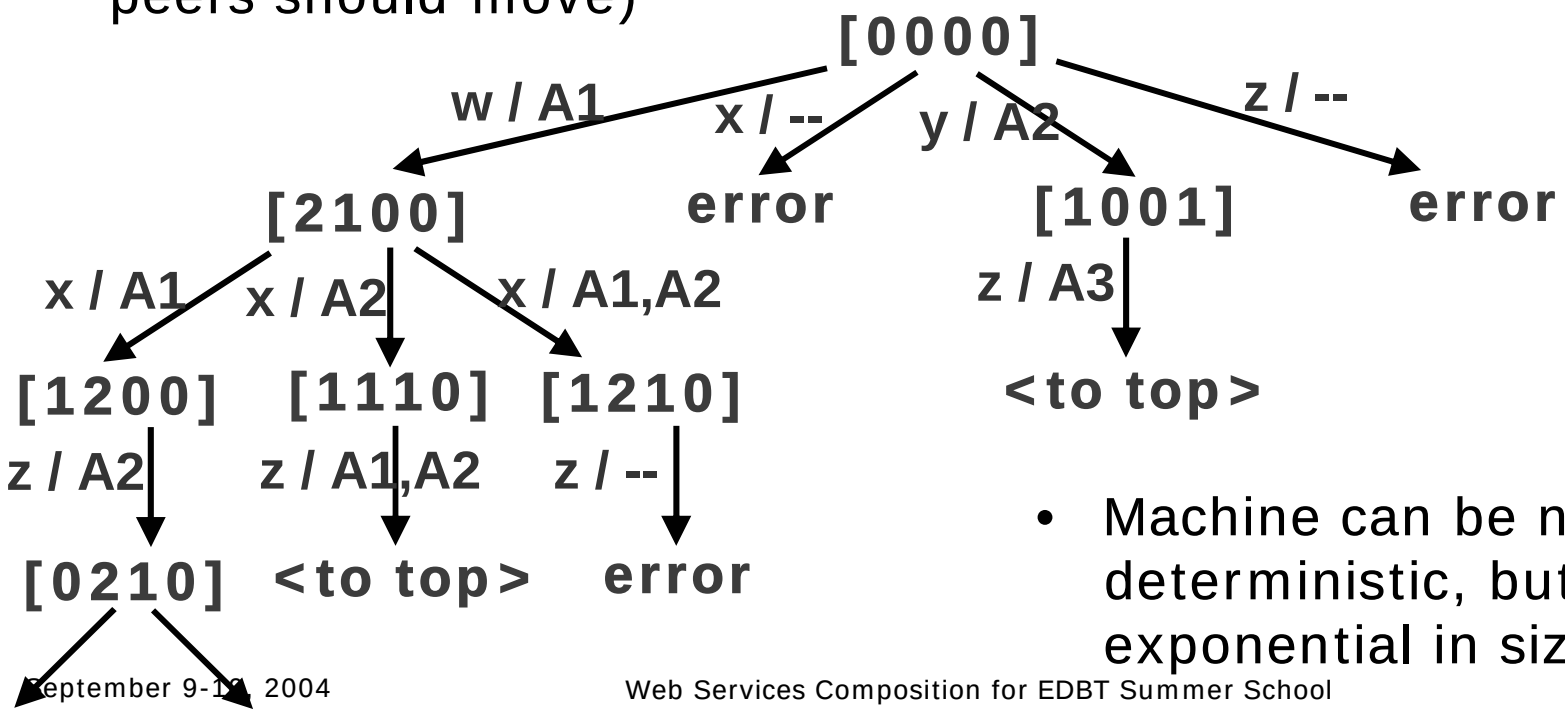

BACKUP

# An automata-based proof of the Roman composition result

[Gerede et. al. ICSOC '04]




- Building a "product" machine (Mealy, but outputting what peers should move)




- Machine can be non-deterministic, but at most exponential in size

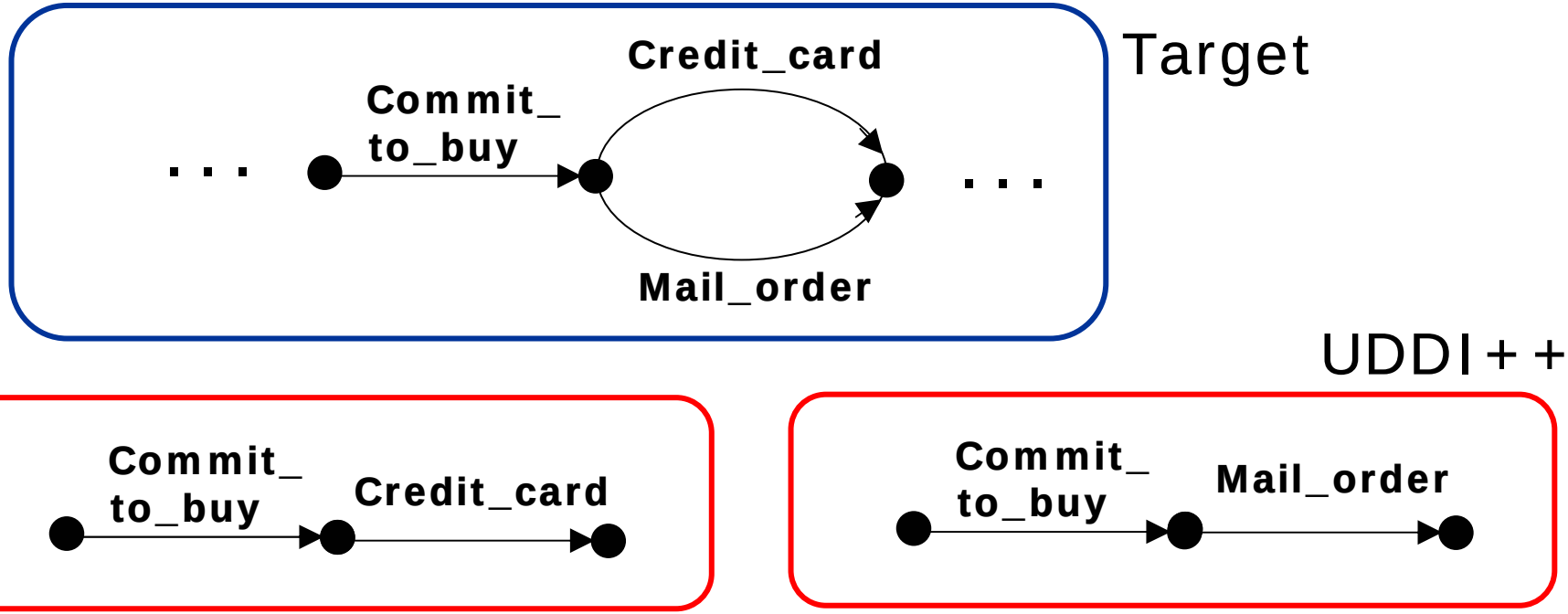
# Proof outline

- Lemma: If there is a delegator, then there is one that is a sub-machine of the product machine
  - Use a splicing argument
- Thm: Can construct delegator, if one exists, in time

$$O( |Target| \times |A1| \times \ldots \times |An| )$$

  - Lemma: can remove “bad” states
  - Lemma: From what remains, any deterministic sub-machine is a delegator

# Enriching the Roman Model: Look Ahead [Gerede et. al. ICSOC '04]




- There is no composition in this case
- Can develop a theory of "k look-ahead" delegators
- This defines a non-collapsing hierarchy, and there are examples needing "infinite" look-ahead
- Can reduce existence of k look-ahead delegator to 0 look-ahead case in PTIME; so finding a k look-ahead delegator is still in EXPTIME

# A Logic-based Proof of Roman Composition Result

[Berardi et. al. ICSOC'03]

- (Deterministic) Propositional Dynamic Logic (DPDL)
  - Well-known modal logic for reasoning about programs
  - [e.g., short intro in The Description Logic Handbook]

Formulas: built from "propositional letters" (or "fluents")

$$\varphi \;==\; f \;|\; \neg\varphi \;|\; \varphi \wedge \varphi' \;|\; [r]\,\varphi \;|\; \langle r \rangle\,\varphi$$

"f is true in current state"

"starting from current state, every execution of r leads to state in which φ is true"

"starting from current state, some execution of r leads to state in which φ is true"

Programs: built from atomic Programs

$$r \;==\; P \;|\; r \cup r' \;|\; r;r' \;|\; r^* \;\; (\,|\; \varphi? \;|\; r^- \,)$$

# Semantics for (D)PDL

- Models have the form (a Kripke structure)

$$M = (S, \{R_P\}, \Pi)$$

Set of "states"

For each atomic program P, a binary relation over S – "transition relation"

$\Pi : S \rightarrow 2^{\{fluents\}}$

(for each state s, which fluents are true in s)

$M, s \models f$ if $f \in \Pi(s)$

$M, s \models \varphi \wedge \varphi'$ if $M, s \models \varphi$ and $M, s \models \varphi'$

. . .

$R_P \subseteq S \times S$ for each atomic program P

$R_{r;r'} = R_r \circ R_{r'}$

. . .

# Key results for (D)PDL

Bell Labs
Innovations for Lucent Technologies

- $M$ *satisfies* $\varphi$ if for some state s, $M$,s $\models \varphi$

  $\varphi$ is *valid* in $M$ if for each state s, $M$,s $\models \varphi$

  $\Gamma \models \varphi$ if $\varphi$ is valid in every $M$ that satisfies $\Gamma$

- Thm:
  - Satisfiability is ExpTime complete
  - $\varphi$ is satisfiable iff there is a model with size at most exponential in $|\varphi|$ (and this can be constructed in exponential time)
- Tree model property: each model can be "unwound" to form a model that has a tree structure

# Embedding of Roman Composition problem into DPDL

Bell Labs
Innovations for Lucent Technologies

- Let [u] denote [ $(\cup_{a \in \Sigma} a)^*$ ]
- For each FSM Ai = $(\Sigma, S_i, s^0_i, \delta_i, F_i)$ (including target)
  - Include fluents for each state, and for $F_i$ (meaning that Ai is in a final state)
  - Also include fluent $moved_i$, meaning that Ai just moved
- Formulas include for Target FSM A0
  - $[u](s \rightarrow \neg s')$ for s, s' distinct
  - $[u](s \rightarrow \langle a \rangle \mathbf{true} \wedge [a](s'))$ for each a such that $\delta_0(s,a) = a'$
  - $[u](s \rightarrow [a]\mathbf{false})$ if $\delta_0(s,a)$ is undefined
  - $[u](F_0 \rightarrow \bigvee_{s \in F_0} s)$
- Formulas for source machines Ai include, e.g.,
  - $[u](s \rightarrow [a]((moved_i \wedge s') \vee (\neg moved_i \wedge s))$
- Also, "all FSM's start in start state", "at least one source FSM moves", "all FSM's end in final state"

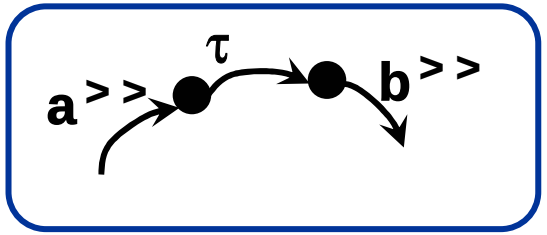
# Summary on DPDL for Roman Composition

Bell Labs
Innovations for Lucent Technologies

- Thm: The DPDL formula constructed as above is satisfiable iff there is a delegator
- Cor: Existence of delegator can be determined (and constructed) in EXPTIME
- Note: There is a close correspondence between (variants of) PDL and (variants of) Description Logics
  - Can use highly optimized tableaux-based algorithms from Description Logics to construct delegators

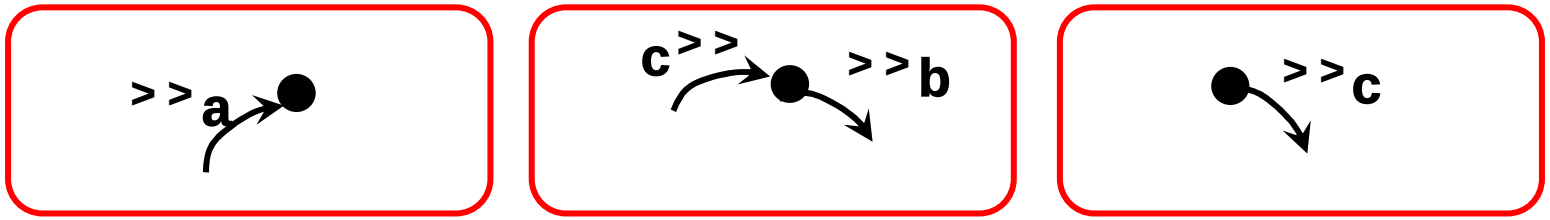
# Enriching the "Roman" Model: Services that Delegate

[Berardi et. al.'04]




Target Service

UDDI++




- New features
  - One service can "initiate" a request against another service
  - Target service can "ignore" a sequence of actions by the services (indicated by a τ move)
- Can generalize previous EXPTIME results
  - Using different variant of Propositional Dynamic Logic
  - Ability to work with richly incomplete information is especially appealing

# Outline

- Introduction
- Standards
- Perspectives on “Composition”
  - Mealy machine/conversations
  - Roman
  - “Semantic”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# Situation Calculi and PSL: Logics with Actions and Model of "World"

- Formal foundations for "semantic" models of services

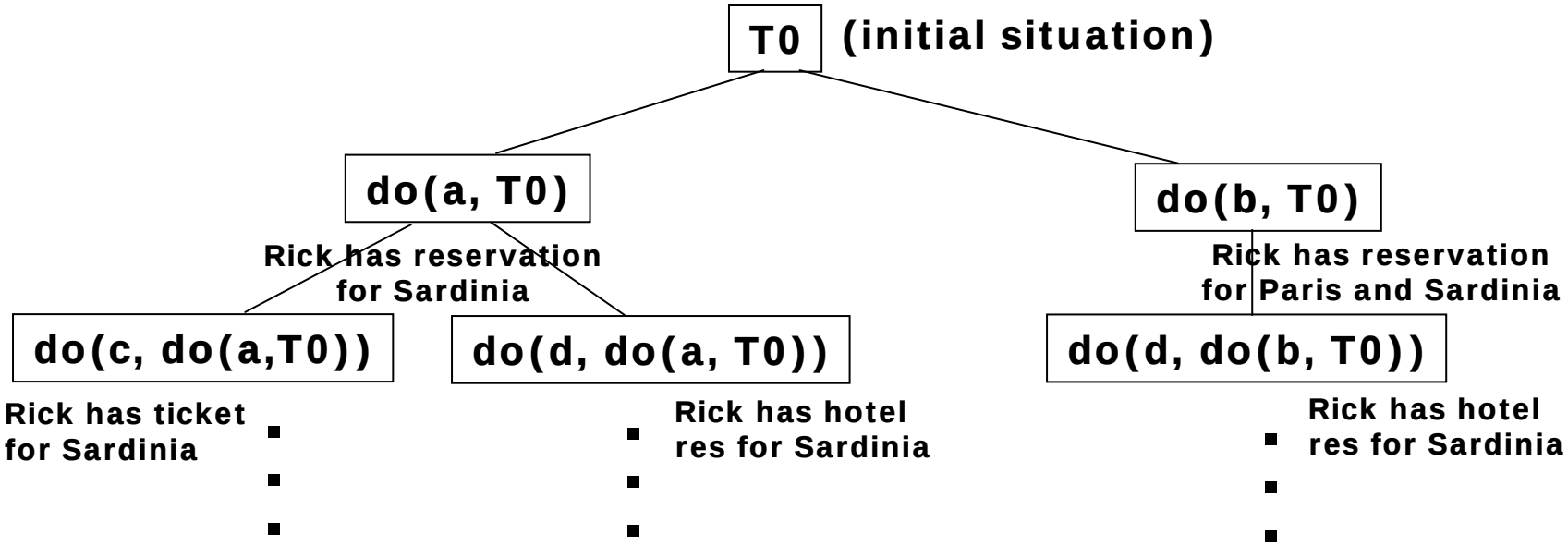
Situation Calculi [cf Reiter's book "Knowledge in Action"]

