

Service Composition

*Technologies, Methods and Tools for Synthesis and Orchestration
of Composite Services and Processes*

**Giuseppe De Giacomo
Massimo Mecella**

*Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza"*

`{degiacomo,mecella}@dis.uniroma1.it`

based on joint work with Daniela Berardi,
Diego Calvanese and Maurizio Lenzerini

Instructors

(i.e., the Odd Couple)

- Giuseppe De Giacomo
(<http://www.dis.uniroma1.it/~degiacom>)
 - Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica
 - Research Interests:
 - AI: Knowledge Representation, Reasoning about Action, Planning, Cognitive Robotics
 - Databases: Databases with Incomplete Information, Data Integration, Semantic Interoperability
 - Dynamic Systems: Reasoning on Dynamic Systems, Process Verification and Synthesis
 - Services: Composition
- Massimo Mecella
(<http://www.dis.uniroma1.it/~mecella>)
 - Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica
 - Research Interests:
 - Software Architectures: Cooperative Information Systems for e-Government and e-Business
 - Information Systems: Workflow Management, Data Quality
 - Distributed Systems: Middleware and Mobile Applications
 - Services: Modeling, Compatibility and Substitution, Composition and Orchestration



Outline

- Part 1 (Massimo Mecella)
 - Basic Concepts
 - Relevant Technologies and Abstractions
 - State of the Art of the Research in Composition (Synthesis and Orchestration)
- Part 2 (Giuseppe De Giacomo)
 - Classification of the Different Approaches to Composition
 - Formal Tools for Composition
 - Discussion and Conclusions

Disclaimer and Copyright Notice: Permission to make digital or hard copies of part or all of this tutorial for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. Some of the figures presented in this tutorial are freely inspired by others reported in referenced works/sources. For such figures copyright and all rights therein are maintained by the original authors or by other copyright holders (Springer Verlag, ACM, IEEE, etc.). It is understood that all persons copying these figures will adhere to the terms and constraints invoked by each copyright holder.

Basic Concepts

Basic Concepts: Outline

- Defining e-Services, Web Services, Services
- Services as the Evolution of Middleware and Enterprise Application Integration
- Setting-up the Stage
 - Conversation
 - Choreography
 - Composition
 - Synthesis
 - Orchestration

e-Services, Web Services, Services ... (1)

- An e-Service is often defined as an **application accessible via the Web**, that provides a set of functionalities to businesses or individuals. What makes the e-Service vision attractive is the ability to automatically discover the e-Services that fulfill the users' needs, negotiate service contracts, and have the services delivered where and when users needs them

Guest editorial. In [VLDBJ01]

- e-Service: **an application component** provided by an organization in order to be assembled and reused in a distributed, Internet-based environment; an application component is considered as an e-Service if it is: (i) **open**, that is independent, as much as possible, of specific platforms and computing paradigms; (ii) **developed mainly for inter-organizations applications**, not only for intra-organization applications; (iii) **easily composable**; its assembling and integration in an inter-organizations application does not require the development of complex adapters.
e-Application: a distributed application which integrates in a cooperative way the e-Services offered by different organizations

M. Mecella, B. Pernici: Designing Wrapper Components for e-Services in Integrating Heterogeneous Systems. In [VLDBJ01]

e-Services, Web Services, Services ... (2)

A Web service is a **software system** identified by a URI, whose **public interfaces** and bindings are defined and described using XML. Its definition can be discovered by **other software systems**. These systems may then **interact with** the Web service in a manner prescribed by its definition, using XML based **messages** conveyed by Internet protocols

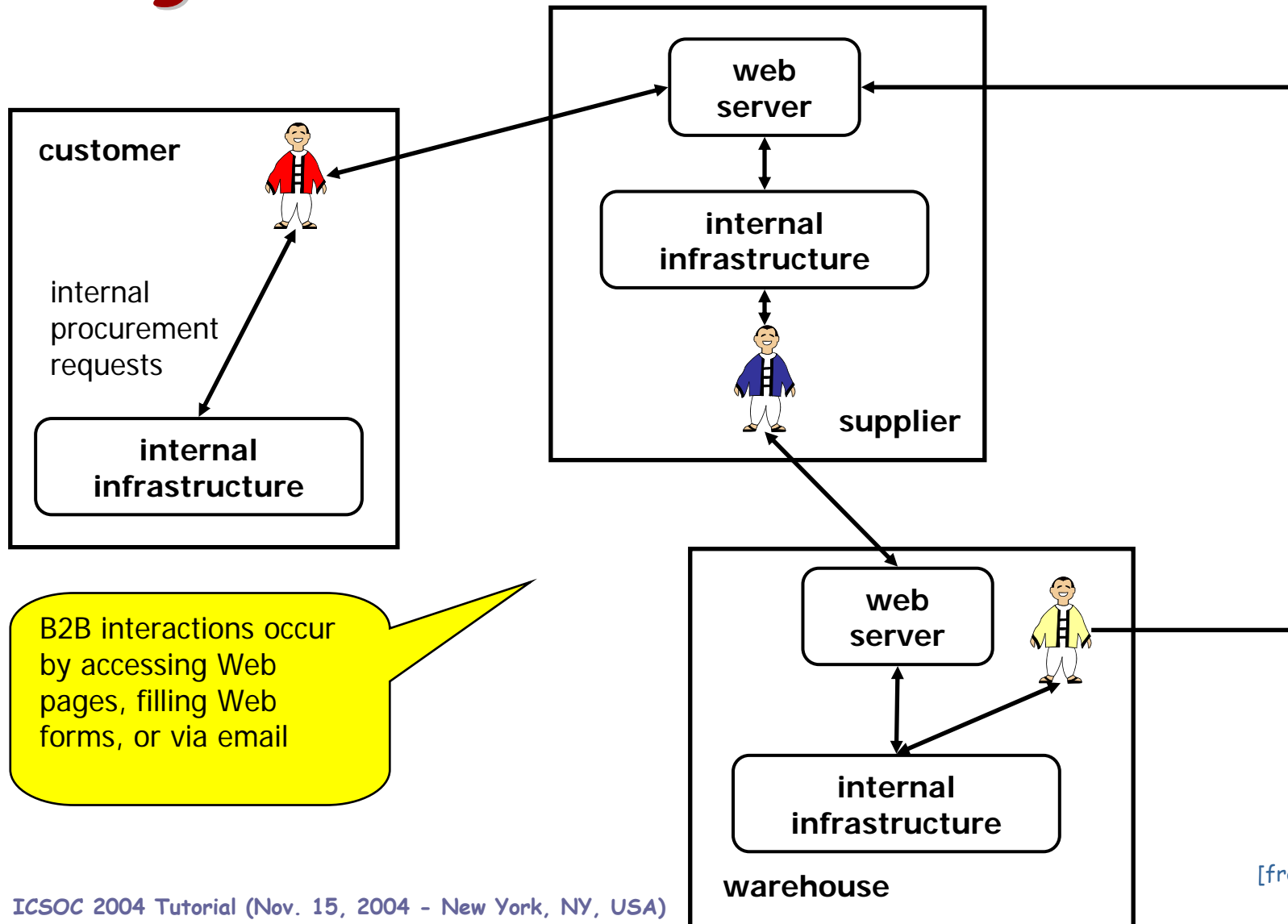
*Web Services Architecture Requirements,
W3C Working Group Note, 11 Feb. 2004,
<http://www.w3.org/TR/wsa-reqs/>*

e-Services, Web Services, Services ... (3)

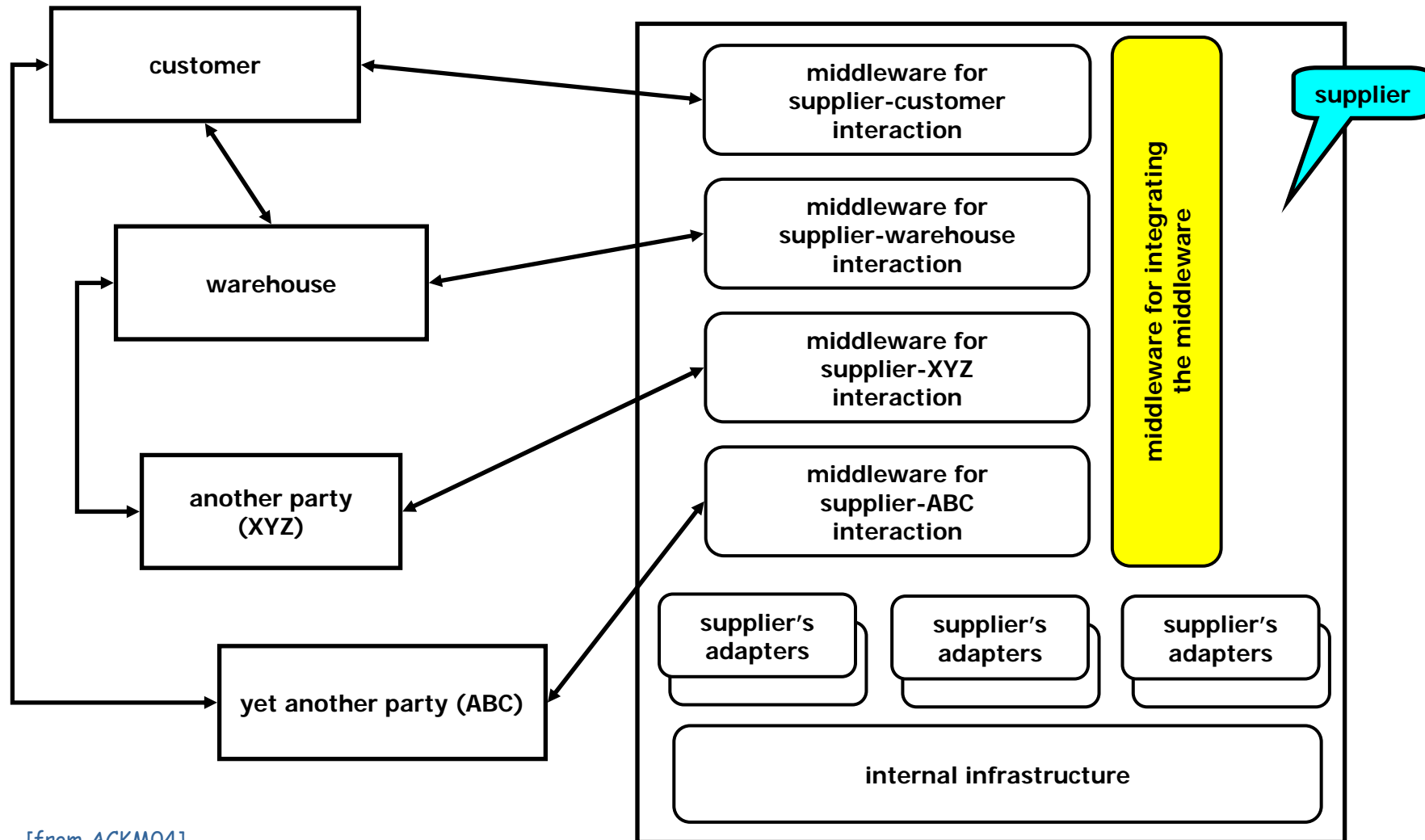
- **Services are self-describing, open components that support rapid, low-cost composition of distributed applications.** Services are offered by service providers — organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. Since services may be offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure for both intra and cross-enterprise application integration and collaboration. Service descriptions are used to advertise the service capabilities, interface, behavior, and quality. Publication of such information about available services provides the necessary means for discovery, selection, binding, and composition of services. In particular, the service capability description states the conceptual purpose and expected results of the service (by using terms or concepts defined in an application-specific taxonomy). The service interface description publishes the service signature (its input/output/error parameters and message types). The (expected) behavior of a service during its execution is described by its service behavior description. Finally, the Quality of Service (QoS) description publishes important functional and nonfunctional service quality attributes [...]. Service clients (end-user organizations that use some service) and service aggregators (organizations that consolidate multiple services into a new, single service offering) utilize service descriptions to achieve their objectives.
- **The application on the Web (including several aspects of the SOA) is manifested by Web services**

Guest editorial. In [CACMO3]

(naive) Business-to-Business Integration



WSs: the Evolution of Middleware and EAI Technologies (1)