- This formalism used widely in AI for reasoning about actions
  - It can provide the semantics underlying OWL-S executions

- Models: Trees whose nodes correspond to "situations"
- "Fluents":
  - Propositions (and predicates) which hold in situation
    - f(x1,…,xn, s) denotes that f(x1,…,xn) holds in situation s
  - Used to test pre-conditions, record effects (e.g., for OWL-S)

- An "unraveled" model of PDL can be viewed as a propositional situation calculus model

# Typical model in Situation Calculus




## Typical axiomatization

- Axioms to describe initial situation T0
- Action precondition axioms *Poss*(a,s), for each primitive action
- Successor state axioms, one for each fluent f:
  - Describes when f(x1…xn, do(a,s)) holds
- Unique name axioms for the primitive actions
- Some foundational, domain independent axioms
  - Second-order closure axiom used to ensure that all branches are finite

# Process Specification Language (PSL)

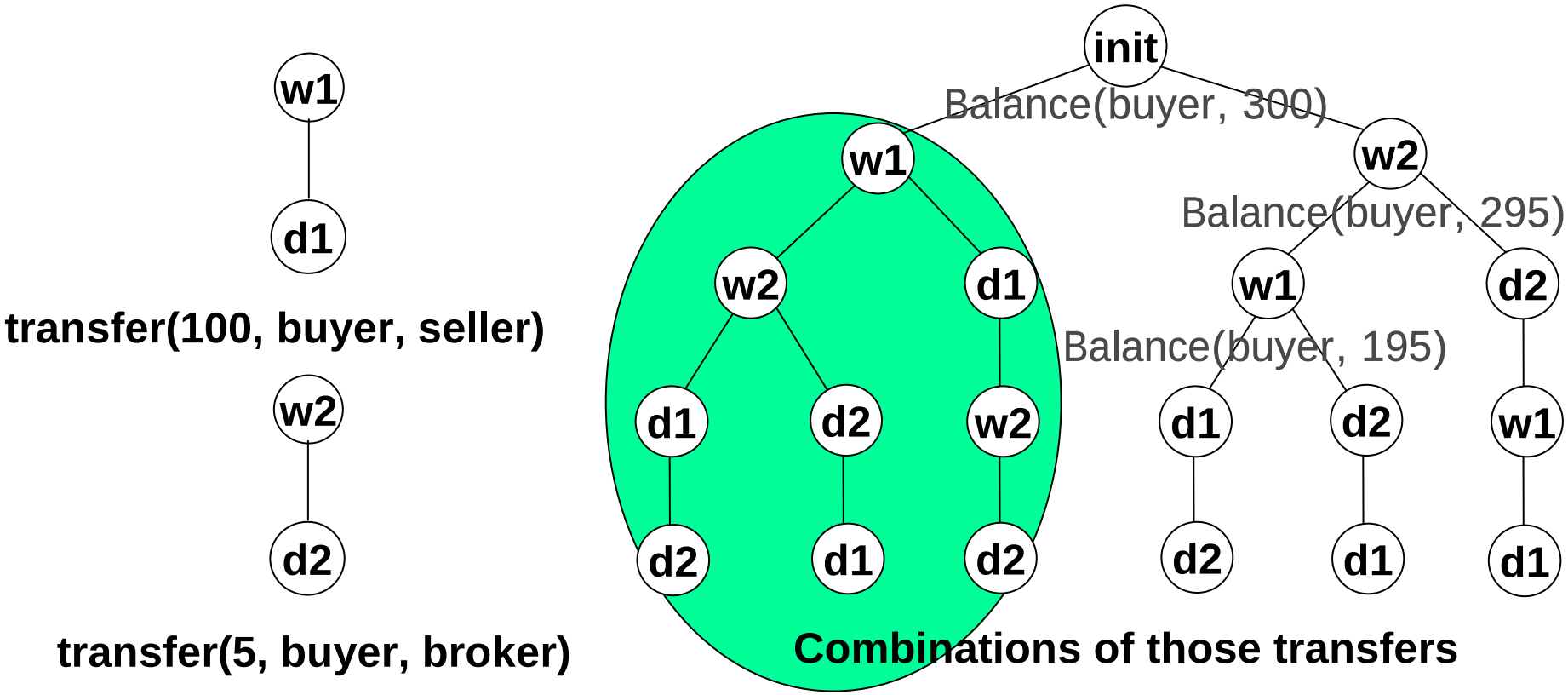
Bell Labs
Innovations for Lucent Technologies

- An ISO standard, originally for sharing manu specs
- Models:
  - Essentially, trees whose nodes correspond to atomic actions
- "Fluents":
  - Propositions (and predicates) which hold between actions
- Vocabulary of PSL (richer than typical situation calculi)
  - Various layers of reified predicates, e.g.,
    - *activity* (a), *activity_occurrence* (o), *timepoint* (t)
    - *occurrence_of* (o, a), *min_precedes* (o1, o2, a), *precedes*(o1, o2, a)
    - *holds* (f, o), *prior* (f, o)
  - Activities and occurrences identified using variables and terms (e.g., *withdraw*(x, y))
- Formal semantics given in first-order logic
  - Family of axioms associated with each layer/variant of PSL

# PSL: Simple illustration of the model theory

**Atomic activities:**

- **w1 = withdraw (100, buyer)**
- **d1 = deposit (100, seller)**
- **w2 = withdraw (5, buyer)**
- **d2 = deposit (5, broker)**




- Can add constraints, e.g., that w1 must precede w2
- Can use FOL inference or domain-specific reasoning

# Expressive power of PSL

Bell Labs
Innovations for Lucent Technologies

## Examples of PSL

- Activities as terms: $\forall$**x,y,z activity( transfer(x,y,z) )**
- Composition relationships:
  $\forall$ **x,y,z subactivity(withdraw(w,y), transfer(x,y,z)**
  $\forall$ **a,y ( a = buy_product(y)** $\supset$ **x,z subactivity( transfer(x,y,z) , a ) )**
- Process description for *buy_product*
  $\forall$ **o,x occurrence_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 ) )**

## Situation Calculi typically less expressive

- Cannot have variables for composite activities/occurrences
- Cannot have variable ranging over fluents

## But: PSL based on first-order entailment; Situation calculi typically use a second-order closure axiom

# (Some) sentences characterizing a financial example

- Activities as terms

  **∀ x activity( buy_products(x) )**
  **∀ x,y,z activity( transfer(x,y,z) )**
  **∀ x,y activity( withdraw(x,y) )**
  **∀ x,y activity( deposit(x,y) )**

- Composition relationships

  **∀ a,y ( a = buy_product(y) □ ⊃ x,z subactivity( transfer(x,y,z) , a ) )**
  **∀ x,y,z subactivity( withdraw(x,y), transfer(x,y,z) )**
  **∀ x,y,z subactivity( deposit(x,z) ), transfer(x,y,z) )**

- Process description for *buy_product*

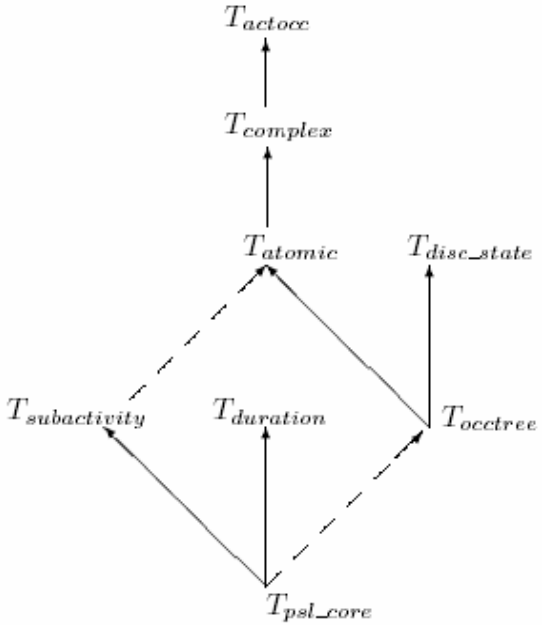
  **∀o,x occurrence_of(o, buy_product(x) ) ⊃ □**
  **∃o1,o2,y,z,w,v (occurrence_of( o1, transfer(y,x,z)**
  **□ ∧ occurrence_of(o2, transfer(w,x,v)**
  **□ ∧ subactivity_occurrence(o1, o )**
  **□ ∧ subactivity_occurrence(o2, o ) )**

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

BACKUP

# PSL: Layers of Ontology and Axioms

Bell Labs
Innovations for Lucent Technologies




- Base layer has only activities, occurrences, ordered time
- Additional layers build out, e.g., "composite activity", duration, legal, possible, …
- Axioms included for each layer

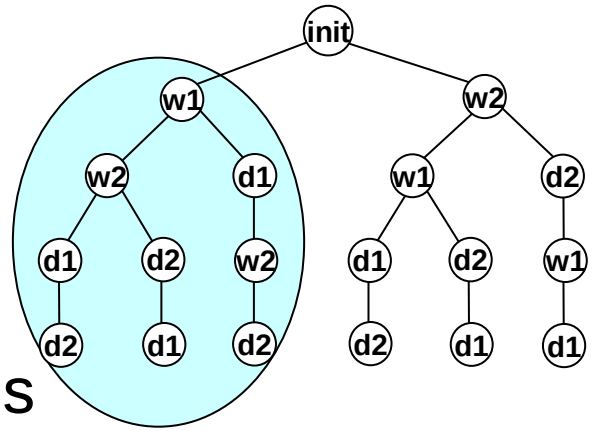
| | | |
|---|---|---|
| $T_{pslcore}$ | $activity(a)$ | $a$ is an activity |
| | $activity\_occurrence(o)$ | $o$ is an activity occurrence |
| | $timepoint(t)$ | $t$ is a timepoint |
| | $object(x)$ | $x$ is an object |
| | $occurrence\_of(o, a)$ | $o$ is an occurrence of $a$ |
| | $beginof(o)$ | the beginning timepoint of $o$ |
| | $endof(o)$ | the ending timepoint of $o$ |
| | $before(t_1, t_2)$ | timepoint $t_1$ precedes timepoint $t_2$ on the timeline |
| $T_{subactivity}$ | $subactivity(a_1, a_2)$ | $a_1$ is a subactivity of $a_2$ |
| | $primitive(a)$ | $a$ is a minimal element of the *subactivity* ordering |
| $T_{atomic}$ | $atomic(a)$ | $a$ is either primitive or a concurrent activity |
| | $conc(a_1, a_2)$ | the activity that the concurrent composition of $a_1$ and $a_2$ |
| $T_{occtree}$ | $successor(a, s)$ | the element of an occurrence tree that is the next occurrence of $a$ after the activity occurrence $s$ |
| | $legal(s)$ | $s$ is an element of a legal occurrence tree |
| | $initial(s)$ | $s$ is the root of an occurrence tree |
| | $earlier(s_1, s_2)$ | $s_1$ precedes $s_2$ in an occurrence tree |
| | $poss(a, s)$ | there exists a legal occurrence of $a$ that is a successor of $s$ |
| $T_{disc\_state}$ | $holds(f, s)$ | the fluent $f$ is true immediately after the activity occurrence $s$ |
| | $prior(f, s)$ | the fluent $f$ is true immediately before the activity occurrence $s$ |
| $T_{complex}$ | $min\_precedes(s_1, s_2, a)$ | the atomic subactivity occurrence $s_1$ precedes the atomic subactivity occurrence $s_2$ in an activity tree for $a$ |
| | $root(s, a)$ | the atomic subactivity occurrence $s$ is the root of an activity tree for $a$ |
| $T_{actocc}$ | $subactivity\_occurrence(o_1, o_2)$ | $o_1$ is a subactivity occurrence of $o_2$ |
| | $root\_occ(o)$ | the initial atomic subactivity occurrence of $o$ |
| | $leaf\_occ(s, o)$ | $s$ is the final atomic subactivity occurrence of $o$ |
| $T_{duration}$ | $timeduration(d)$ | $d$ is a timeduration |
| | $duration(t_1, t_2)$ | the timeduration whose value is the "distance" from timepoint $t_1$ to timepoint $t_2$ |
| | $lesser(d_1, d_2)$ | the linear ordering relation over timedurations |

BACKUP

 [Gruninger '03] 

# “Programming” as constraints

- Golog: a “programming language” for the situation calculus
  - “Constructs” such as
    - Sequence: $\delta_1$ **;** $\delta_2$
    - Conditional: **if** $\varphi$ **then** $\delta_1$ **else** $\delta_2$ **endif**
    - Loop: **while** $\varphi$ **do** $\delta$ **endWhile**
  - Interpreted as temporal constraints on permitted paths
    - E.g., “ w1; w2 ” is satisfied by 3 of the 6 branches
  - ConGolog: generalization to incorporate concurrent activities




- Two-tier “program” specification
  - First tier: use the “constructs” from above
    - Identifies a set of possible execution sequences
  - Second tier: arbitrary constraints
    - Further restricts set of possible execution sequences

# Examples of Golog expressions

Bell Labs
Innovations for Lucent Technologies

- Golog program fragments

```
while x(hotel(x) ∧ goodLoc(x)) do
        checkAvailability(x, dest, dDate, rDate)
endwhile
```

- Personalization constraints

```
If ∧ hotelAvailable(dest,dDate,rDate) then
   bookB&B(cust, dest, dDate, rDate)
```

- Procedure definition

```
proc  Travel( cust, origin, dest, dDate, rDate, purpose);
   if registrationRequired then Register;
   BookTransport(cust, origin, dest, dDate, rDate);
   BookAccommodations(cust, dest, dDate, rDate);
   UpdateExpenseClaim(cust);  Inform(cust);
endproc
```

BACKUP

# Composition with OWL-S and ConGolog interpreter

[McIlraith+Son '02]

Framework based on two tiers:

- *Generic programs* and *Customization via constraints*

- Start with family of atomic OWL-S services, with pre-conditions and effects
- Write Golog program capturing constraints on generic flow of control and parameter passing
- Write additional constraints (in situation calc) to capture personalization
  - Typically express them as Horn formulas
- Use ConGolog interpreter to find one (or more) branches in situation calc tree that satisfies all constraints
  - "Middle-Ground Optimization" based on gathering data in advance of world-altering activities