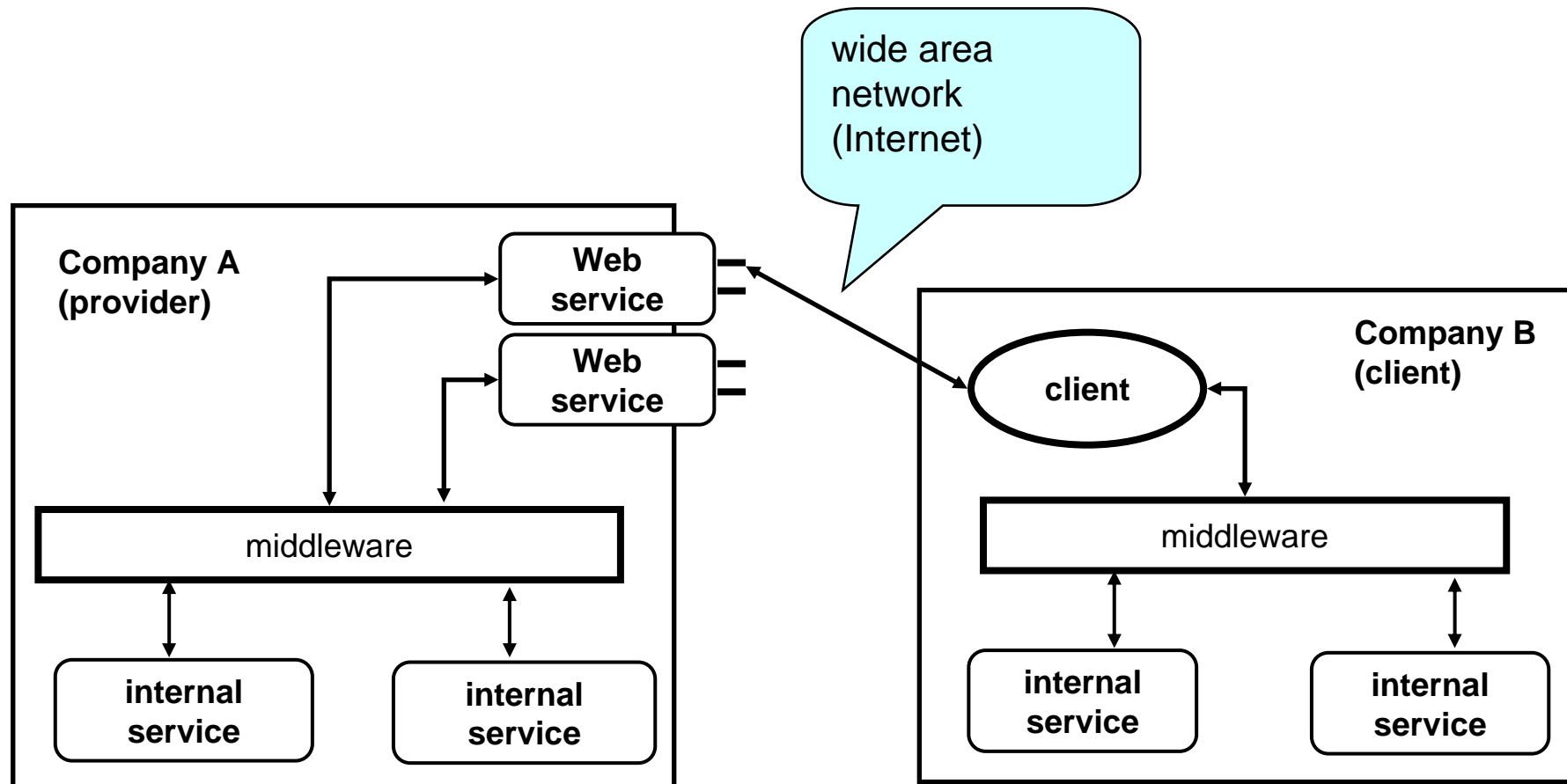
[from ACKM04]

ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Giuseppe De Giacomo & Massimo Mecella

10

WSs: the Evolution of Middleware and EAI Technologies (2)



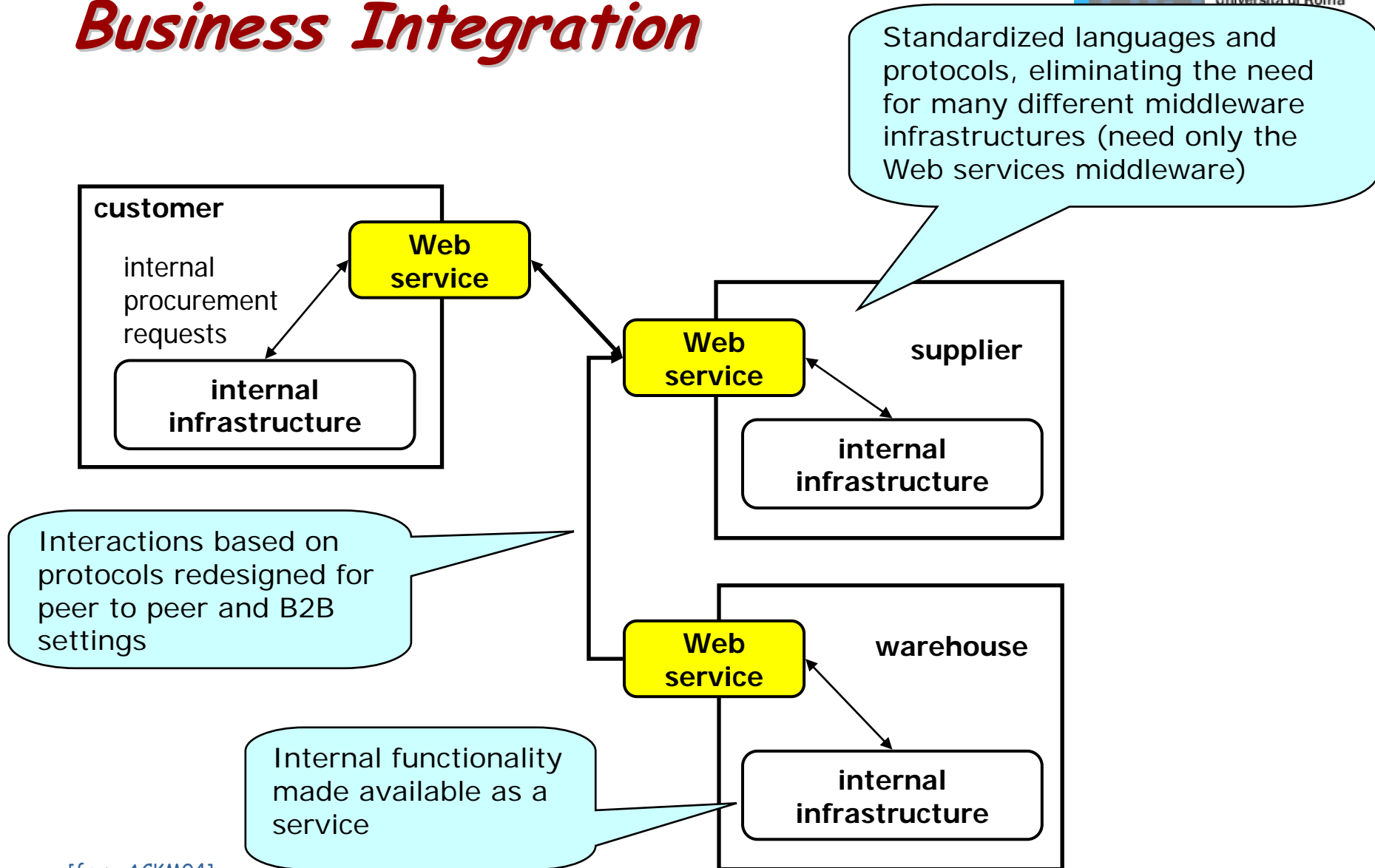
[from ACKM04]

ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Giuseppe De Giacomo & Massimo Mecella

11

(WS-based) Business-to-Business Integration



[from ACKM04]

ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Giuseppe De Giacomo & Massimo Mecella

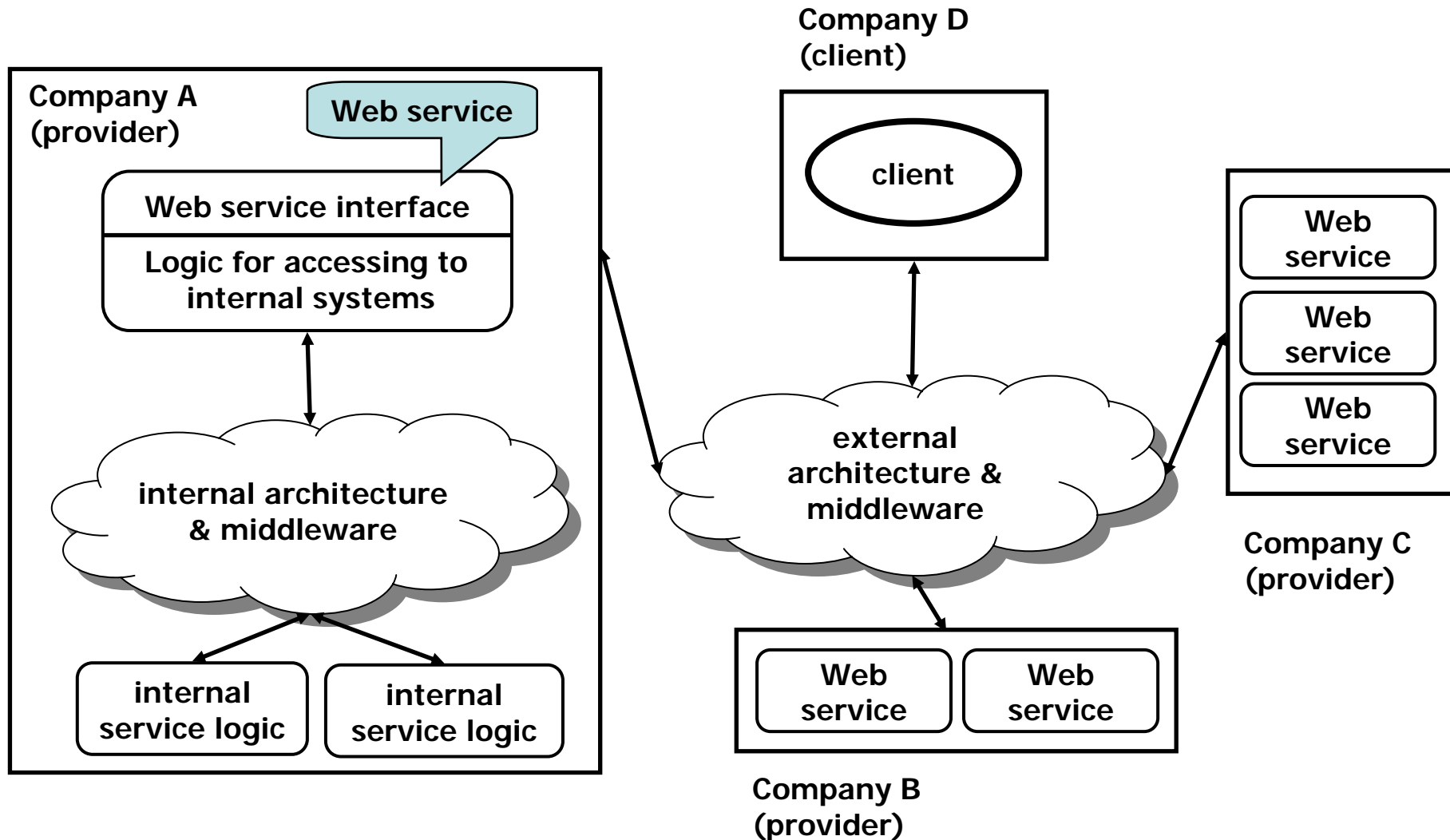
12

When Web Services Should Be Applied ?