# Automatic Composition for OWL-S via Petri nets [Narayanan+McIlraith '02]

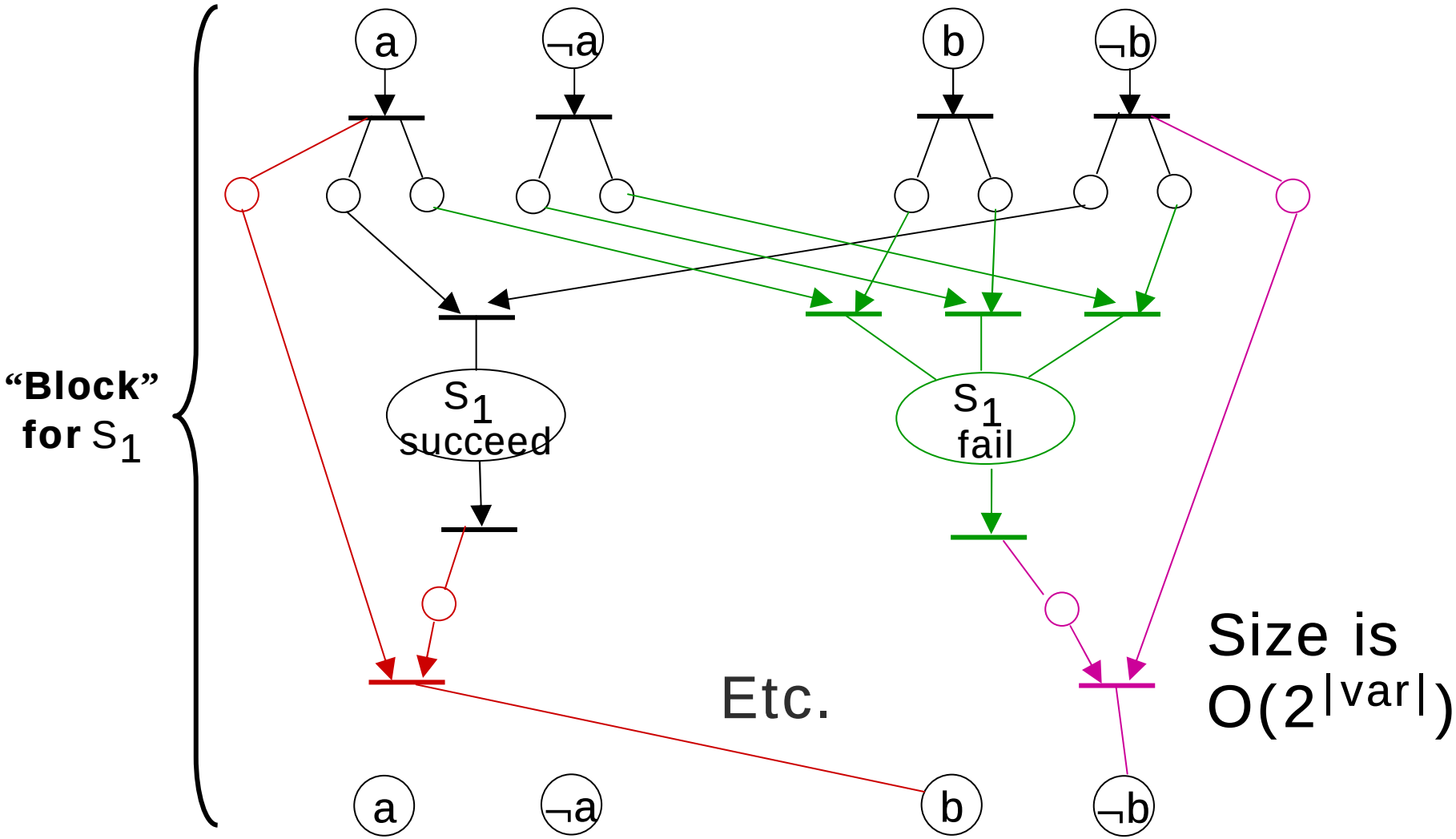
- Can simulate OWL-S via 1-safe Petri nets
  1. For set of atomic e-services, create Petri Net that represents all possible (single-use) combinations of them
  2. Specify desired goal as a state of this Petri Net
  3. Determine if this goal state is reachable
- In this framework reachability is PSPACE-complete in size of Petri net
  - Different complexities depending on OWL-S constructs permitted
  - Petri net itself may be exponential in size of atomic e-services
  - Heuristics can be used to avoid full construction

# Illustration of OWL-S automated composition via Petri nets

- Simplifying assumption for this example
  - Assume only propositional variables
- Given:
  - Finite set of OWL-S atomic services $S_1$, …, $S_n$
  - Pre-condition (e.g., in DNF)
  - Goal – the desired post-condition (again in DNF)
- Determine whether there is a sequence $S_{i_1}$ , …, $S_{i_j}$ that:
  - When started under the pre-condition it achieves the goal
  - No repeats of an atomic service
  - Note: in an execution, some services in the sequence may fail
- Technique: Build a Petri-net that captures
  - Impact of pre-conditions
  - Impact of (attempted) execution of an atomic service
  - Satisfaction of post-condition as some place in the Petri-net
- If the post-condition can be reached, then goal achievable

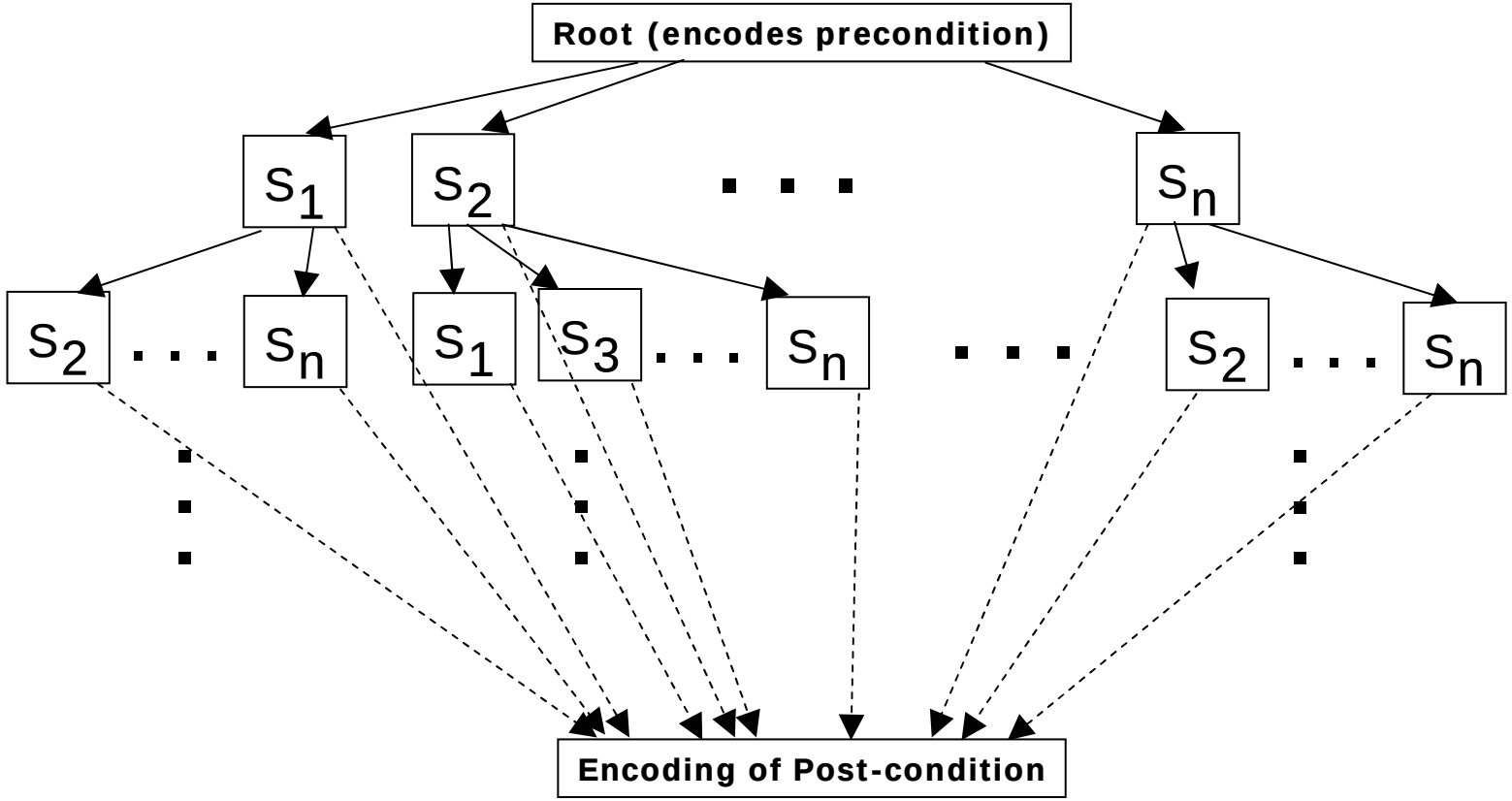
# Block for one atomic service

- Assume just propositional variables a, b
- Assume service $S_1$ has { Pre-condition: $a \wedge \neg b$; Post-condition: if a then b becomes true




Size is $O(2^{|var|})$

# Assembling the full Petri-net



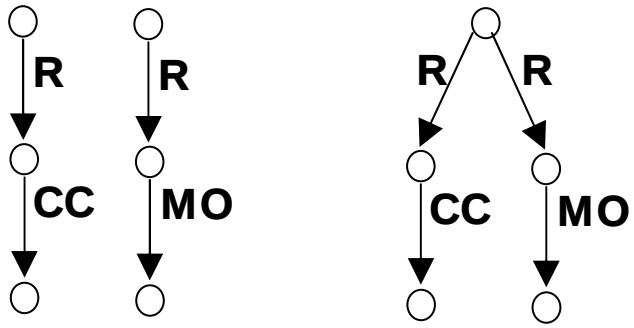

- Size is $O(n!) = O(2^n)$ x size of service block;
  Overall size is $O(2^{|\text{problem statement}|})$
- Reachability in 1-safe Petri-nets is PSPACE
- Overall decision procedure is EXPSPACE

# 2 Flavors of Composition

- Example: reserving and paying for a plane ticket
  - If cheap, use CreditCard; if expensive use MoneyOrder

"All Knowledge in Advance"

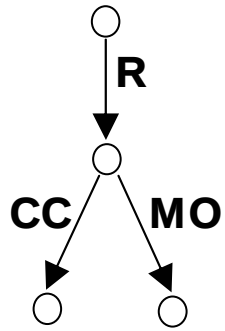
- Know in advance whether the ticket is cheap or not




- Can describe in a linear temporal logic
  - "After R there is CC or MO"
- Can use "transition system" as underlying framework

"Knowledge obtained along the way"

- Find out in middle of execution whether ticket is cheap or not
- Might be learned from external world, or from user




- Need a branching logic to describe
  - "After R there is a branch with CC and a branch with MO"
- Need to use something richer
- Cf. the k-look-ahead results

# Outline

Bell Labs
Innovations for Lucent Technologies

- Introduction
- Standards
- Perspectives on “Composition”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# Analysis of Web Services

Bell Labs
Innovations for Lucent Technologies

- Service properties:
  - Statements on functional logic, service guarantees, …
  - Statement on execution (deadlock, safety, …)
- Analysis may be in more demand:
  - Dynamic composition
  - Difficulties in testing
  - Immature service oriented development environments
- Possible approaches:
  - Static: model checking, theorem proving, …
  - Runtime monitoring

# Model Checking

Bell Labs
Innovations for Lucent Technologies

- The target of interest is given as a state transition system
- Properties are specified in some temporal logic, e.g., linear temporal logic (LTL), branching time logic (CTL*),..
  - Model checking within $2^{O(||\varphi||)}|\text{transition system}|$
- The entire state space is examined systematically
  - Explicit (automata techniques): e.g., SPIN, CWB, …
  - Symbolic, using forward or backward fixpoint: e.g., SMV
    - BDDs can be used to symbolically represent sets of states
- Model checking and compositions: The challenge is to map a composition analysis problem to a model checking problem
  - May need an approximation of the composition model

# Approaches Examined

Bell Labs
Innovations for Lucent Technologies

- Model checking
  - With finite state machines
    [Fu et al WWW'04] [Foster et al ASE'03]
  - With process algebra [Koshikina-van Breugel 2003]
- OWL-S services analysis with Petri nets
  [Narayanan-McIlraith WWW'02]
- Rule-based services [Deutsch et al PODS'04]
- Analysis of a Natural Fragment of CTR
  [Davulcu et al PODS'98]
- Dynamic verification of protocol compliance in commitments [Venkatraman and Singh '99]
- An approach for queues and spawning, based on counters [a work in progress]

# Explicit Model Checking

Bell Labs
Innovations for Lucent Technologies

- Transition system $R$ and initial states $I$
  → automaton $M_R$

- Property $P$ → $\neg P$ → automaton $M_{\neg P}$

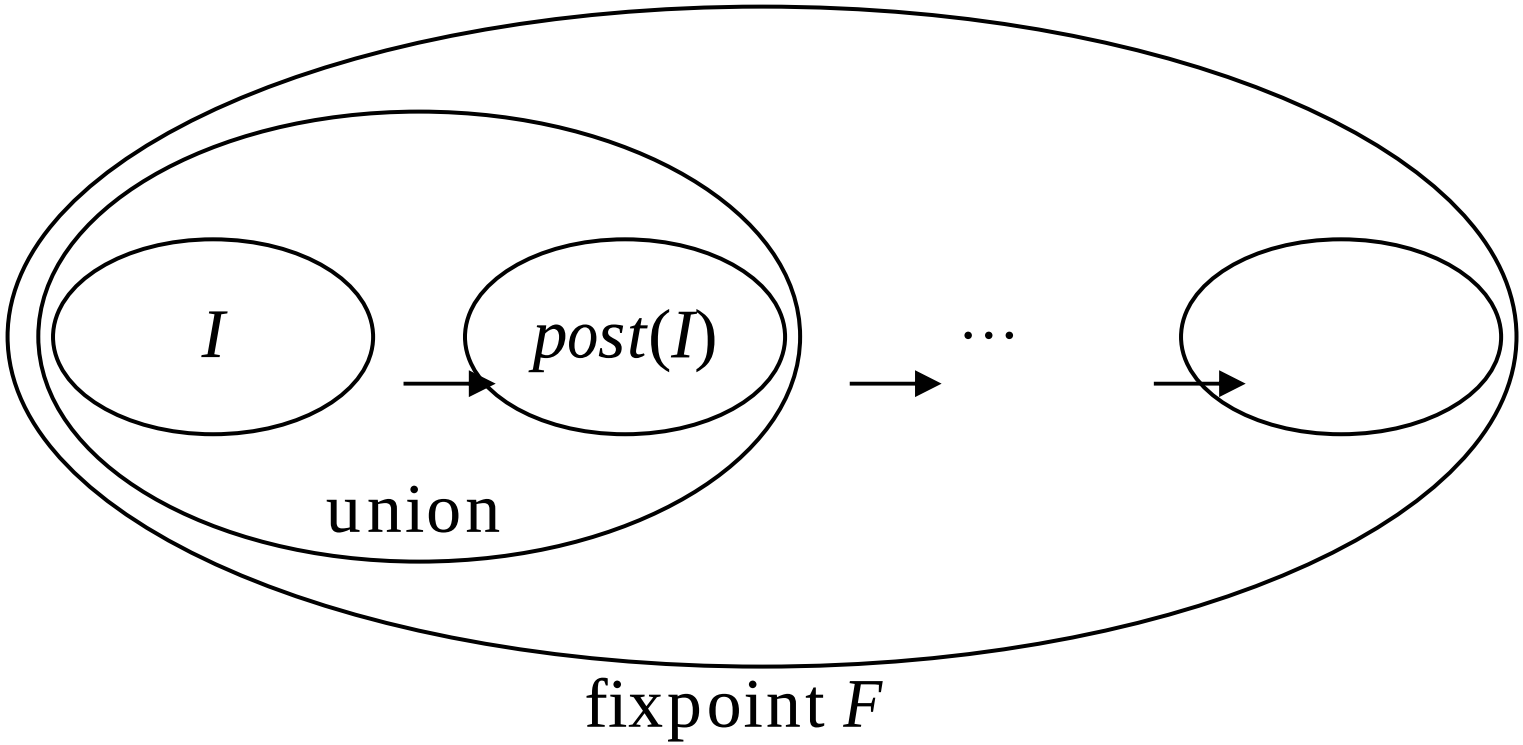
- Question: $M_R \cap M_{\neg P} = \emptyset$ ?