- When it is not possible to easily manage deployment so that all requesters and providers are upgraded at once
- When components of the distributed system run on different platforms and vendor products
- When an existing application needs to be exposed over a network for use by unknown requesters

*Web Services Architecture,
W3C Working Group Note, 11 Feb. 2004,
<http://www.w3.org/TR/ws-arch/>*

Two Architectures (and Middlewares) (1)



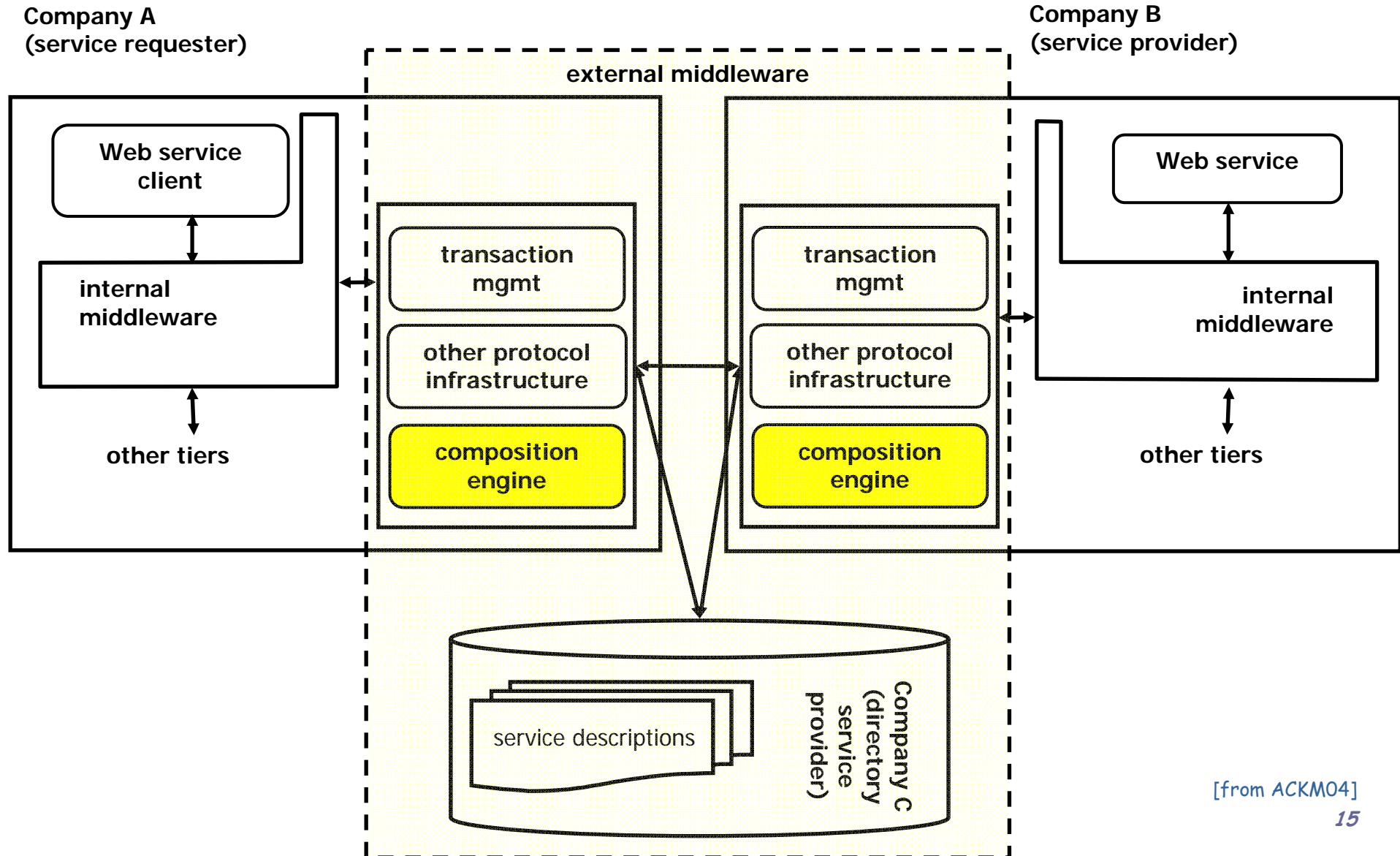
[from ACKM04]

ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Giuseppe De Giacomo & Massimo Mecella

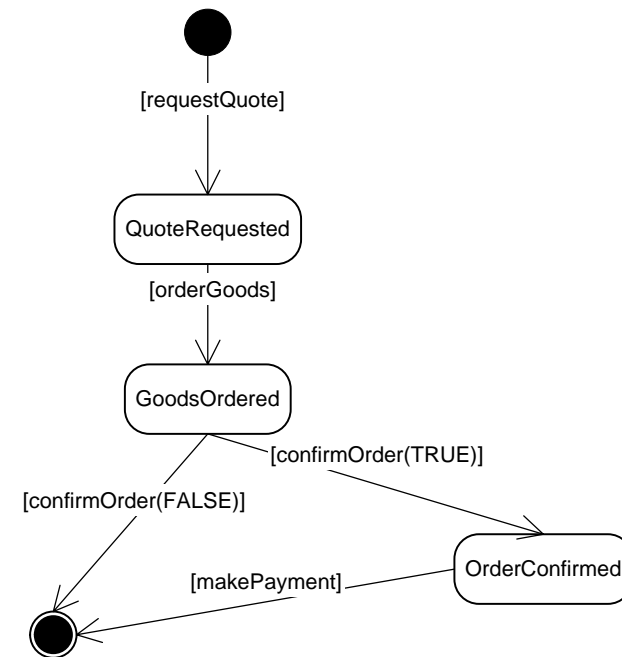
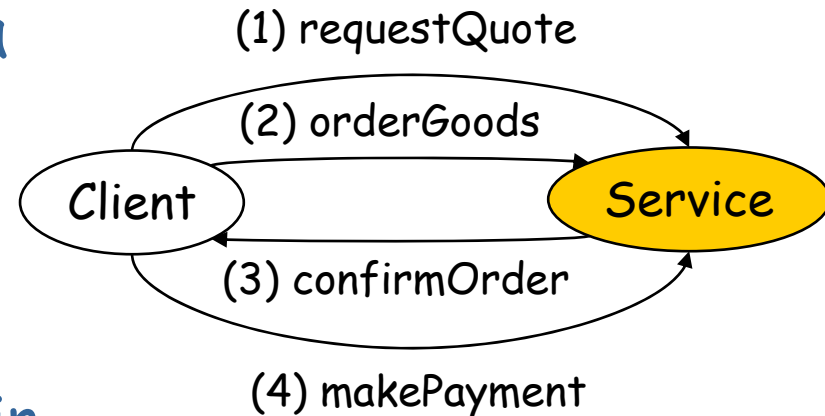
14

Two Architectures (and Middlewares) (2)



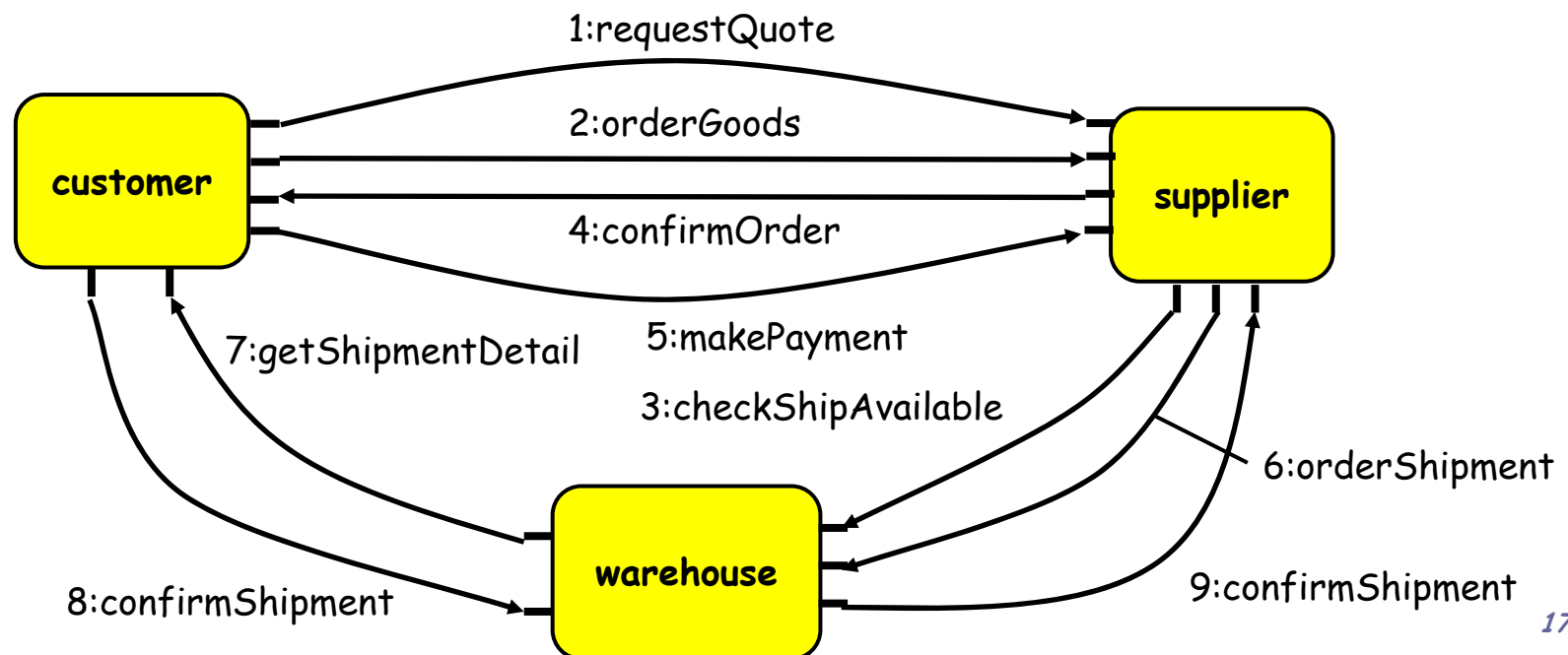
Conversations

- A service is not simply a set of independent operations
 - Using a service typically involves performing sequences of operations in a particular order (**conversations**)
 - During a conversation, the client typically chooses the next operation to invoke (on the basis of previous results, etc.) among the ones that the service allows at that point

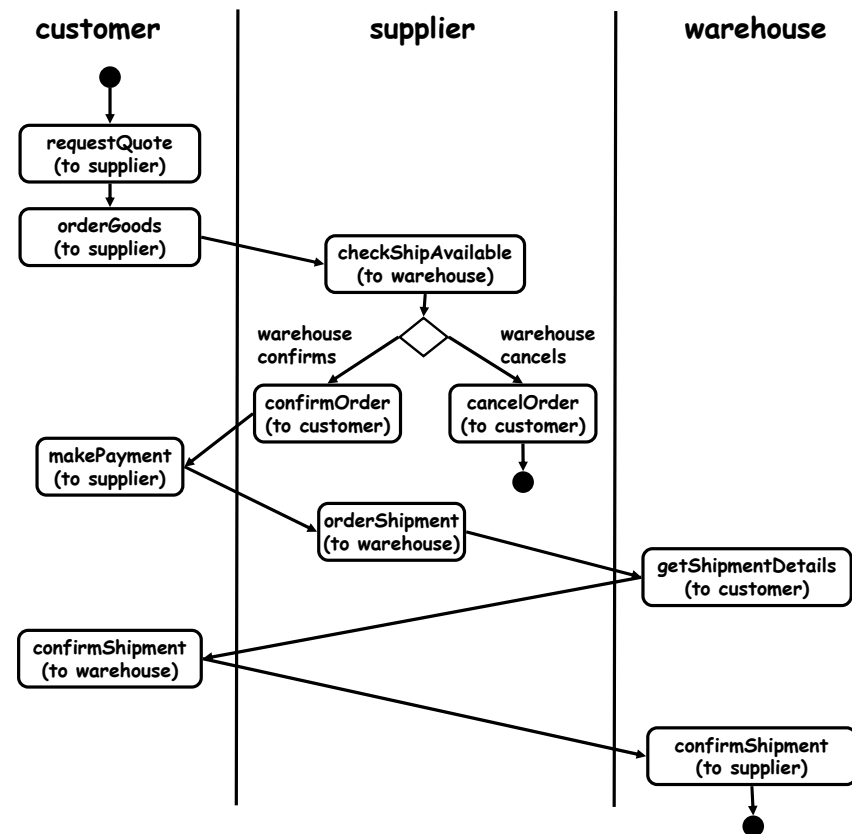
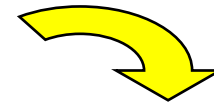
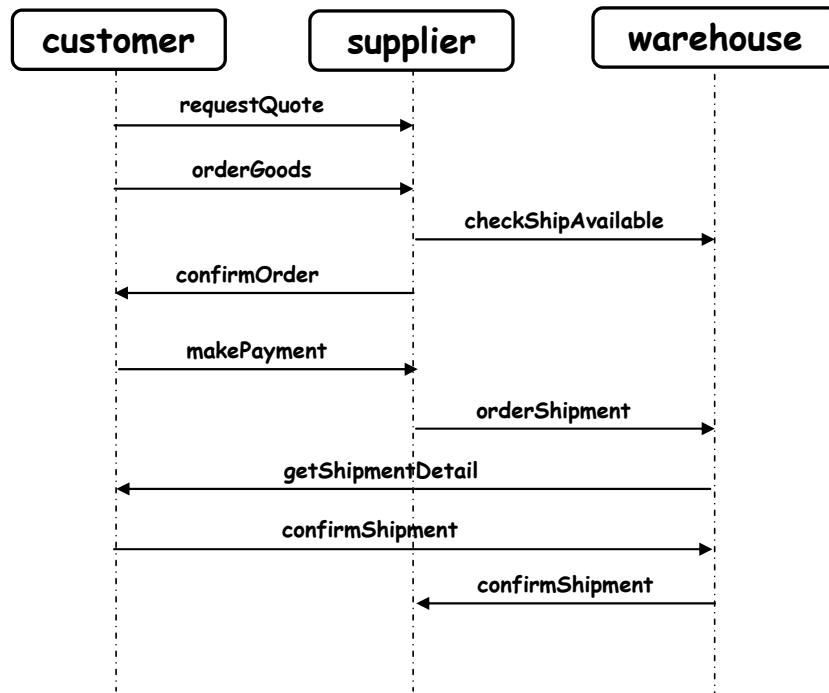