BACKUP

# Symbolic Model Checking: Forward Fixpoint

Bell Labs
Innovations for Lucent Technologies

Initial states $I$, Transition relation $R$, Property $P$



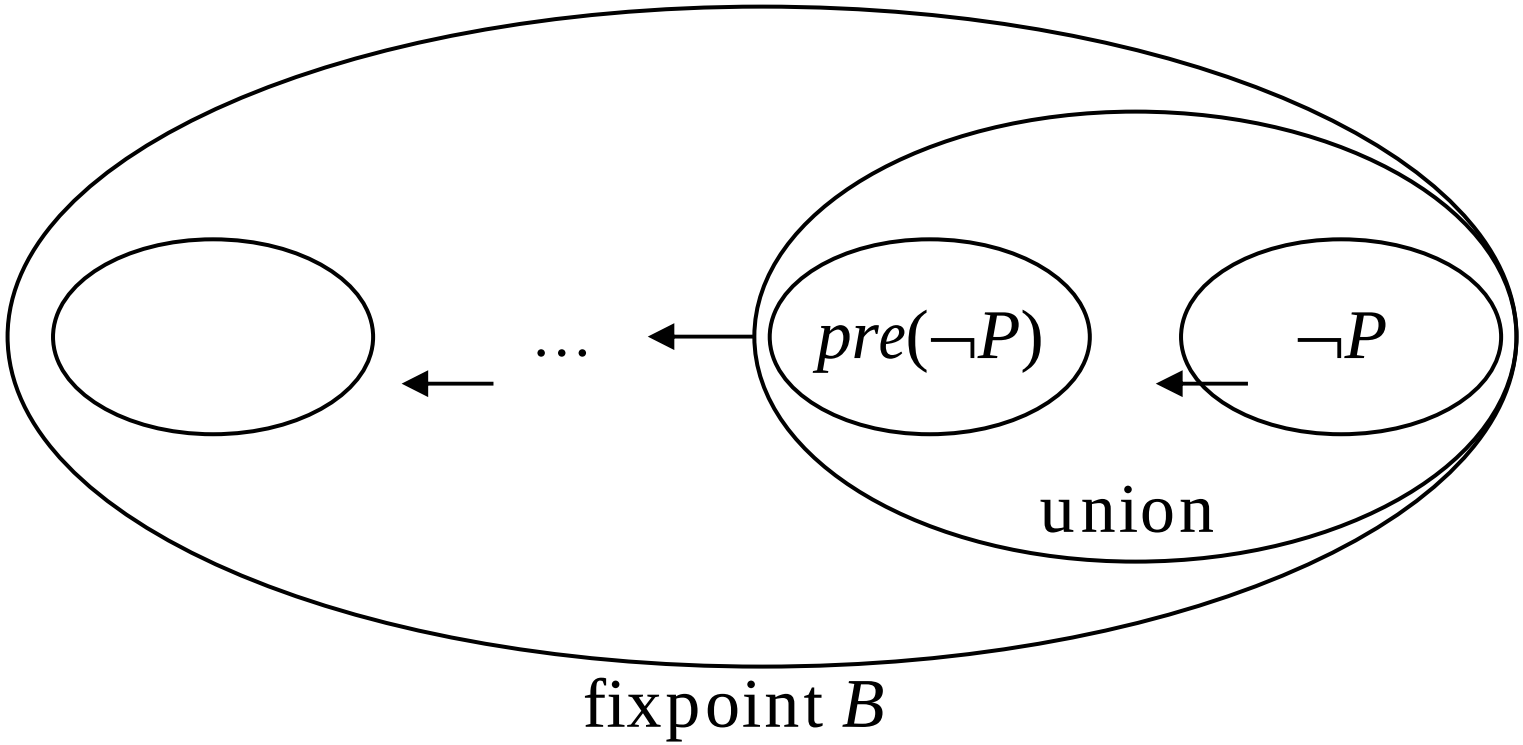

Question: $F \subseteq P$ ?

BACKUP

# Symbolic Model Checking: Backward Fixpoint

Bell Labs
Innovations for Lucent Technologies

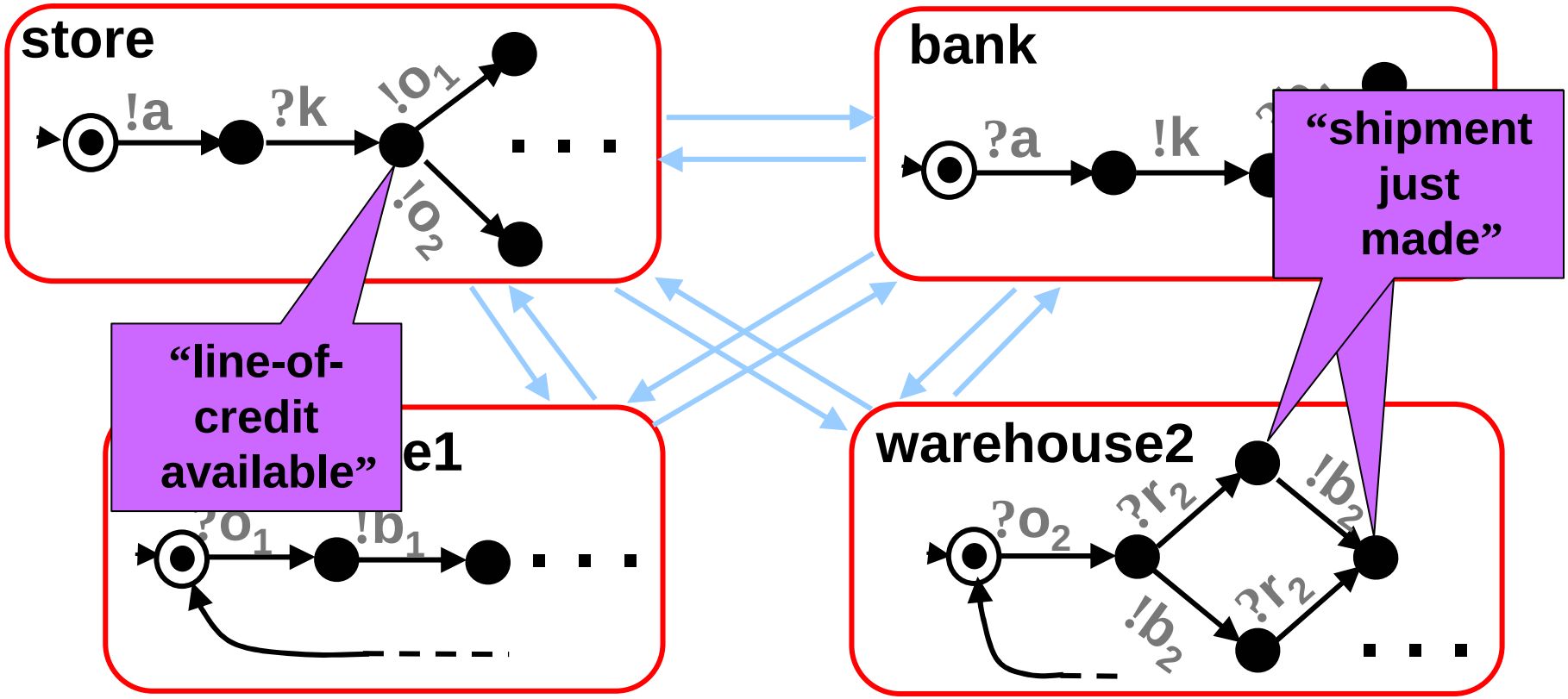
Initial states $I$, Transition relation $R$, Property $P$




Question: $B \cap I = \emptyset$ ?

BACKUP

# Verifying Temporal Properties of Mealy Compositions



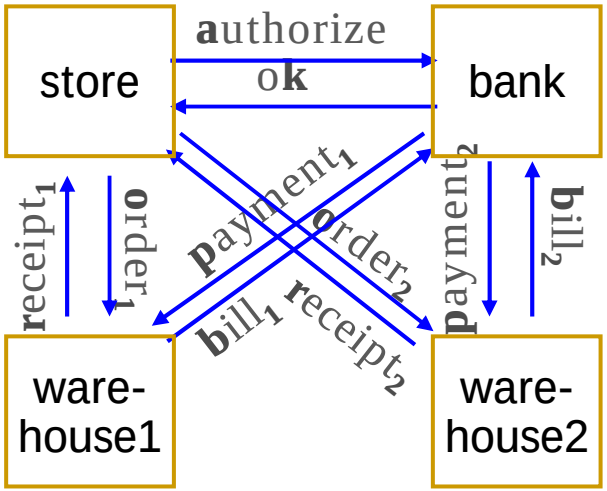

- Label states with propositions
  - Level of indirection between states and "observables"
- Express temporal formulas, e.g.,
  - "shipment just made" only after "line-of-credit avail"

# Results on Temporal Verification

- Long history, e.g., [Clarke et.al. '00]
- E.g., verification for fsa's and propositional LTL
  - Complexity: PSPACE in size of formula + fsa
    linear time in size of fsa
- Application to Mealy compositions
  - Results apply to open and closed case
  - Bounded queues
    - Composition can be simulated as Mealy machine
    - Verification is decidable
    - Standard techniques to reduce cost
  - Unbounded queues
    - In general, undecidable
    - Approximation techniques can be applied

# Verification of Web services

- Model execution as finite state machines
  [Foster et al ASE '03] [Fu et al WWW '04]
- [Fu et al WWW '04] Verifying conversation among a set of BPEL composite services




conversation

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | k | $o_1$ | $o_2$ | $b_1$ | $p_1$ | $r_1$ | $r_2$ | $b_2$ | $p_2$ | | | |

LTL properties: Every **a**uthorize followed by some bill?

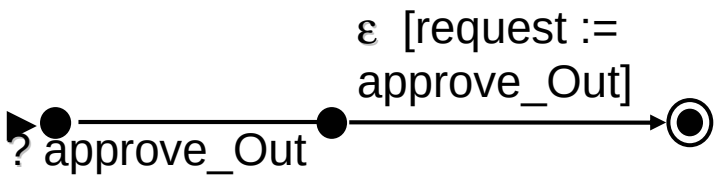
- Approach: model BPEL services as "guarded automata"
  - Mealy machines + conditions on transitions + XML messages + XML local data
  - Translated to Promela (input language of SPIN)

# BPEL to Guarded Automata

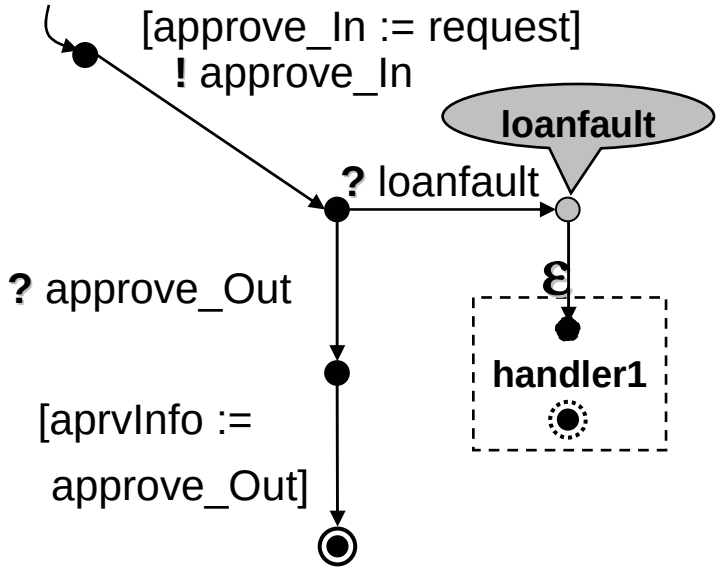
## [Fu et al WWW '04]

- Each atomic activity → an automaton with single entry, single exit

```
<receive …
    operation = "approve"
     variable = "request" />
```



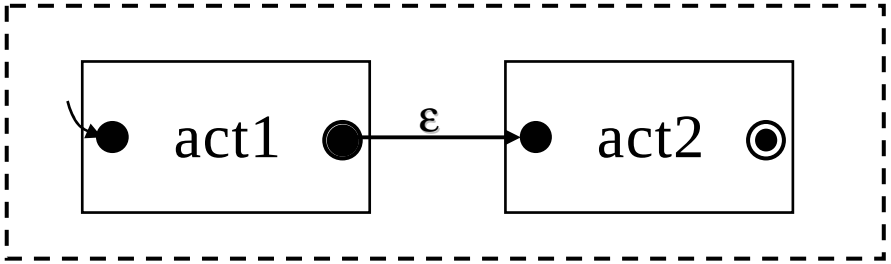

```
<invoke operation="approve",
        invar="request",
    outvar="aprvInfo" >
  <catch  faultname="loanfault">
   < … handler1 … />
  </catch>
</invoke>
```



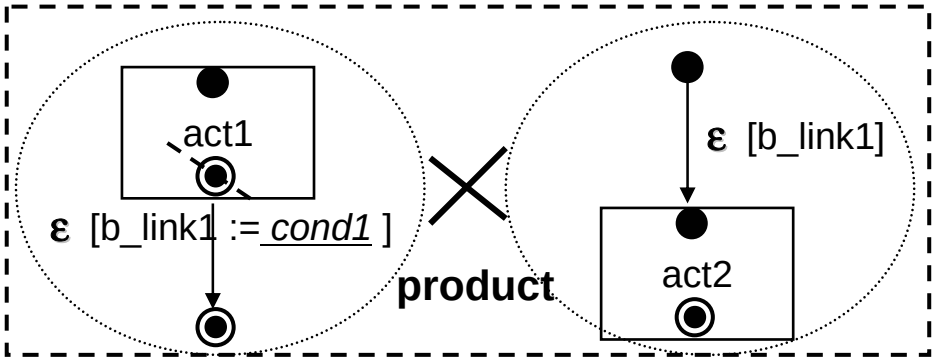
# BPEL to Guarded Automata

- Control flow constructs: assemble Mealy machines

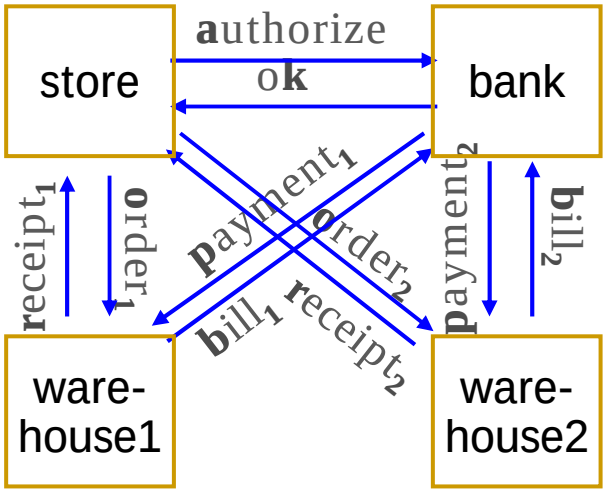
```
<sequence .../>
    <... act1.../>
    <... act2.../>
</sequence .../>
```




```
<flow .../>
    <... act1 ...>
        <source linkname = "link1"
         condition = "cond1  .../>
    </act1 >
    <... act2 ... >
        <target linkname = "link1"  />
    </act2 >
</flow .../>
```



# Partial and Complete Verification (with SPIN)




conversation

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | k | $o_1$ | $o_2$ | $b_1$ | $p_1$ | $r_1$ | $r_2$ | $b_2$ | $p_2$ | | | |

LTL properties: Every **a**uthorize followed by some bill?