Choreography: Coordination of Conversations of N Services

- Global specification of the conversations of N **peer** services (i.e., multi-party conversations)
 - Roles
 - Message exchanges
 - Constraints on the order in which such exchanges should occur



Choreography: Coordination of Conversations of N Services



[from ACKM04]

ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Giuseppe De Giacomo & Massimo Mecella

18

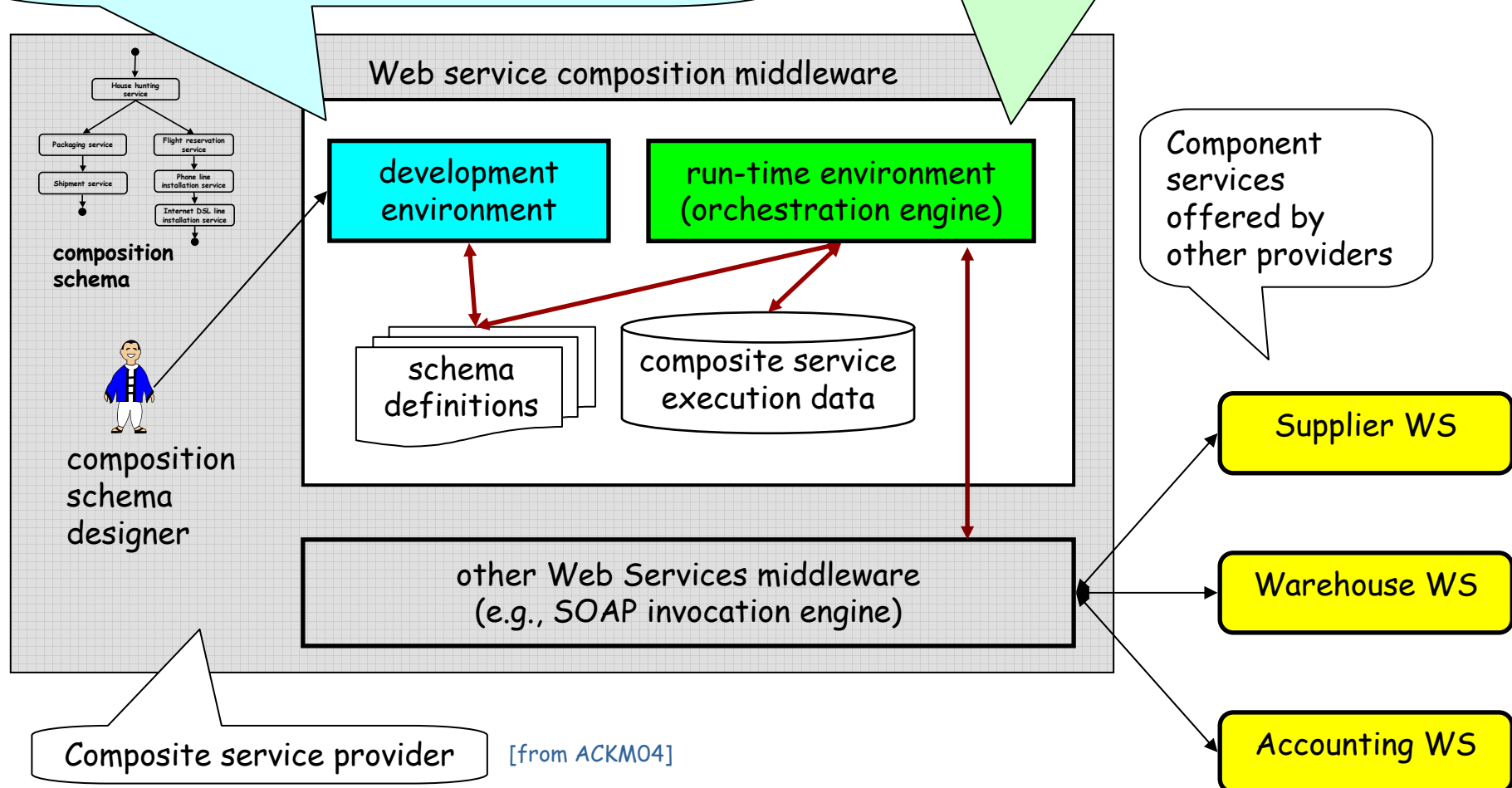
Composition

- Deals with the **implementation** of an application (in turn offered as a service) whose application logic **involves the invocation of operations offered by other services**
 - The new service is the *composite service*
 - The invoked services are the *component services*

The Composition Engine/Middleware

Through the development environment, a **composition schema** is **synthesized**, either manually or (semi-)automatically. A service composition model and a language (maybe characterized by a graphical and a textual representation) are adopted

Orchestration: the run-time environment executes the composite service business logic by invoking other services (through appropriate protocols)



Synthesis and Orchestration

- (Composition) Synthesis: building the specification of the composite service (i.e., the composition schema)
 - Manual
 - Automatic
- Orchestration: the run-time management of the composite service (invoking other services, scheduling the different steps, etc.)
 - Composition schema is the "program" to be executed
 - Similarities with WfMSs (Workflow Management Systems)

Composition Schema

- A composition schema specifies the "process" of the composite service
 - The "workflow" of the service
- Different clients, by interacting with the composite service, satisfy their specific needs (reach their goals)
 - A specific execution of the composition schema for a given client is an orchestration instance

Choreography (Coordination) vs. Composition (Orchestration)

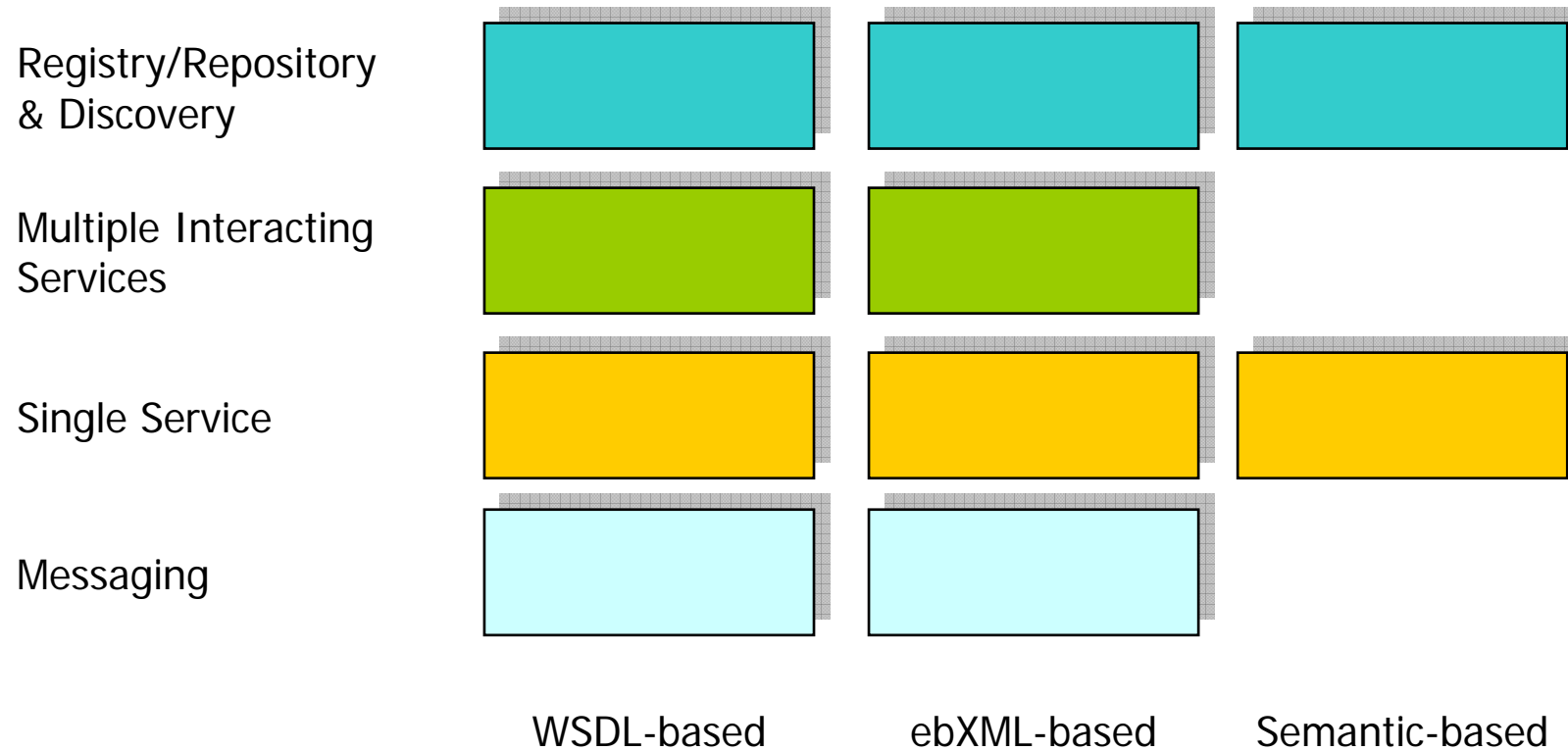
- Composition is about implementing new services
 - From the point of view of the client, a composite service and a basic (i.e., implemented in a traditional programming language) one are indistinguishable
- Choreography is about global modeling of N peers, for proving correctness, design-time discovery of possible partners and run-time bindings
- N.B.: There is a strong relationship between a service internal composition and the external choreographies it can participate in
 - if A is a composite service that invokes B, the A's composition schema must reflect the coordination protocol governing A - B interactions
 - in turn, the composition schema of A determines the coordination protocols that A is able to support (i.e., the choreographies it can participate in)

Relevant Technologies and Abstractions

Relevant Technologies and Abstractions: Outline

- Outline of the Different "Stacks"
 - WS-DL, WS-BPEL, WS-CDL
 - A Brief Introduction to ebXML
 - A Brief Introduction to DAML Semantic Web Services
- Transition Systems (TS)
 - WS-DL --> TS
 - TS --> WS-BPEL