- Promela: input language of SPIN
  - Concurrent processes communicating with bounded queues
- Each guarded automaton → a Promela program
  - Bound the queue length → partial verification
- Synchronizable → complete verification

# Synchronizability Analysis Results

| Web service | | (Guarded automata) Size | | | Synchronizable? |
|---|---|---|---|---|---|
| Source | Name | #msg | #states | #trans. | |
| ISSTA'04 | SAS | 9 | 12 | 15 | yes |
| IBM Conv. Support Project | CvSetup | 4 | 4 | 4 | yes |
| | MetaConv | 4 | 4 | 6 | no |
| | Chat | 2 | 4 | 5 | yes |
| | Buy | 5 | 5 | 6 | yes |
| | Haggle | 8 | 5 | 8 | no |
| | AMAB | 8 | 10 | 15 | yes |
| BPEL spec | shipping | 2 | 3 | 3 | yes |
| | Loan | 6 | 6 | 6 | yes |
| | Auction | 9 | 9 | 10 | yes |
| Collaxa. com | StarLoan | 6 | 7 | 7 | yes |
| | Cauction | 5 | 7 | 6 | yes |

BACKUP

# Automata → Promela

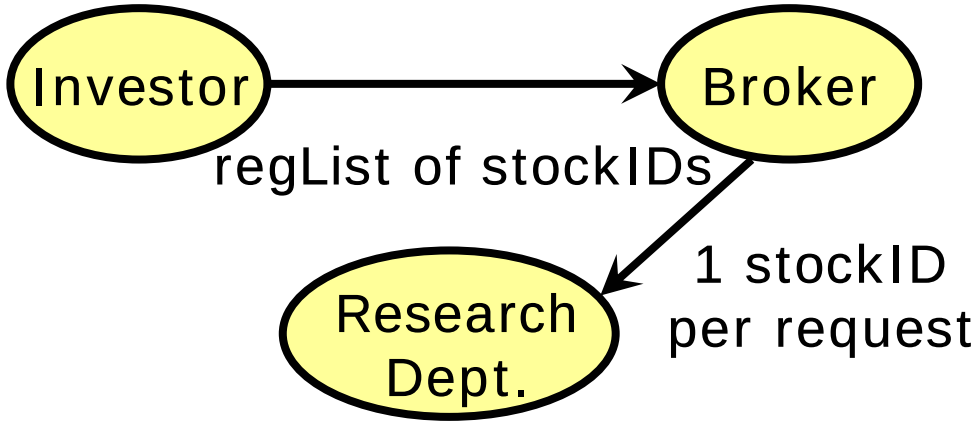
Bell Labs
Innovations for Lucent Technologies

- Translate to Promela (input lang. of SPIN)
  - 1 web service composition → 1 Promela specification
  - 1 Guarded Automaton → 1 Promela Process
  - Each Promela process has one Channel
  - Channels are bounded!
- Challenge: handling XML Data [ISSTA'04]
  - (bounded) XML Schema Type → Type system in Promela
  - Symbolic emulation of XPath expression
  - Special handling of position() and last() functions
    - Map each function to an integer variable

BACKUP

# Verification Experience

- Handling of XPath helps find intricate bugs!
  - Stock Analysis Service [Fu et. al. ISSTA'04]




```
Find stockID in last
    Request;
If last ID in regList
    exit;
Else
    send its subsequent
    stockID;
```

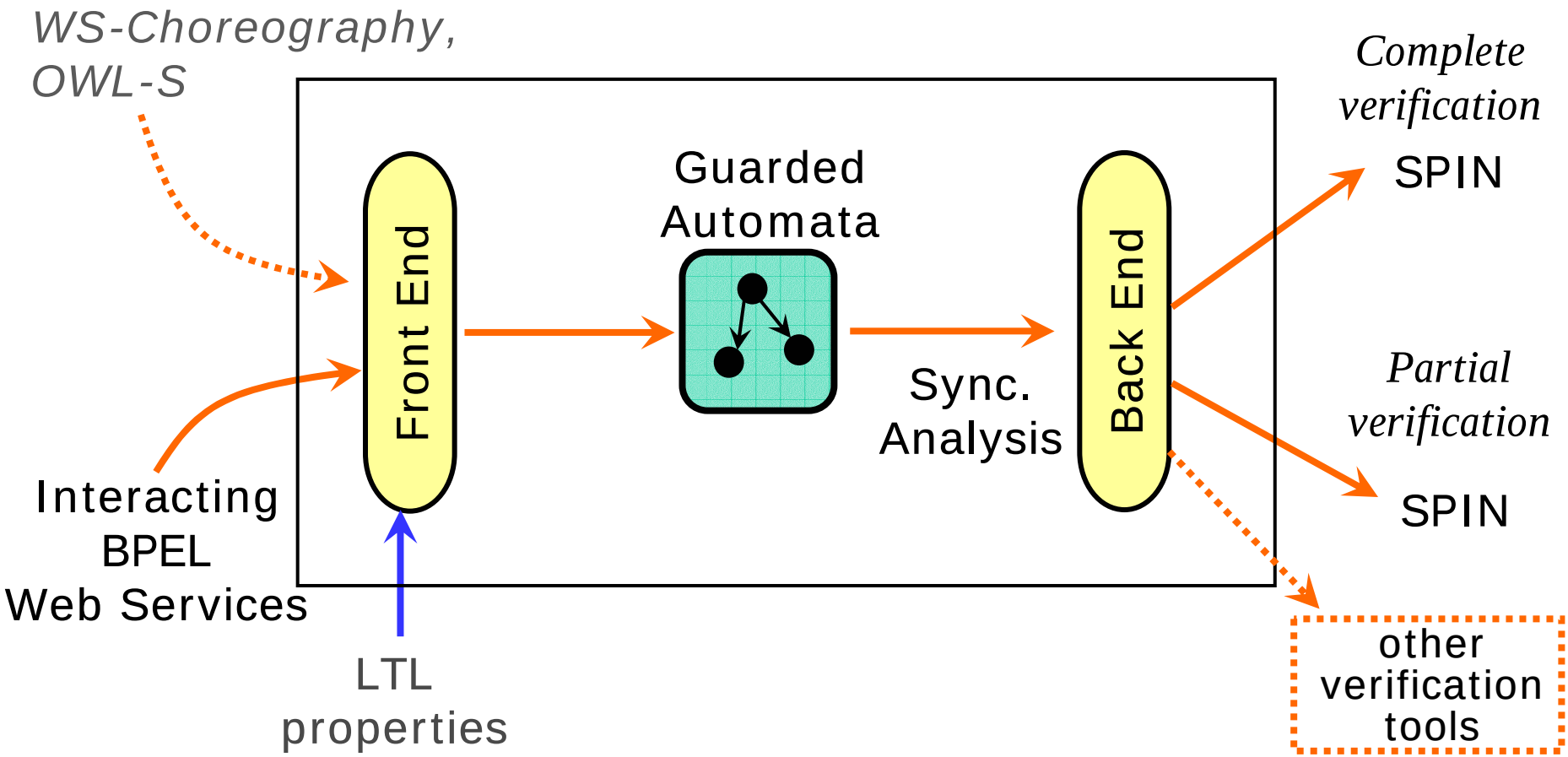
- Experience with other examples
  - Exhaustive search takes a lot of time
  - Performance satisfactory when finding errors
  - SPIN does not scale well with data domains
  - Use of Symbolic Verification may help

BACKUP

# Web Service Analysis Tool (WSAT)

[Santa Barbara group WWW'04, ISSTA'04, CAV'04]

- An aid for debugging web service compositions
  - Peer-to-peer or mediated



# Verification with Process Algebra

[Koshikina-van Breugel 2003]

- BPEL control structures → BPE-calculus
- BPE-calculus → PAC and then to CWB
- Concurrency Workbench [Cleaveland and Sims CAV'96]
  - Model checking tool for CCS and CSP

- Checking if a BPEL composition is deadlock-free
  - In general, temporal properties
- Message contents and local data contents are not modeled

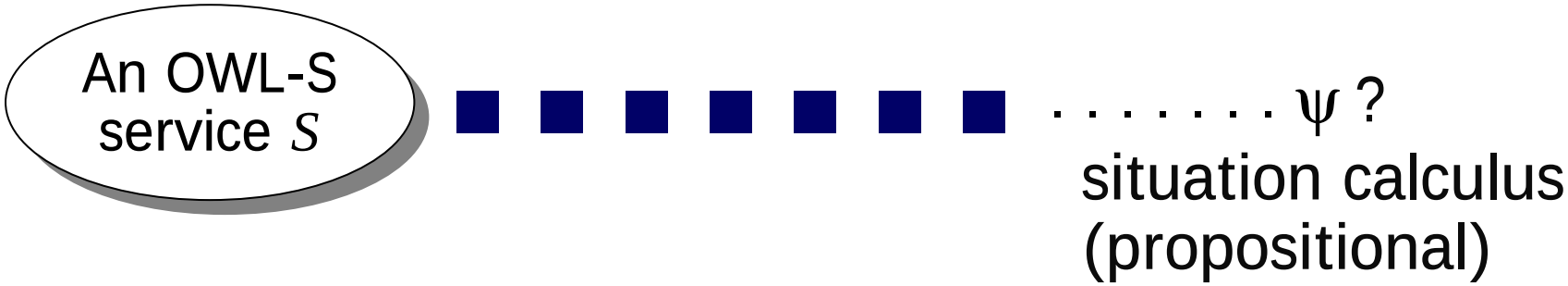
BACKUP

# Verification of OWL-S services

## [Narayanan-McIlraith WWW'02]

Analyzing and automated composition of OWL-S

- Analysis: Does an OWL-S service satisfy some property?

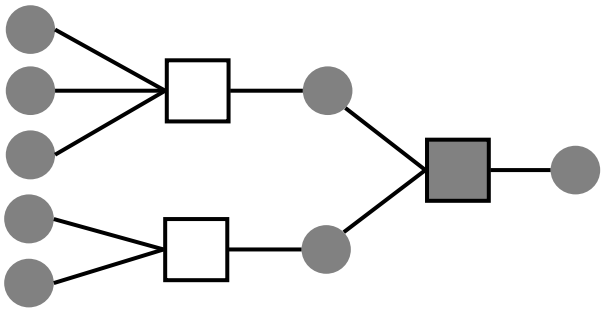



- Approach:
  - Simulating $\boldsymbol{S}$ using a Petri net
  - Conduct Petri net reachability analysis
- DAML-S (v0.5) analysis is PSPACE complete
  - Reachability of 1-safe nets (each place is marked 1 or less)

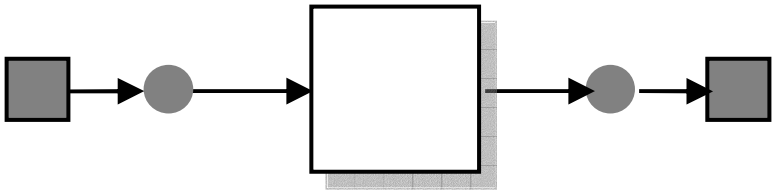
# Mapping OWL-S to Petri Net

Bell Labs
Innovations for Lucent Technologies

- Petri nets:
  - Places: hold tokens
  - Transitions: consume input tokens and produce output tokens
  - Marking: a snapshot
  - Reachability: one marking to another via transitions
- OWL-S to Petri net mapping:
  - Conditions (in situation calculus): places
  - Atomic services: transitions
    - Pre- and post-conditions



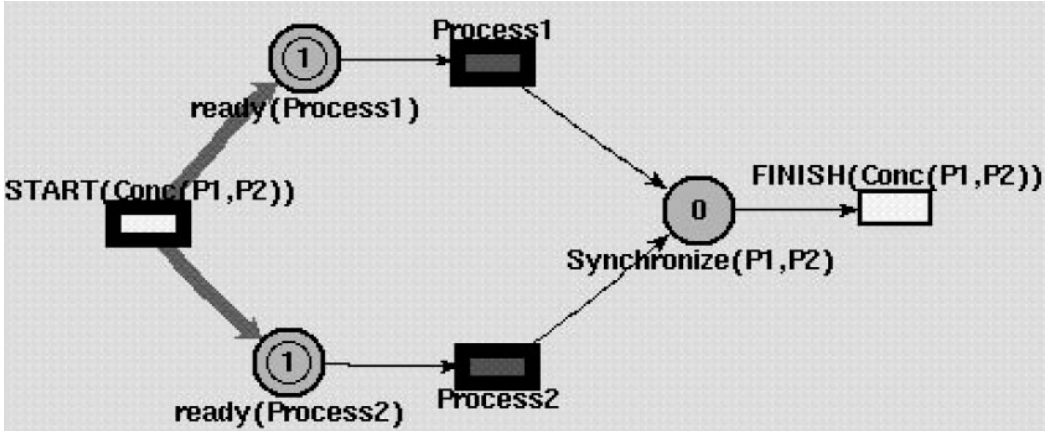
- Inductive mapping from services to Petri nets

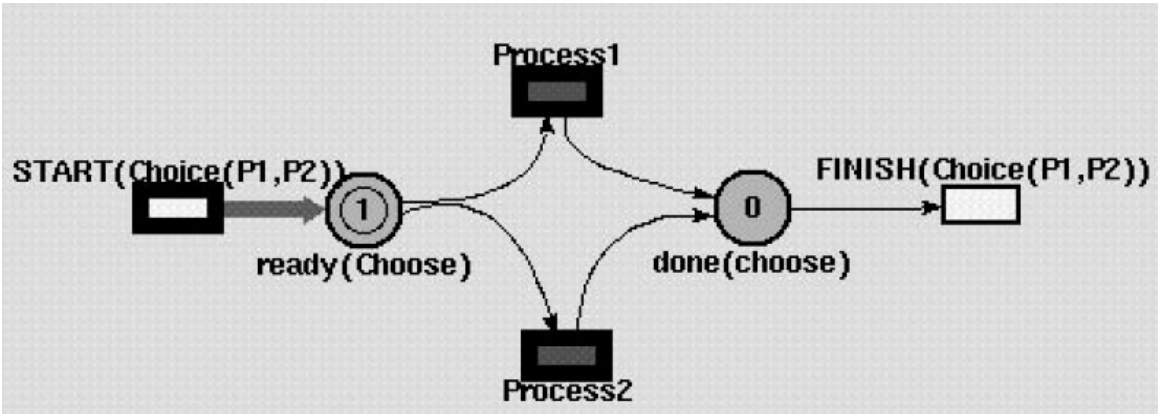


BACKUP

# OWL-S to Petri Net

- Control structures “glue” pieces together

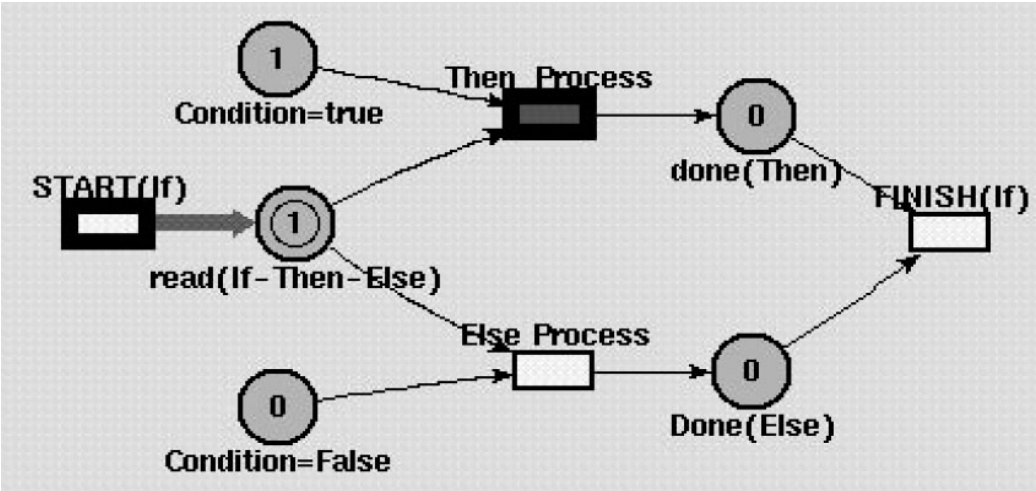


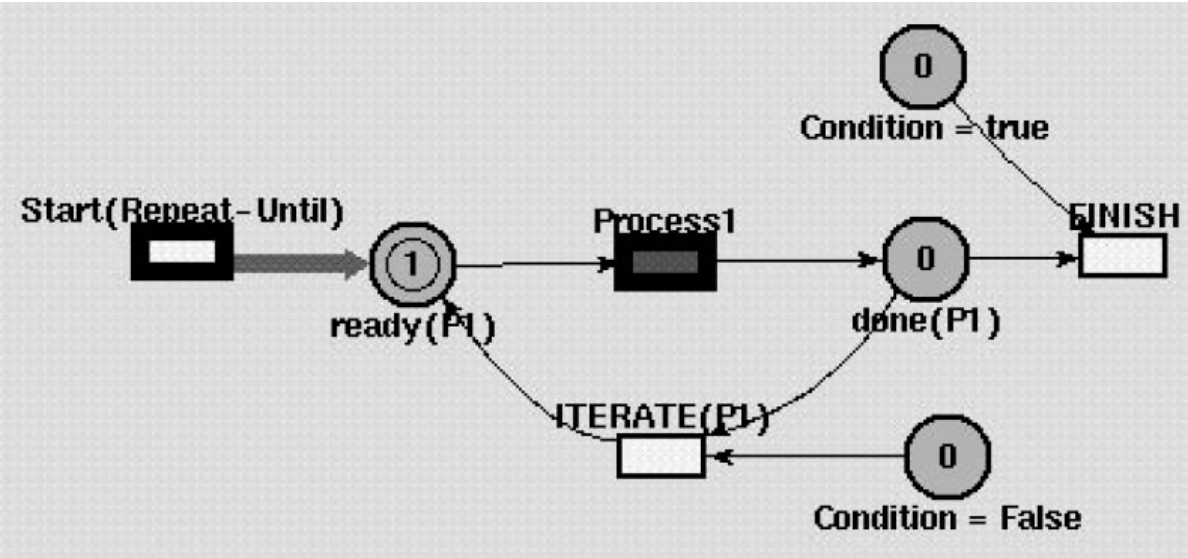




BACKUP

# If-Then-Else and Loops

Bell Labs
Innovations for Lucent Technologies



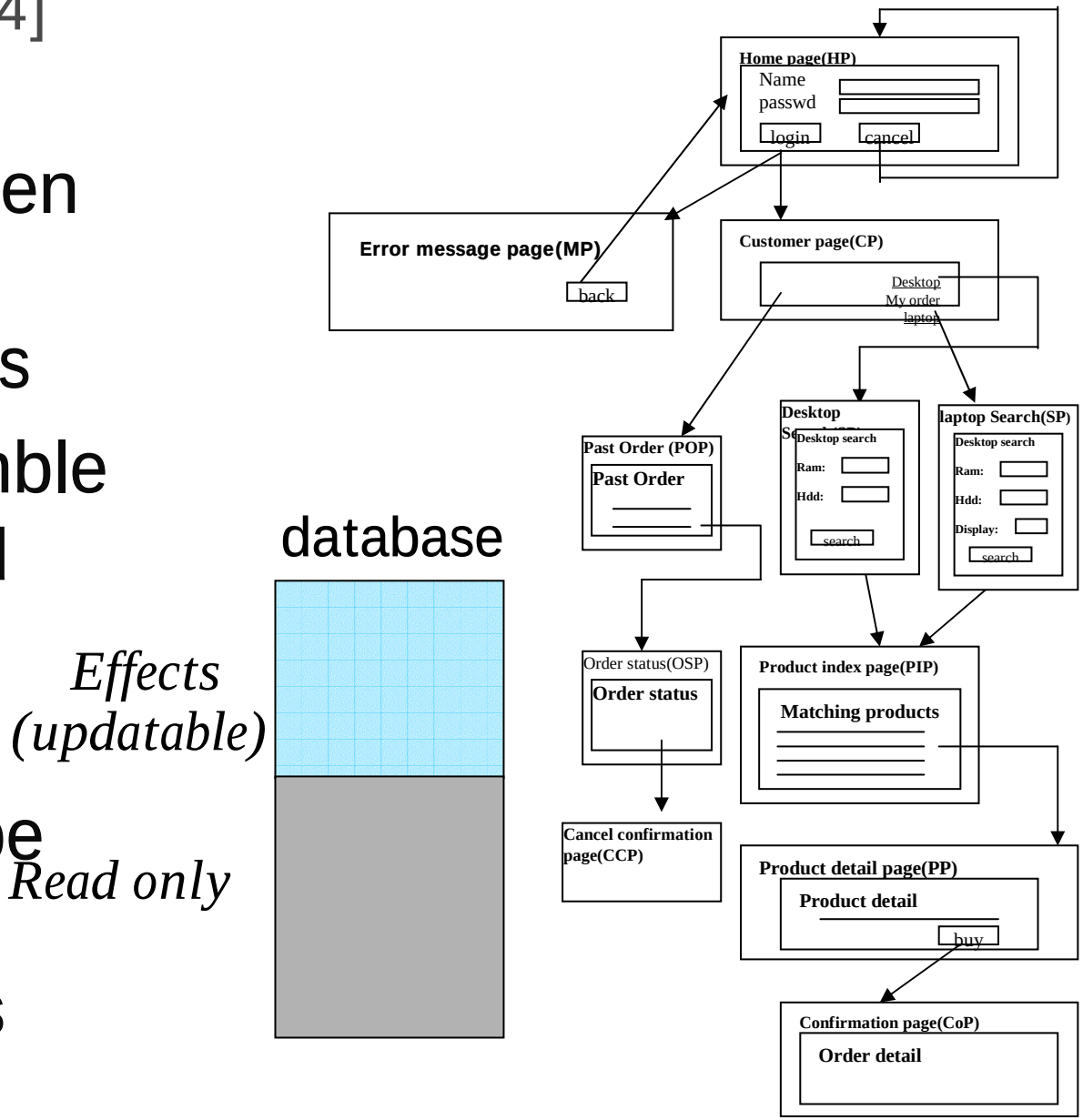




BACKUP

# Data Driven Web Service

[Deutsch et al PODS'04]

- Emphasis is on interaction between control flow and database contents
- Transitions resemble Datalog rules and update the database
- Rules + DB can be used to simulate control structures



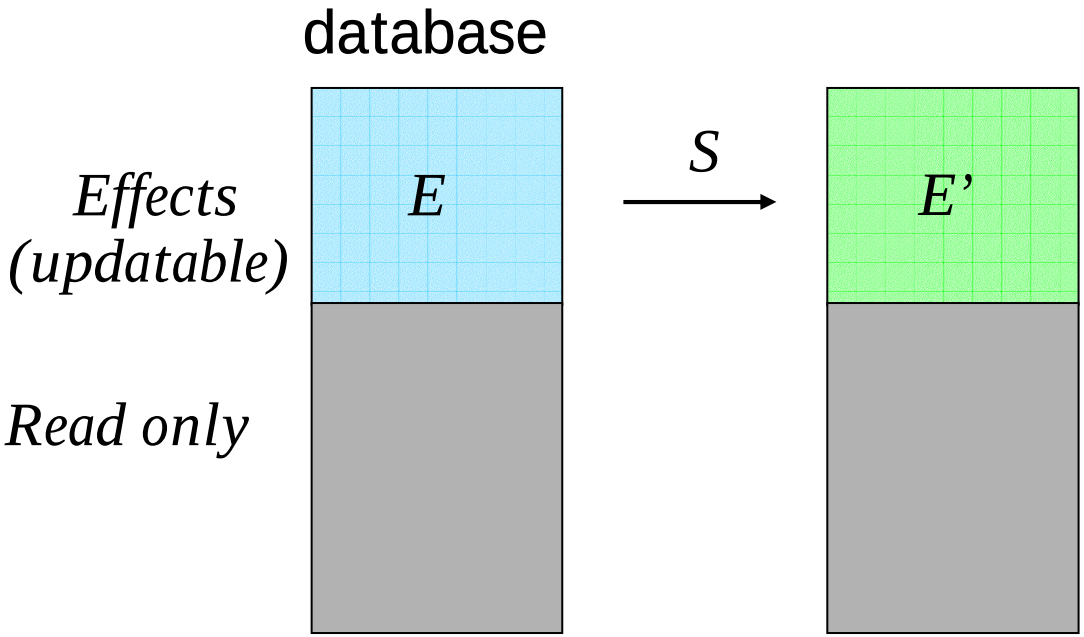
# An Abstract Perspective: Rule-based Control

- A hybrid combining automata and logic

Condition based on database query

Updates to the database

If $\psi$ then run $S$




A complex service:

If $\psi_1$ then run $S_1$

:

If $\psi_n$ then run $S_n$

# Verification of Rule Based Service

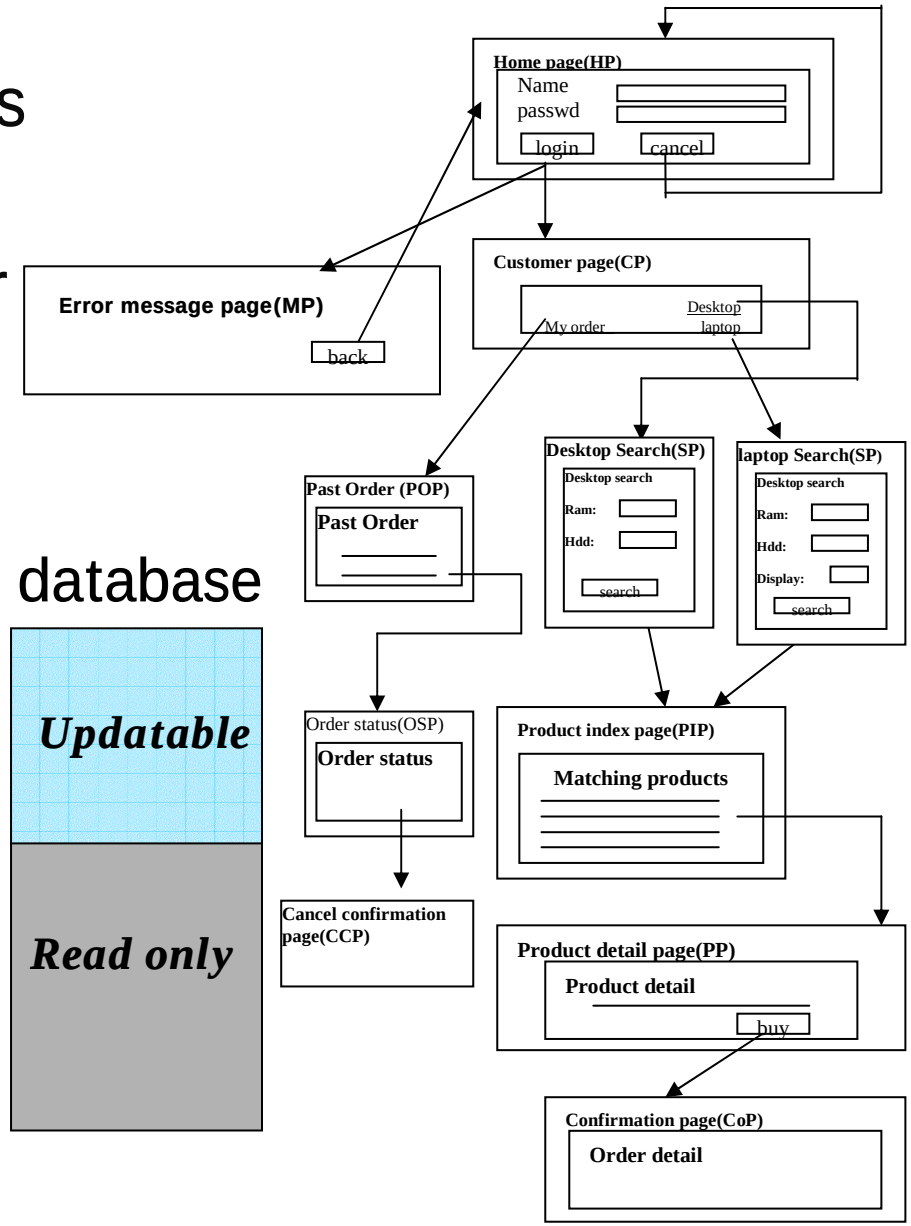
[Deutsch et al PODS'04]