The "Stacks" of Service Technologies



Relevant Technologies

The WSDL-based Stack

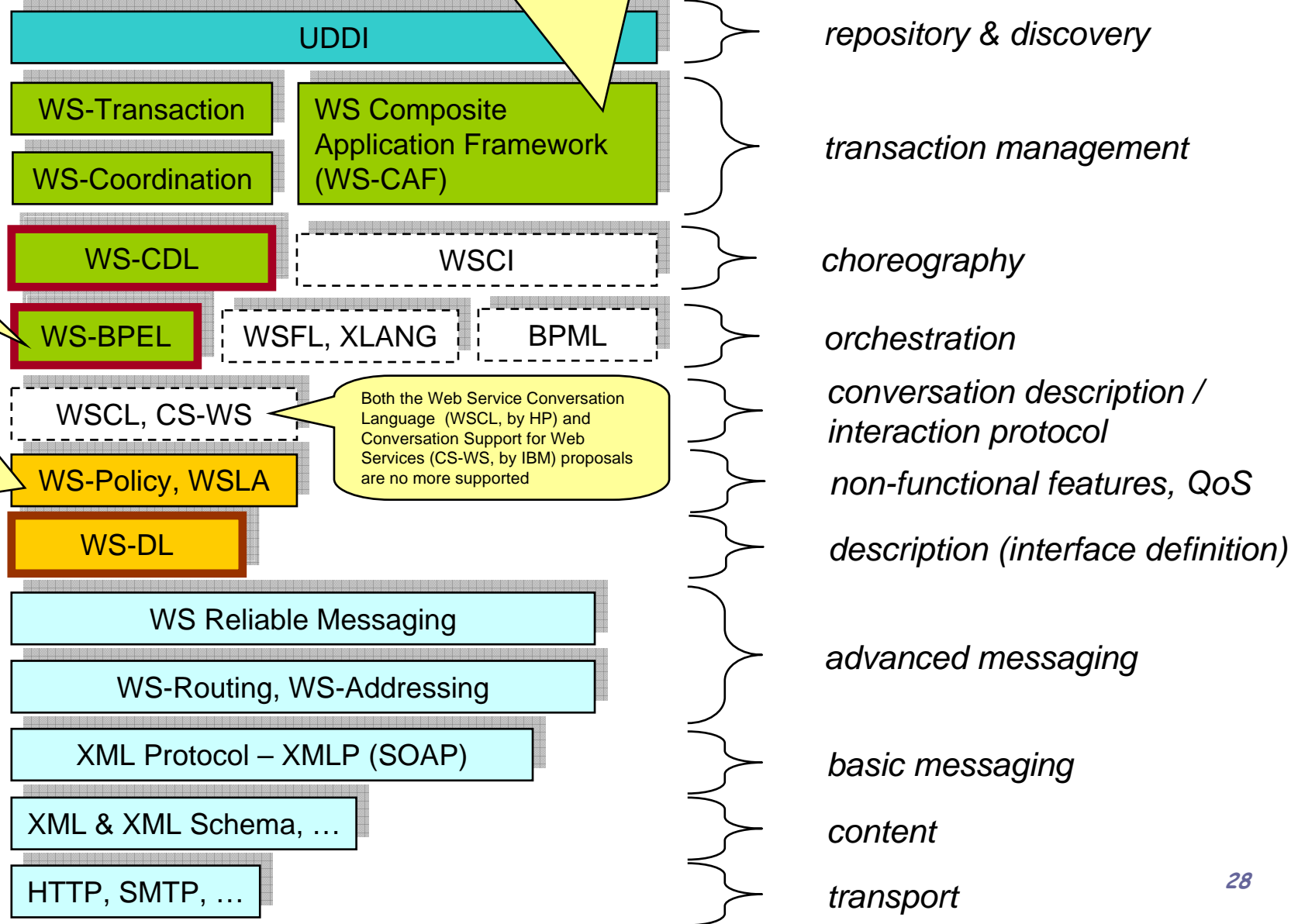
The WSDL-based "Stack"

Includes 3 specifications:
 (i) Web Service Context (WS-CTX)
 (ii) Web Service Coordination Framework (WS-CF)
 (iii) Web Service Transaction Management (WS-TXM)

Formerly BPEL4WS (BPEL for short)

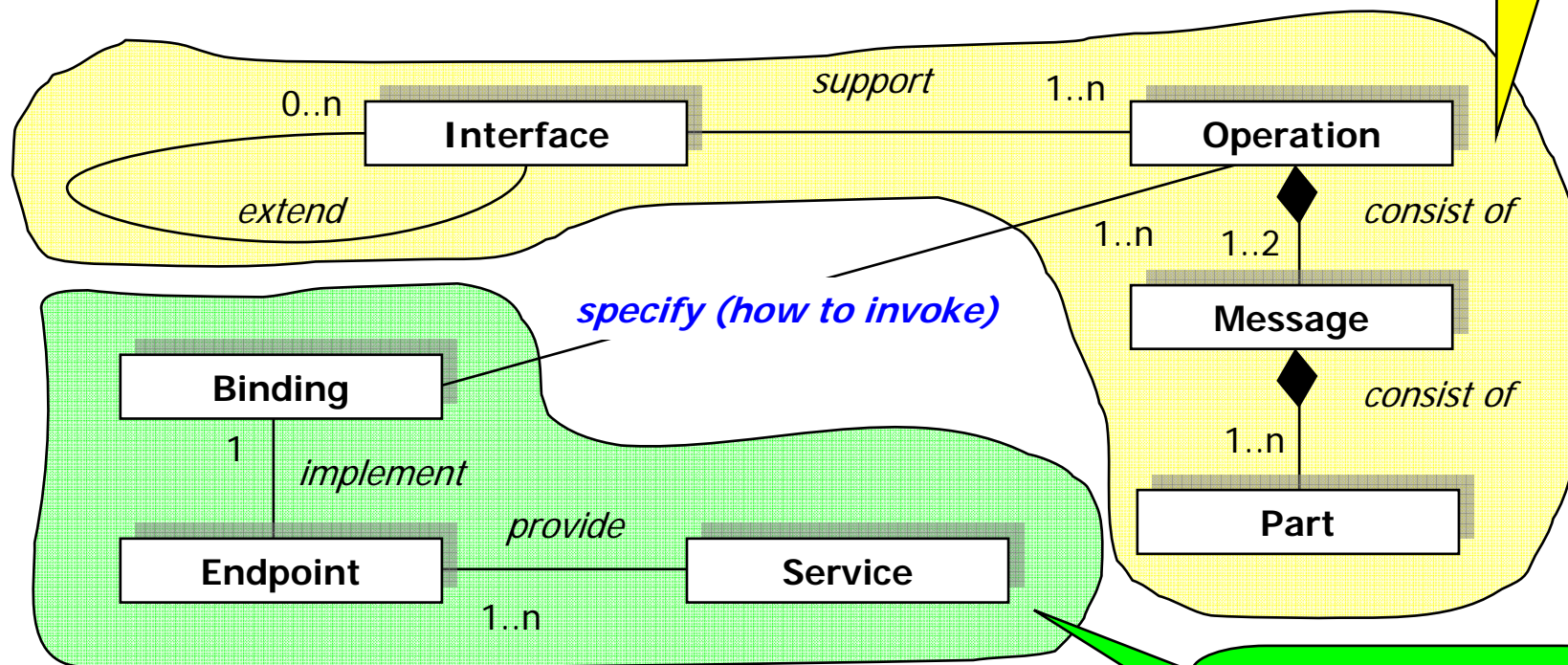
(i) formerly also WSEL (Web Service Endpoint Language) by IBM
 (ii) also the research/academic proposal Web Service Offering Language [WSOL]

Both the Web Service Conversation Language (WSCL, by HP) and Conversation Support for Web Services (CS-WS, by IBM) proposals are no more supported



Web Service Definition Language (WS-DL)

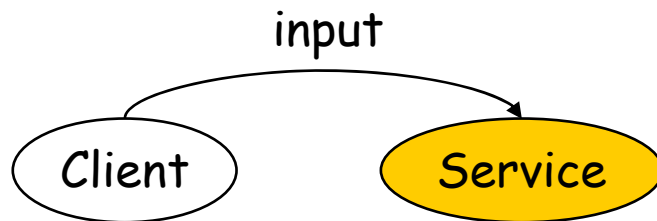
- WS-DL (v2.0) provides a framework for defining
 - Interface: operations and input/output formal parameters
 - Access specification: protocol bindings (e.g., SOAP)
 - Endpoint: the location of service



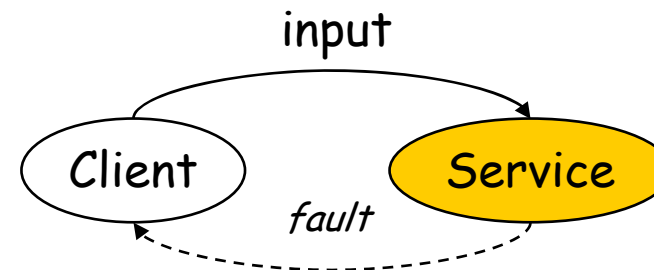
Service interface
(abstract definition)

Service implementation
(concrete definition)

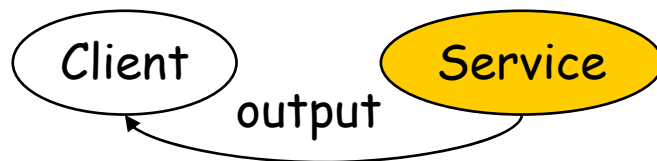
Message Exchange Patterns (1)



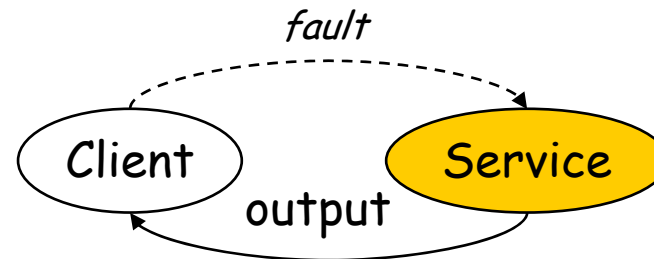
in-only (no faults)



robust in-only (message triggers fault)

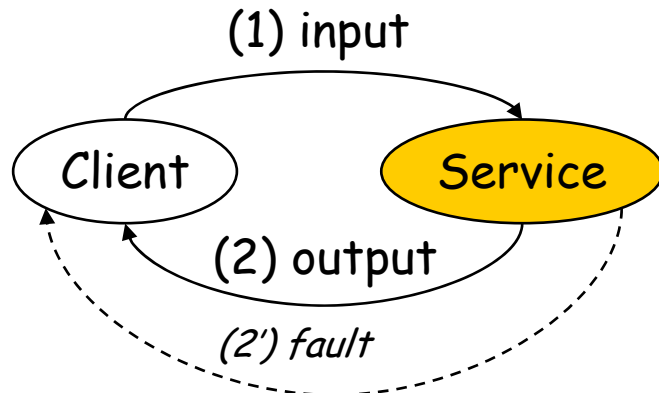


out-only (no faults)

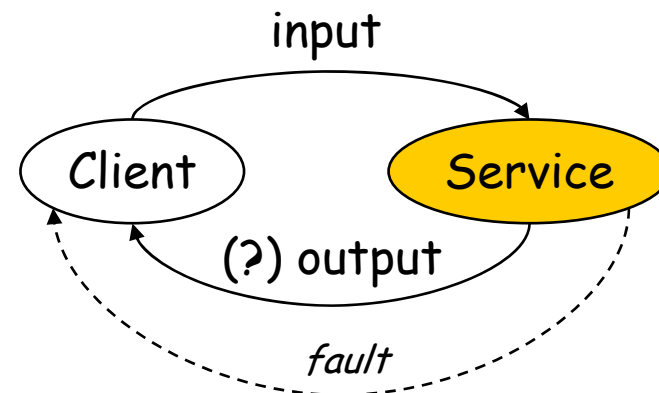


robust out-only (message triggers fault)

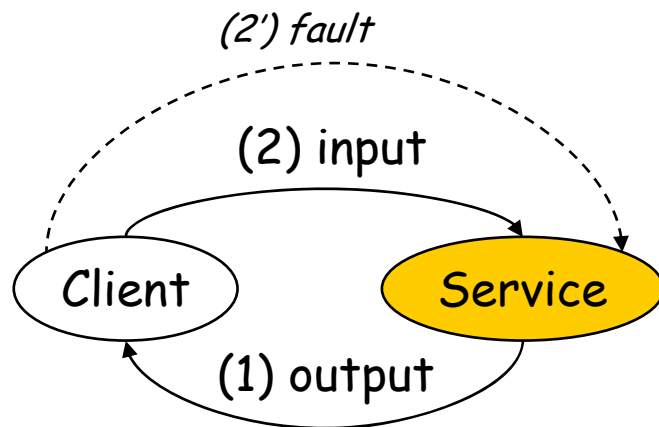
Message Exchange Patterns (2)