First order temporal properties (FO-LTL)

- Verification is in PSPACE for a restricted model (input-bounded services)
- Undecidable with slight generalization
- Use and extend technique for abstract state machines [Spielman PODS'01]

Verification of CTL properties

- Decidable for the propositional case



# Outline

Bell Labs
Innovations for Lucent Technologies

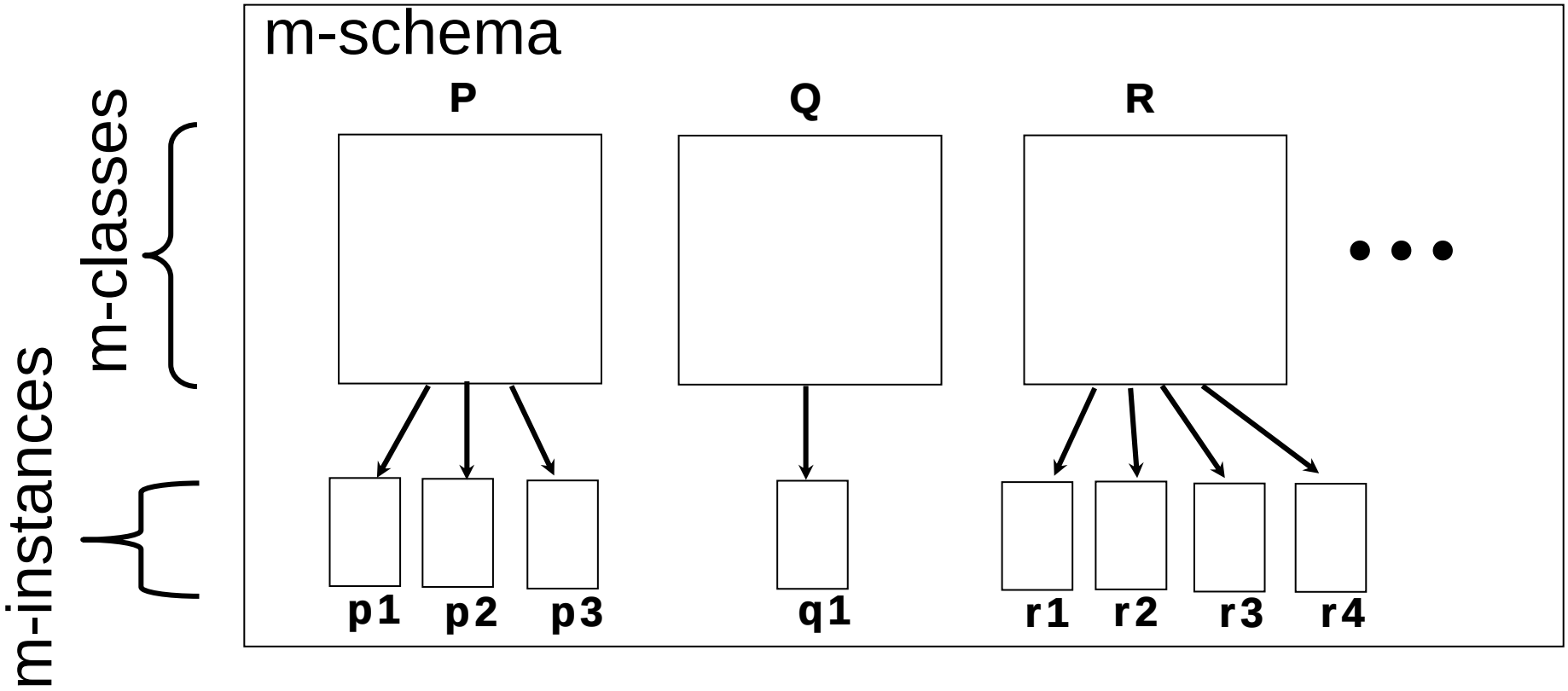
- Introduction
- Standards
- Perspectives on “Composition”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

# Telecom services: 2 Critical Differentiators

- Spawning of like processes, e.g.,
  - Adding participants to a teleconf
  - Re-connecting people that dropped
  - BPEL does not support internal spawning
- Graceful handling of “arbitrary” asynchronous events
  - People losing wireless connection
  - People becoming present
  - Note: BPEL requires scoping of points waiting for asynchronous events
- (Draft) “Minoan” model: combines OO, spawning, asynchonous events
- Preliminary result: Applies counter machines to verify properties of Minoan compositions

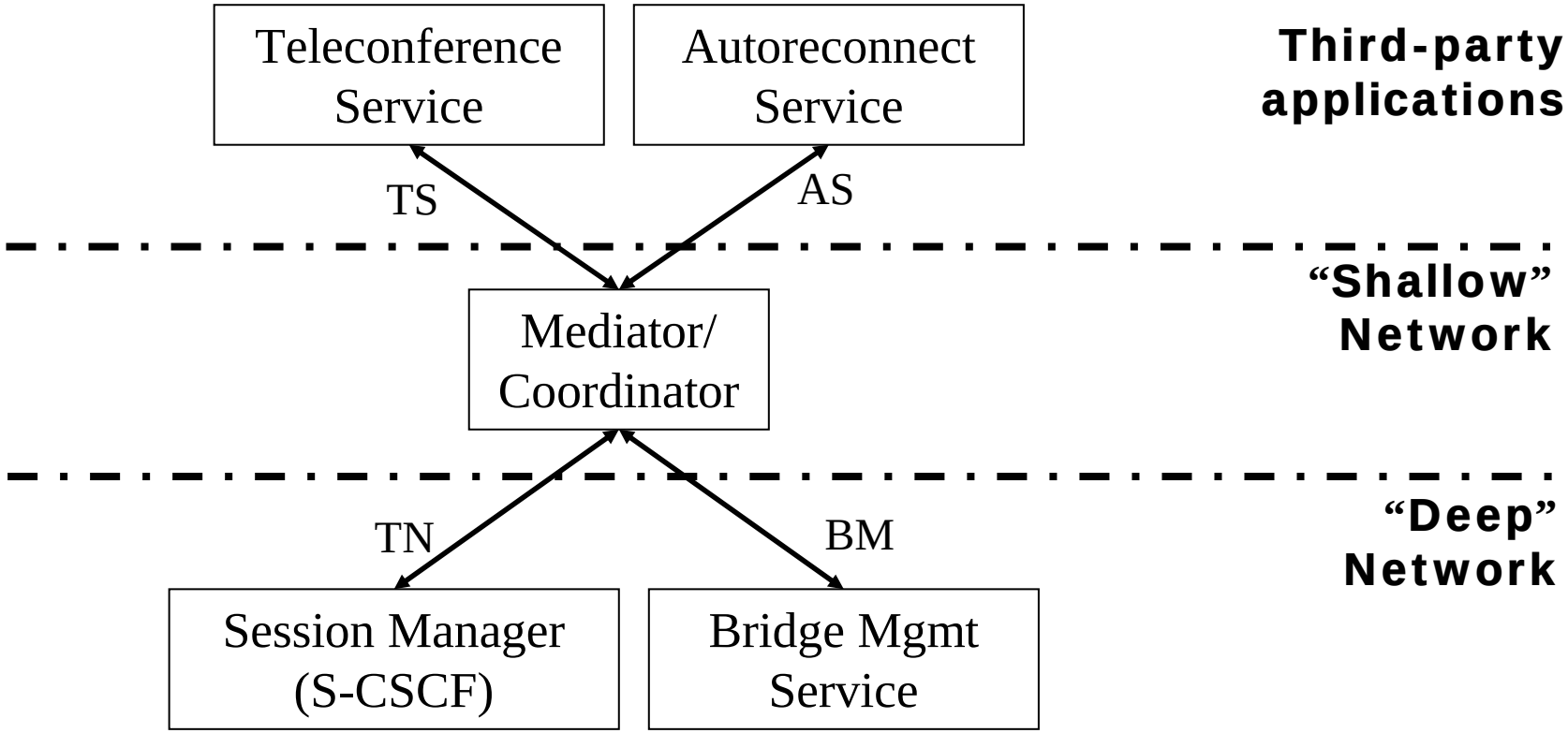
# (Draft) Minoan Model

[Benedikt, Christophides, Hull, Karvounarakis, Kumar, Tannen, Su]




- M-classes are analogous to objects from OOPLS
- Internally: guarded automata
- Messages go to existing process or spawn new one
- Various approaches to local data stores

# IMS-inspired example

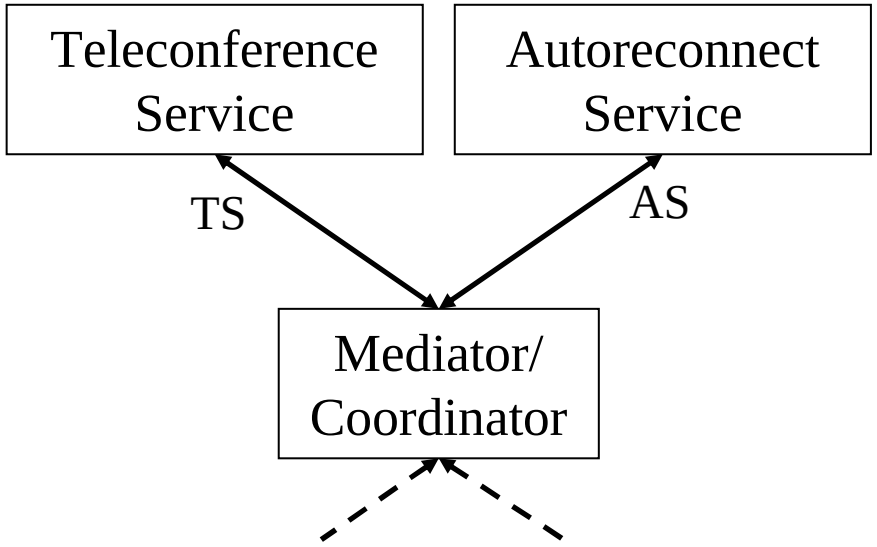



Two applications, written independently

- TS: Sets up, maintains teleconferences
- AS: Automatic reconnect for wireless users that drop

Mediator/Coordinator: Gives TS and AS illusion of working independently

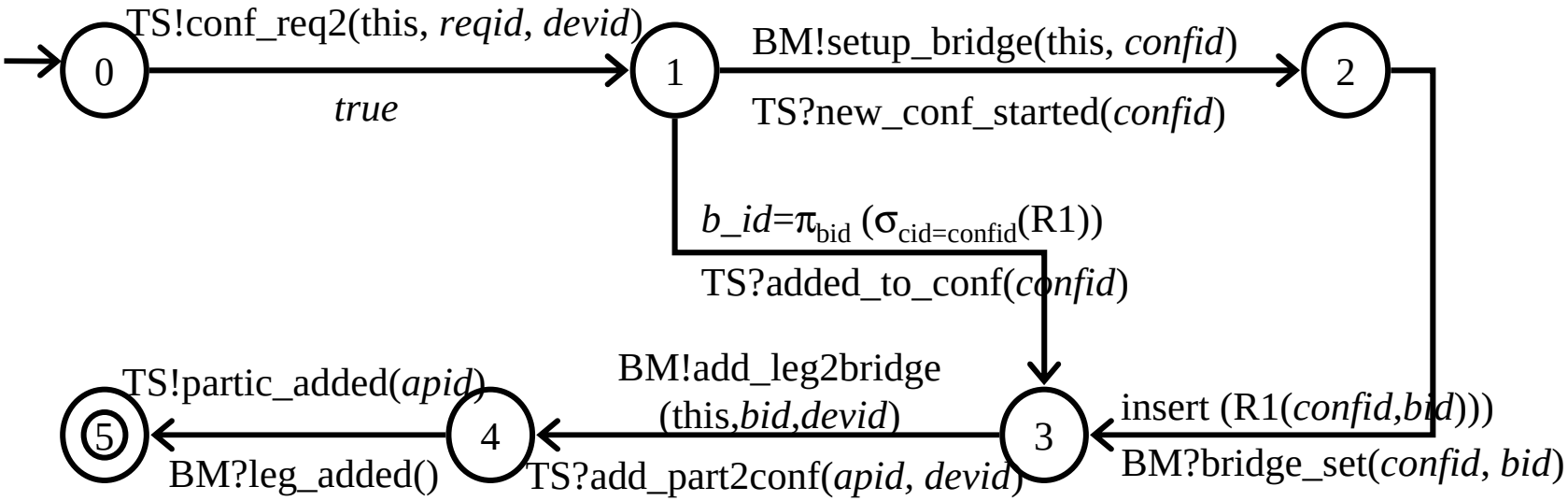
# Inside the services and mediator




- All three modules have a “root” process
- Additional M-classes in Mediator
  - Forward request to participate in teleconf to TS
    - If subscriber has paid for AS, also forward info to AS
    - Handle TS requests (e.g., telling BM to add someone to conf)
  - Handle situation where user drops connection
    - Forward to AS, and deal with AS response
- Additional M-classes in TS:
  - Manage teleconf, manage participant in teleconf

# Representative M-class of Mediator

Mediator.Conference_Request (params: reqid, devid, isprem) (1 per request)




<u>Local vars:</u> *reqid*, *devid*, *confid*, *bid*, *apid*, *aid*: ids ; *is_prem*: bool

- Even in a single teleconf, there may be multiple M-instances for this M-class
- Unbounded spawning ⇒ verification undecidable
- As an abstraction, we will view the "tokens" moving around this FSM as uniform

# A Basic Question, answered via counters

- "Bounded Stability" : For each possible state of the system, if a finite number of events come in from the deep network, will processing eventually terminate?
- Towards sufficient condition (and algorithm) for bounded stability
  - Ignore data stores and message contents
  - Assume one message type per queue
  - For each M-class, for each state of the M-class, keep a count of number of M-instances that are in that state
  - Also count the number messages queued
- "Stability Ordering" for the counters: one such that each valid move leads to decrease of counter values, according to lexigraphical ordering