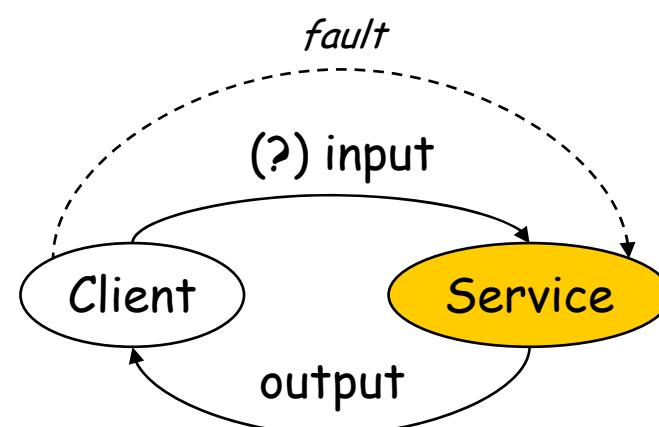
in-out (fault replaces message)



in-optional-out (message triggers fault)



out-in (fault replaces message)



out-optional-in (message triggers fault)

An Example (1)

```
<definitions ... >
  <types>
    <element name="ListOfSong_Type">
      <complexType><sequence>
        <element minOccurs="0" maxOccurs="unbound"
          name="SongTitle" type="xs:string"/>
      </sequence></complexType>
    </element>
    <element name="SearchByTitleRequest">
      <complexType><all>
        <element name="containedInTitle"
          type="xs:string"/>
      </all></complexType>
    </element>
    <element name="SearchByTitleResponse">
      <complexType><all>
        <element name="matchingSongs"
          xsi:type="ListOfSong_Type"/>
      </all></complexType>
    </element>
  </types>
</definitions>
```

Definition of a
message and its
formal
parameter

An Example (2)

```
<element name="SearchByAuthorRequest">
  <complexType><all>
    <element name="authorName"
      type="xs:string"/>
  </all></complexType>
</element>
<element name="SearchByAuthorResponse">
  <complexType><all>
    <element name="matchingSongs"
      xsi:type="ListOfSong_Type"/>
  </all></complexType>
</element>
<element name="ListenRequest">
  <complexType><all>
    <element name="selectedSong"
      type="xs:string"/>
  </all></complexType>
</element>
```

An Example (3)

```
<element name="ListenResponse">
  <complexType><all>
    <element name="MP3fileURL" type="xs:string"/>
  </all></complexType>
</element>
<element name="ErrorMessage">
  <complexType><all>
    <element name="cause" type="xs:string"/>
  </all></complexType>
</element>
</types>
```

An Example (4)

Definition of a service interface

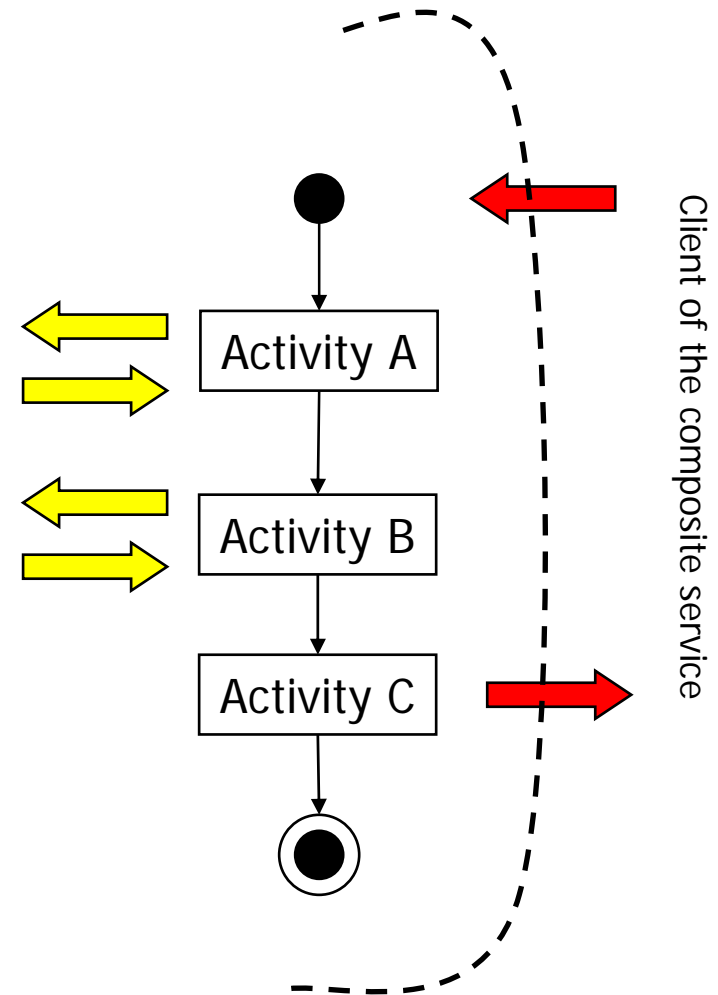
```
<interface name="MP3ServiceType">
  <operation name="search_by_title" pattern="in-out">
    <input message="SearchByTitleRequest"/>
    <output message="SearchByTitleResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
  <operation name="search_by_author" pattern="in-out">
    <input message="SearchByAuthorRequest"/>
    <output message="SearchByAuthorResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
  <operation name="listen" pattern="in-out">
    <input message="ListenRequest"/>
    <output message="ListenResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
</interface>
</definitions>
```

Definition of an operation and its message exchange pattern



Business Process Execution Language for Web Services (WS-BPEL)

- Allows specification of composition schemas of Web Services
 - Business processes as coordinated interactions of Web Services
 - Business processes as Web Services
- Allows abstract and executable processes
- Influenced from
 - Traditional flow models
 - Structured programming
 - Successor of WSFL and XLANG
- Component Web Services described in WS-DL (v1.1)



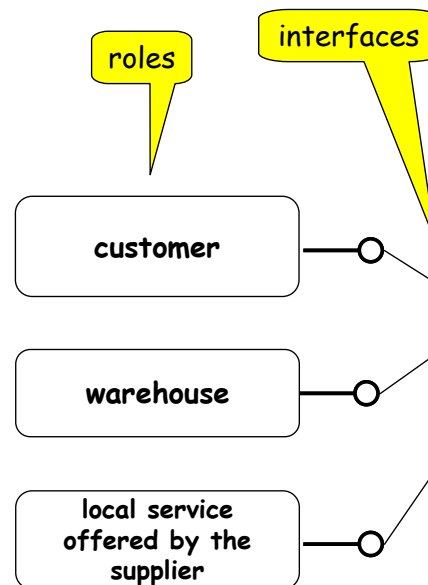
WS-BPEL Specification

An XML document specifying

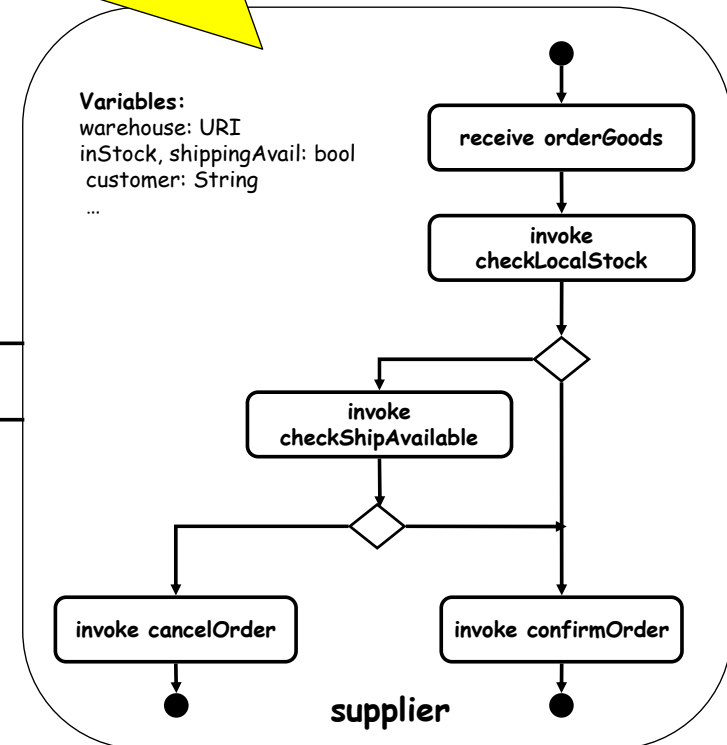
- Roles exchanging messages with the composite service/process
- The (WSDL) interfaces supported by such roles

• The orchestration of the process

- Variables and data transfer
- Exception handling
- Correlation information



Orchestration
- variables and data transfers,
- exception handling,
- correlation information (for instance routing)



Process Model (Activities)

- Primitive
 - **invoke**: to invoke a Web Service (in-out) operation
 - **receive**: to wait for a message from an external source
 - **reply**: to reply to an external source message
 - **wait**: to remain idle for a given time period
 - **assign**: to copy data from one variable to another
 - **throw**: to raise exception errors
 - **empty**: to do nothing
- Structured
 - **sequence**: sequential order
 - **switch**: conditional routing
 - **while**: loop iteration
 - **pick**: choices based on events
 - **flow**: concurrent execution (synchronized by **links**)
 - **scope**: to group activities to be treated "transactionally" (managed by the same fault handler, within the same transactional context)

A link connects exactly one source activity S to exactly one target activity T; T starts only after S ends. An activity can have multiple incoming (possibly with join conditions) and outgoing links. Links can be guarded

Process Model

(Data Manipulation and Exception Handling)

- Blackboard approach
 - a blackboard of variables is associated to each orchestration instance (i.e., a shared memory within an orchestration instance)
 - variables are not initialized at the beginning; they are modified (read/write) by assignments and messages
 - manipulation through XPath
- Try-catch-throw approach
 - definition of fault handlers
 - ... but also event handlers and compensation handlers (for managing transactionality as in the SAGA model)

Choreography

(As Reported in Literature: Classical Ballet Style)

- Consider a dance with more than one dancer
 - Each dancer has a set of steps that they will perform. They orchestrate their own steps because they are in complete control of their domain (their body)
 - A choreographer ensures that the steps all of the dancers make is according to some overall, pre-defined scheme. This is a choreography
 - The dancers have no control over the steps they make: their steps must conform to the choreography
 - The dancers have a single view-point of the dance
 - The choreographer has a multi-party or global view-point of the dance

Choreography

(A Possible Evolution: Jam Session Style)

- Consider a jazz band with many players
 - There is a rhythm and a main theme. This is the choreography
 - Each player executes his piece by improvising variations over the main theme and following the given rhythm
 - The players still have a single view-point of the music; in addition they have full control over the music they play
 - There is a multi-party or global view-point of the music, but this is only a set of "sketchy" guidelines

WS-BPEL vs. WS-CDL

- Orchestration/WS-BPEL is about describing and executing a single peer
- Choreography/WS-CDL is about describing and guiding a global model (N peers)
- You should derive the single peer from the global model by projecting based on participant

WS-CDL Basics (1)

- **Participants & Roles**
 - **Role type**
 - Enumerate the observable behavior that a collaborating participant exhibits
 - Behavior type specifies the operations supported
 - Optional WSDL interface type
 - **Relationship type**
 - Specify the mutual commitments, in terms of the Roles/Behavior types, **two** collaborating participants are required to provide
 - Note: all multi-party relationships are transformed into binary ones
 - **Participant type**
 - Enumerate a set of one or more Roles that a collaborating participant plays

WS-CDL Basics (2)

- Channels
 - A channel realizes a *dynamic* point of collaboration, through which collaborating participants interact
 - Where & how to communicate a message
 - Specify the *Role/Behavior* and the *Reference* of a collaborating participant
 - Identify an *Instance* of a Role
 - Identify an instance of a conversation between two or more collaborating participants
 - A conversation groups a set of related message exchanges
- One or more channel(s) *MAY* be passed around from a Role to one or more other Role(s), possibly in a daisy fashion through one or more intermediate Role(s), creating new points of collaboration dynamically
 - A Channel type *MAY* restrict the types of Channel(s) allowed to be exchanged between the Web Services participants, through this Channel
 - A Channel type *MAY* restrict its usage, by specifying the number of times a Channel can be used