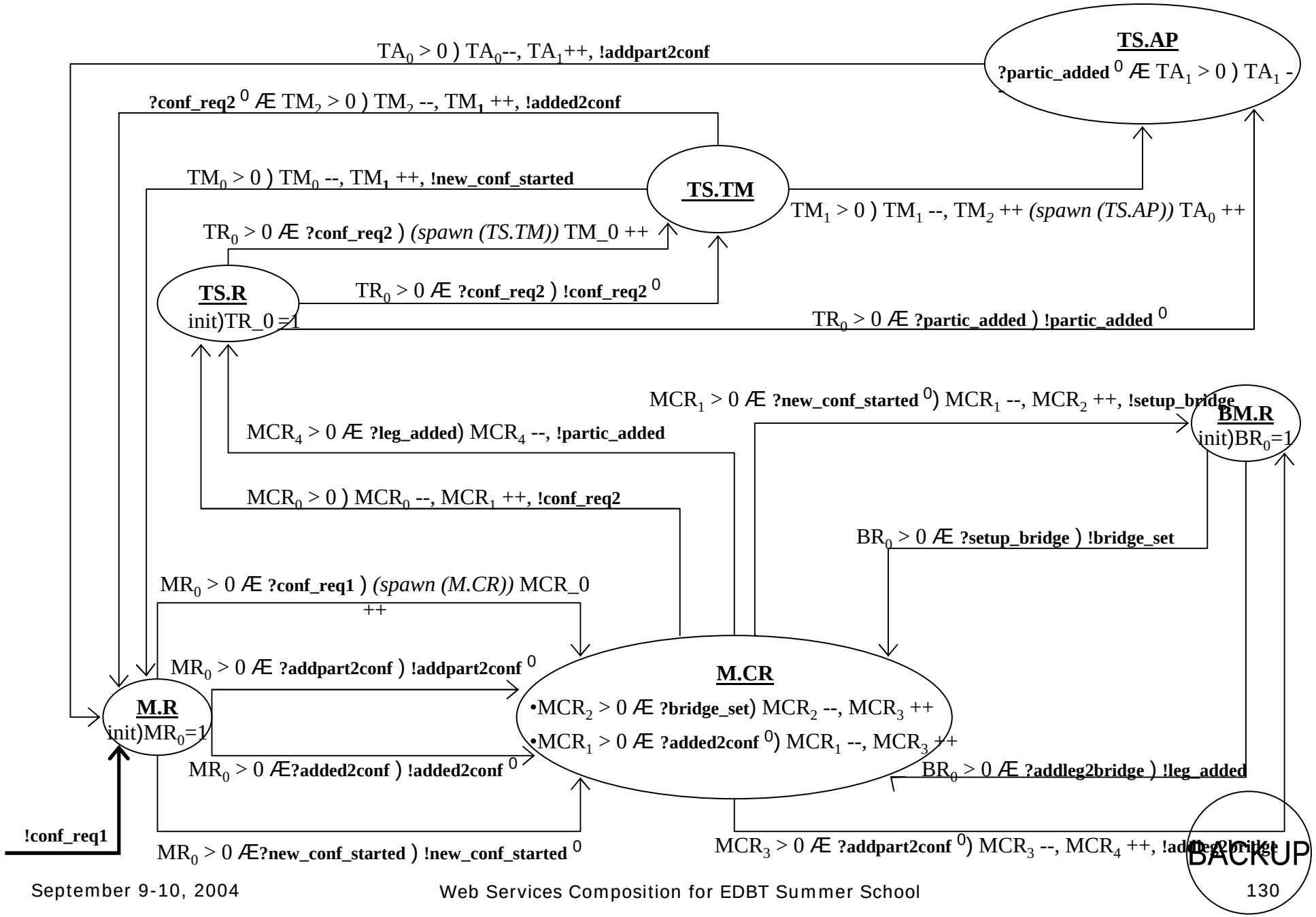
# Results so far

Bell Labs
Innovations for Lucent Technologies

- Thm: If there is a stability ordering, then the system has bounded stability
- Algorithm: searches for stability ordering, by adding constraints based on set of M-classes
  - If no cycle found, then abstract and concrete systems have bounded stability
  - If cycle found, then the abstract system does not bounded stability

- We see this as a first step towards (approximate) solution of general properties of Minoan systems

# Part of system view, with queues and counters for M-class states



# Abstract view of (part of) system, entirely counter-based

init)TR_0 =1
init)MR_0=1
init)MCR_$0_2$=1

$MR_0$ > 0 Æ **conf_req1** > 0 ) **conf_req1--,** *(spawn (M.CR))* $MCR_0$ ++

$MR_0$ > 0 Æ **new_conf_started** > 0 ) **new_conf_started --, new_conf_started** $^0$ ++

$MR_0$ > 0 Æ **addpart2conf** > 0 ) **addpart2conf -- , addpart2conf** $^0$++

$MR_0$ > 0 Æ **added2conf** > 0 ) **added2conf--, added2conf** $^0$ ++

$MCR_0$ > 0 ) $MCR_0$ --, $MCR_1$ ++, **conf_req2** ++

$MCR_1$ > 0 Æ **new_conf_started** $^0$ > 0 ) **new_conf_started** $^0$ --, $MCR_1$ --, $MCR_2$ ++, **setup_bridge++**

$MCR_1$ > 0 Æ **added2conf** $^0$ > 0 ) **added2conf** $^0$--, $MCR_1$ --, $MCR_3$ ++

$MCR_2$ > 0 Æ **bridge_set** > 0 ) **bridge_set --,** $MCR_2$ --, $MCR_3$ ++

$MCR_3$ > 0 Æ **addpart2conf** $^0$ > 0 ) **addpart2conf** $^0$ --, $MCR_3$ --, $MCR_4$ ++, **addleg2bridge**++

$MCR_4$ > 0 Æ **leg_added** > 0 ) $MCR_4$ --, **leg_added --, partic_added**++

$TR_0$ > 0 Æ **conf_req2** > 0 ) *(spawn (TS.TM))* **conf_req2 --,** $TM_0$ ++

$TR_0$ > 0 Æ **conf_req2** > 0 ) **conf_req2 --, conf_req2** $^0$ **++**

$TR_0$ > 0 Æ **partic_added** > 0 ) **partic_added --, partic_added** $^0$ ++

$TM_0$ > 0 ) $TM_0$ --, $TM_1$ ++, **new_conf_started++**

$TM_1$ > 0 ) $TM_1$ --, $TM_2$ ++ *(spawn (TS.AP))* $TA_0$ ++

$TM_2$ > 0 Æ **conf_req2** $^0$ >0) $TM_2$ --, $TM_1$ ++, **conf_req2**$^0$ **--, added2conf**++

$TA_0$ > 0 ) $TA_0$--, $TA_1$++, **addpart2conf++**

$TA_1$ > 0 Æ **partic_added** $^0$ > 0 ) **partic_added** $^0$ --, $TA_1$ --

$BR_0$ > 0 Æ **setup_bridge** > 0 ) **setup_bridge --, bridge_set** ++

$BR_0$ > 0 Æ **addleg2bridge** > 0 ) **addleg2bridge --, leg_added**++

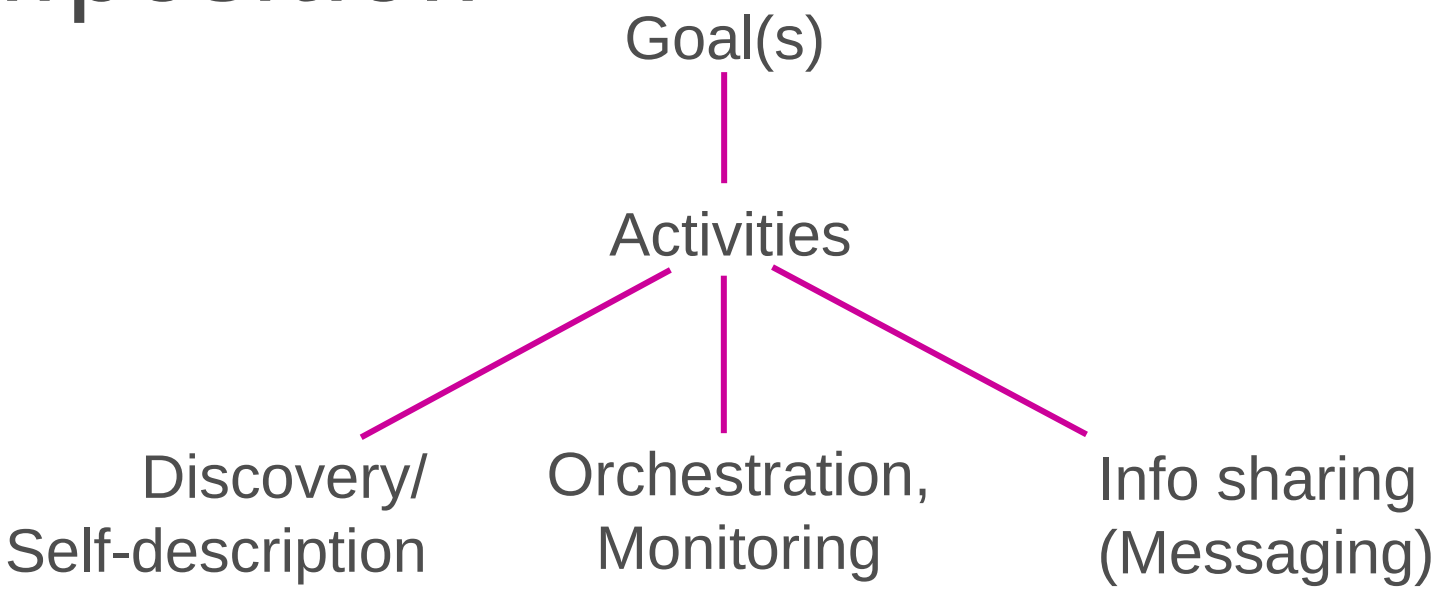
BACKUP

# Outline

Bell Labs
Innovations for Lucent Technologies

- Introduction
- Standards
- Perspectives on “Composition”
- Analysis and Verification
  - A coordination model for telecom
- A Glimpse of the Future

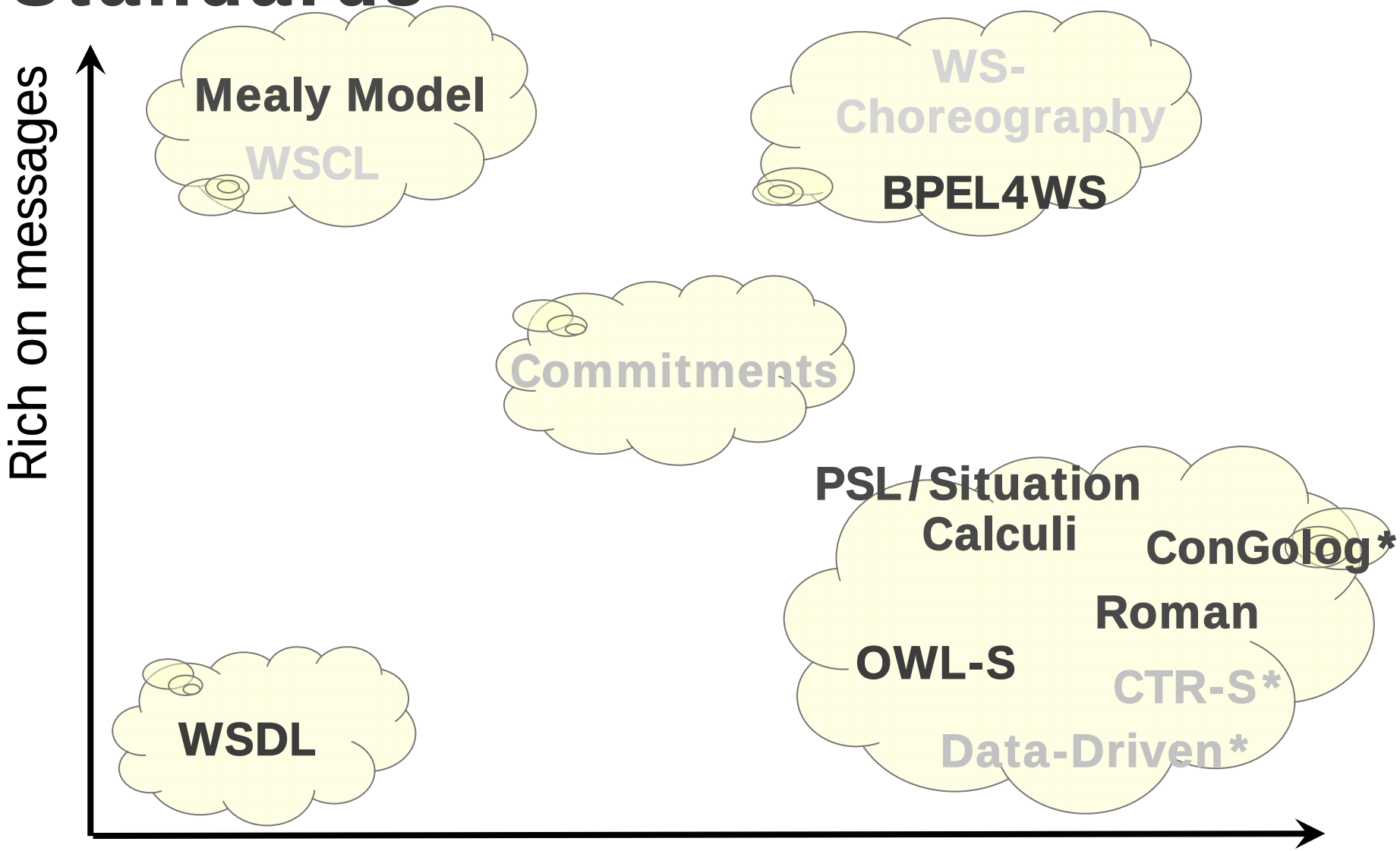
# Anatomy of Web Services Composition




- No unified model:
  - BPEL, Mealy: Strong on orchestration, info sharing
  - OWL-S: Strong on goals, activities, discovery
  - “Roman” model: Strong on activities, orchestration

# Composition **Models** and **Standards**

Bell Labs
Innovations for Lucent Technologies




***if interpreted as a composition model**

# Elements of a unified model

[based on recent discussions with D. Berardi, D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Marcello]

| Crucial Feature | Possible starting point ? |
| --- | --- |
| • Usable model of the "world" | • Fluents, as in PSL or sit. calc. |
| • Notion of atomic actions that examine/impact the world | • OWL-S atomic services |
| • Notion of web service that wraps<br>– Atomic actions<br>– Message send/receive<br>– Process/Enactments | • Many choices . . . |
| • Will probably need a notion of "semantic behavior signature" to describe services | • Guarded automata (generalization of Mealy) |
| • Message-based communication between services | • Use lessons from Mealy machines |
| • Automatic composition | • Generalize Roman approach? Maybe high complexity or undecidable; issue of knowledge gained during execution |
| • Spawning | • Leave for another day… |

# Additional concern: The structure of processes and enactments

- Typical to conceptualize a process spec, and a process instance for each request
  - Cf. object-oriented style
  - Tendency to build models, reasoning on “per enactment” basis
- But one order might have multiple sub-requests
  - E.g., one order might involve processing each item separately
  - Possible argument for “unbounded” spawning of processes
- Sub-request activities might be grouped
  - E.g., parts of several book orders may be combined into one shipment
  - Reasoning on “per enactment” basis will not be sufficient
- Telecom and “always-on” services
  - Enactments may have unbounded duration
  - Process spawning appears useful for specifying behaviors

# Reasons for Optimism

Bell Labs
Innovations for Lucent Technologies

- Broad and growing interest, across several disciplines
  - AI
  - Services oriented Computing
  - Databases
  - Verification, Logic in Computer Science
- Consortia have formed
  - Semantic Web Services Initiative (SWSI)
    - Re-thinking OWL-S, e.g., to overcome limitations re variable passing
  - Web Services Model Framework (WSMF)
- Government research funding
  - Substantial in Europe
  - US: DARPA's DAML program was significant beginning; follow-on may be created
- Substantial groundwork has been laid