WS-CDL Basics (3)

- **Activities** are the building blocks of a choreography
 - Basic Activity
 - **Interaction**: message exchange between participants
 - Only in-out and in-only
 - **Assign**: within one role, assign the value of a variable to another one
 - Variables can be about information (exchanged documents), states and channels
 - **No action**: do null
 - Ordering structure
 - **Sequence** (P.Q)
 - **Parallel** (P | Q)
 - **Choice** (P + Q)
 - **Perform**: a complete, separately defined choreography is performed
 - Basis for scalable modeling

Attention: a choreography performing another one is referred to as "choreography composition" in the standard

WS-CDL Basics (4)

- A Choreography combines all previous elements, forming a collaboration unit of work
 - Enumerate all the binary relationships interactions act in
 - Localize the visibility of variables
 - Using variable definitions
 - Prescribe alternative patterns of behavior
 - Using work/units and reactions
 - Enable Recovery
 - Using work/units and reactions
 - Backward: handle exceptional conditions
 - Forward: finalize already completed activities

Relevant Technologies

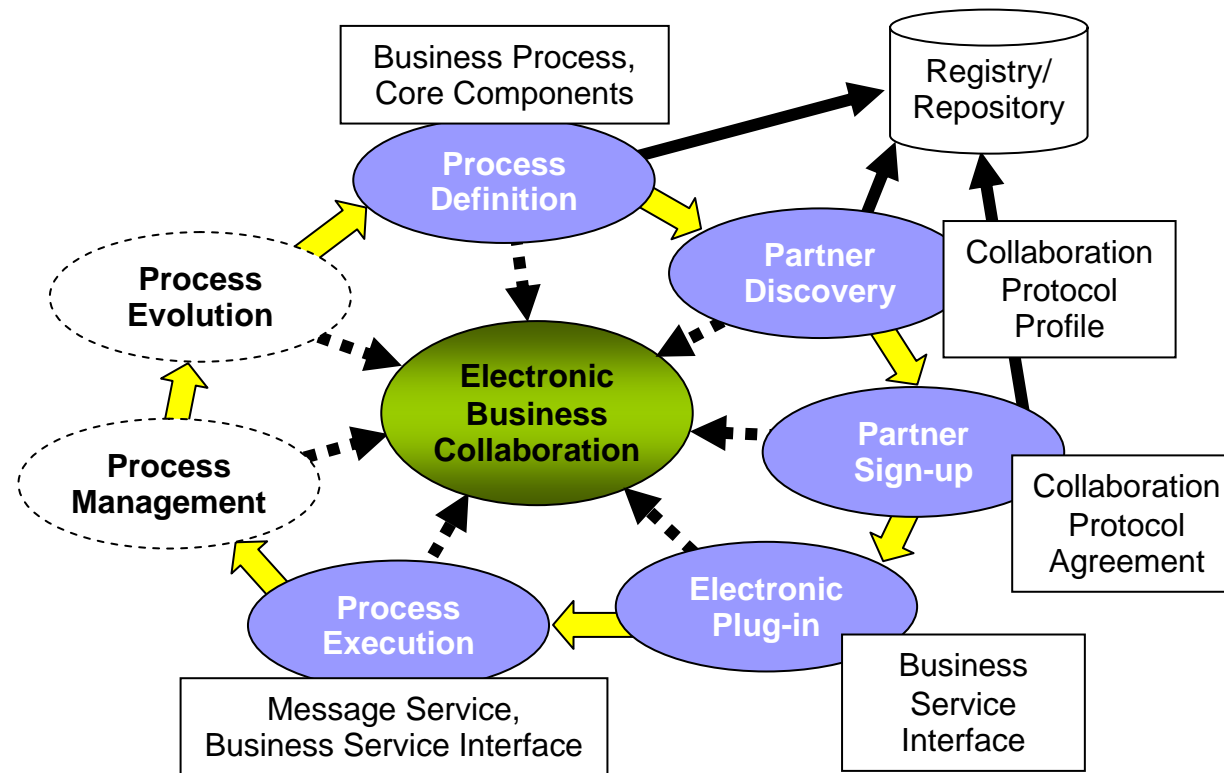
ebXML

ebXML

- ebXML is more a standardized "conceptual framework", a "reference model", than a real stack of standard technologies
 - *Stable version in 2001/2002*
 - Technical Architecture Specification (v1.04)
 - Business Process Specification Schema (v1.01)
 - Registry Information Model (v2.0)
 - Registry Services Specification (v2.0)
 - Requirements Specification (v1.06)
 - Collaboration-Protocol Profile and Agreement Specification (v2.0)
 - Message Service Specification (v2.0)
- Currently in revision
 - Indeed, many Technical Committees (TCs) are working in synergy with the promoters of the W3C/WSDL-based "stack"
 - E.g., UDDI v2 has been developed in the context of ebXML/OASIS, currently WS-BPEL and WS-CAF are being evolved/developed in the context of specific TCs, etc.

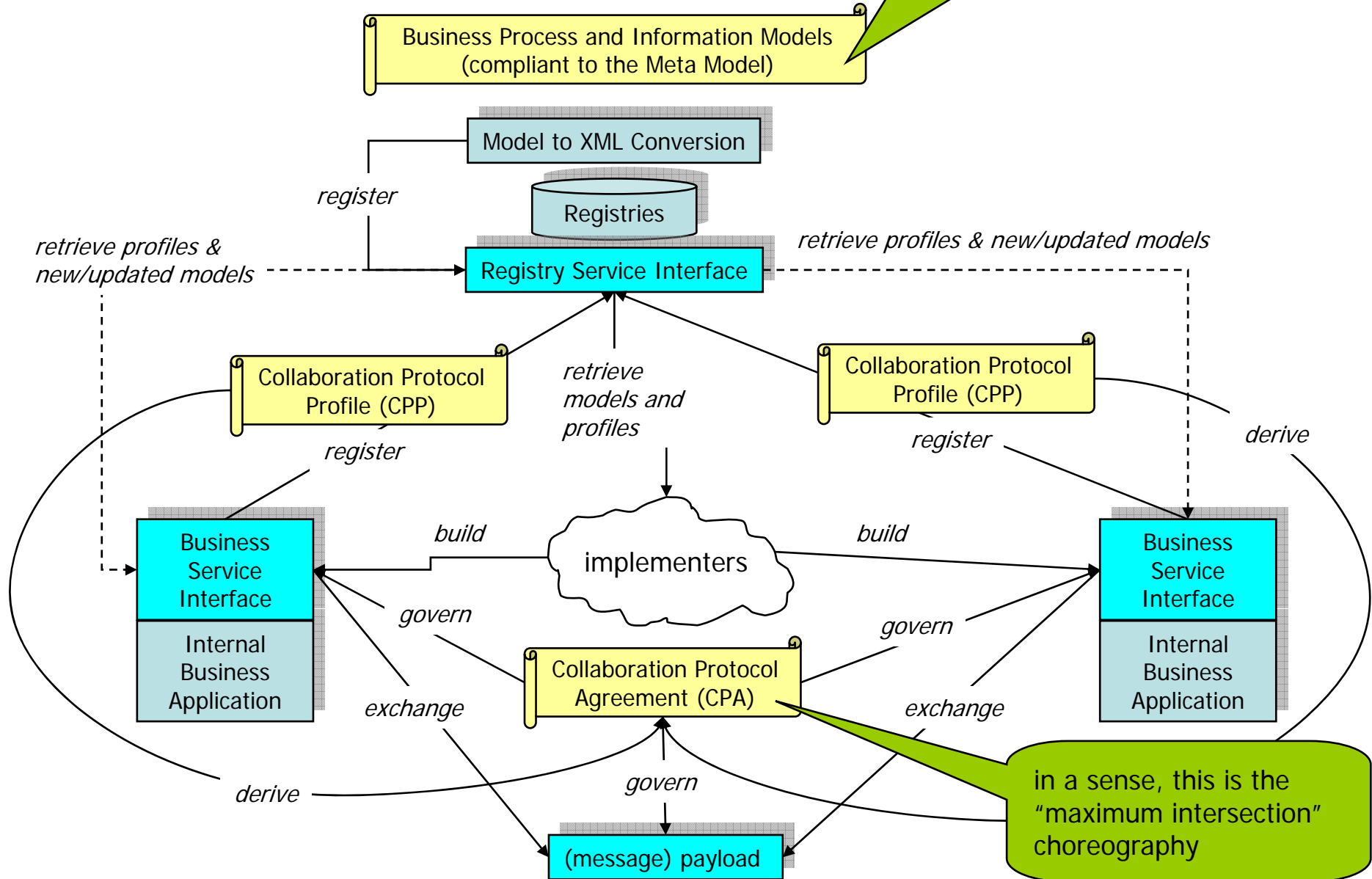
ebXML: Aims

- To define an open & public infrastructure, based on XML, for distributed electronic commerce
 - Special attention to SMEs and developing countries



ebXML: How ?

by using BPSS
(Business Process
Specification Schema)



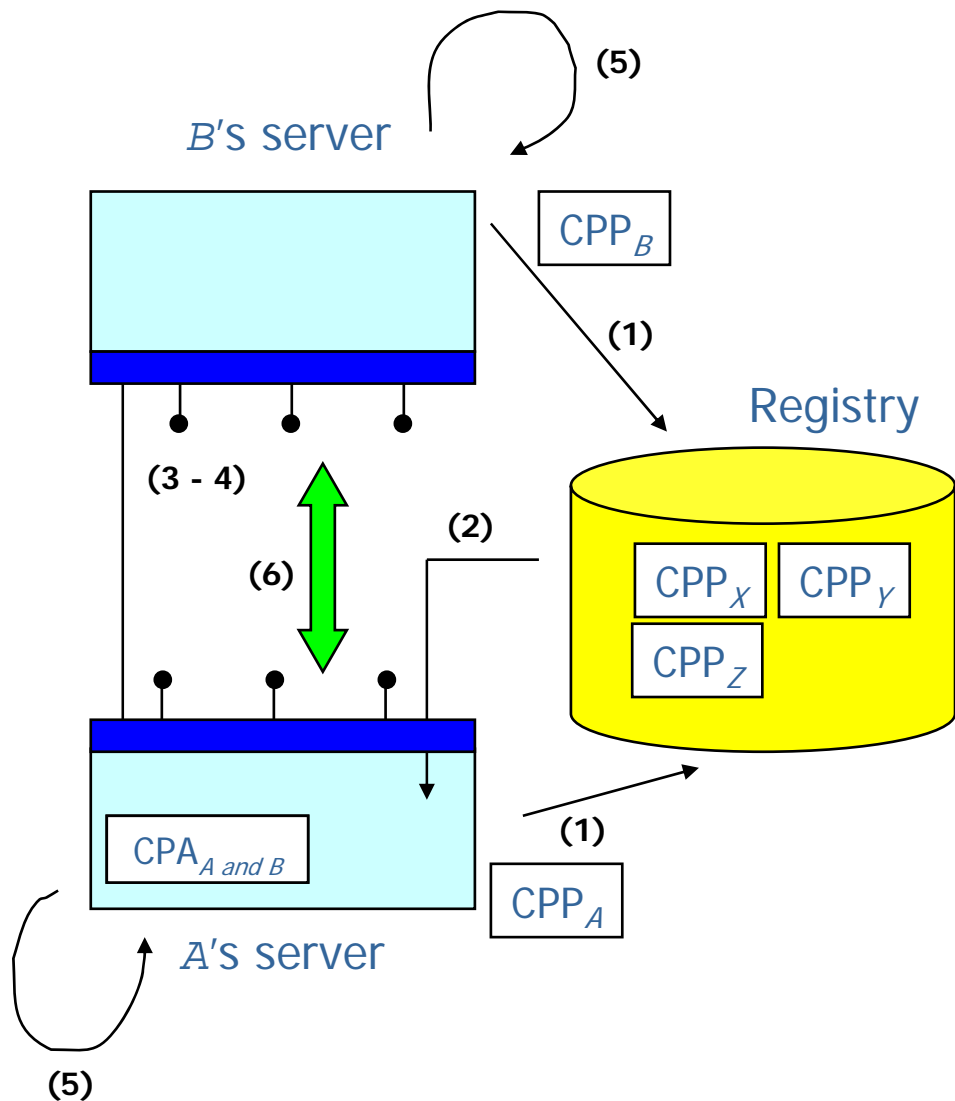
ebXML: BPSS, CPP e CPA (1)

- BPSS is used for modeling a business process, thus obtaining a BPS (Business Process Specification)
 - Partners, roles, *collaborations* and document exchanges (*business transactions*)
 - Collaboration: set of *activities*, an activity is a business transaction or again a collaboration
 - Business transaction: a partner is the requester, the other is the responder, in a *business document flow*
- CPP: expresses the capabilities of a partner in participating in a BPS

ebXML: BPSS, CPP e CPA (2)

- ✓ D wants to make electronic business with E; D is the acquirer and E the vendor; the process underlying the business is already defined in a BPS
- ✓ D discovers the E#s CPP in a registry
- A CPA is created, as the intersection of D#s CPP and E#s CPP
- On the basis of the CPA, the D#s and E#s business service interfaces are configured in order to support the business transactions

ebXML: BPSS, CPP e CPA (3)



1. Each partner has registered its own CPP in the registry
2. Partner D# discovers E in the registry and download CPP_E on its system
3. Partner D# creates CPA_{D#and E#} and sends it to E
4. After a negotiation (both manual or automatic), both D# and E# register identical copies of the agreed upon CPA_{D#and E#} in their systems
5. Both D# and E# configure their systems for runtime on the basis of CPA_{D#and E}
6. Finally D# and E# engage their e-Commerce process

Relevant Technologies

DAML Semantic Web Services

OWL-S (formely DAML-S)

- An emerging standard to add semantics
 - An upper ontology for **describing properties & capabilities** of Web Services using OWL
- **Enable** automation of various activities (e.g., service discovery & selection)



OWL-S Service Profile

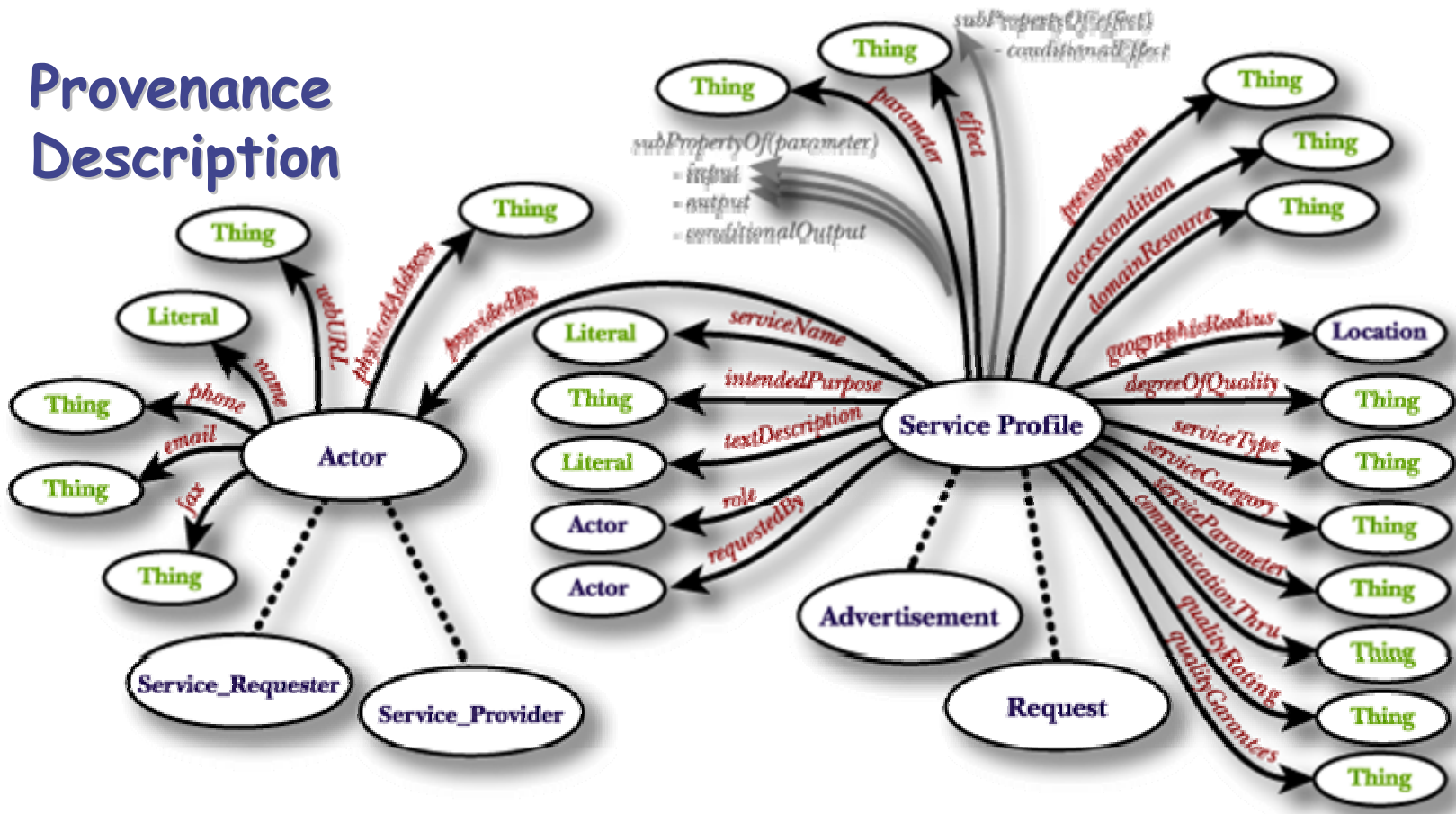
(What it does)

- High-level characterization/summary of a service
 - Provider & participants
 - Capabilities
 - Functional attributes (e.g., QoS, region served)
- Used for
 - Populating service registries
 - A service can have many profiles
 - Automated service discovery
 - Service selection (matchmaking)
- One can derive:
 - Service advertisements
 - Service requests

OWL-S Service Profile

Capability Description

Provenance Description

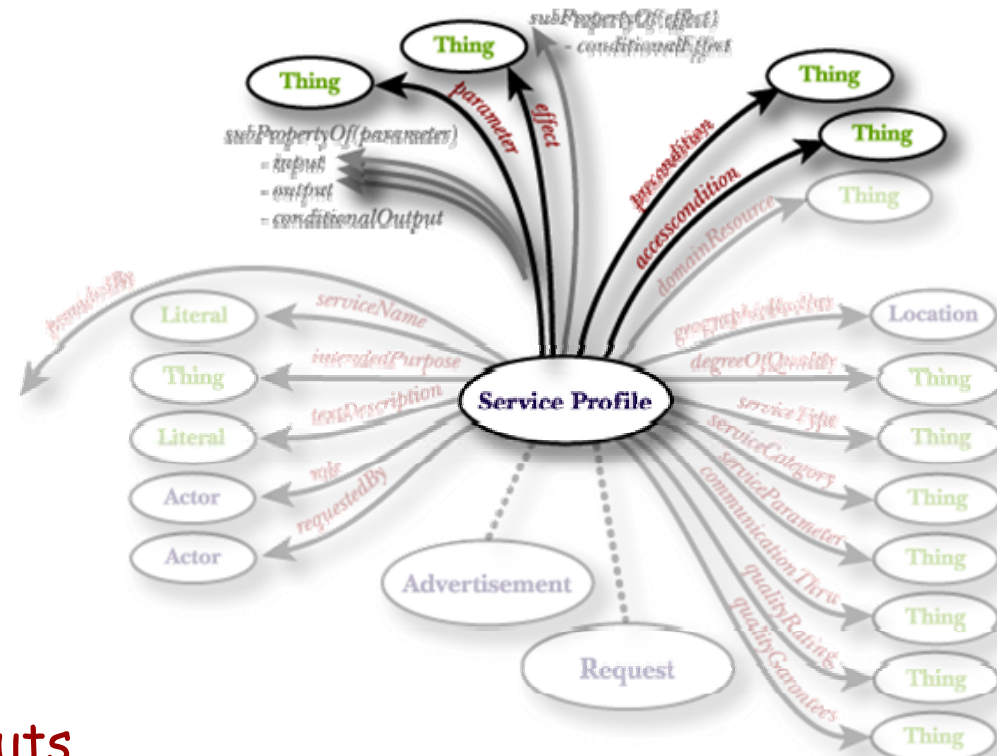


Functional Attributes

[from DAML-S]
ICSOC 2004 Tutorial (Nov. 15, 2004 - New York, NY, USA)

Capability Description

- Specification of what the service provides
 - High-level functional representation in terms of:
 - preconditions
 - inputs
 - (conditional) outputs
 - (conditional) effects

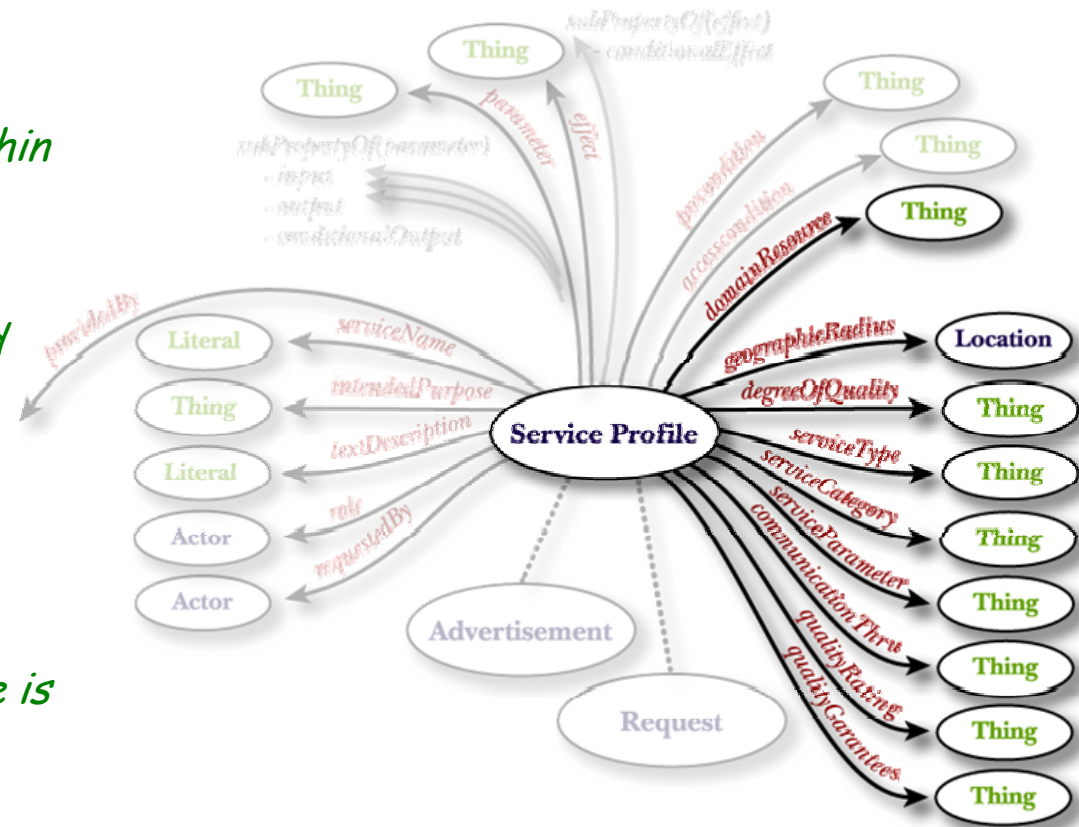


- **Inputs**
 - Set of necessary inputs that the requester should provide to invoke the service
- **(Conditional) Outputs**
 - Results that the requester should expect after interaction with the service provider is completed
- **Preconditions**
 - Set of conditions that should hold prior to service invocation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

Functional Attributes

Provide supporting information about the service, including:

- geographical scope
Pizza Delivery only within the Pittsburgh area
- quality descriptions and guarantees
Stock quotes delivered within 10 secs
- service types, service categories
Commercial / Problem Solving, etc.
- service parameters
Average Response time is currently ...



OWL-S Process Ontology

- **Atomic processes:** directly invocable, no subprocesses, executed in a single step
- **Composite processes:** consist of other (non-composite or composite) processes
- **Simple processes:** a virtual view of atomic process or composite process (as a "black box")

Process Model

- Constructs for complex processes
 - Sequence
 - Concurrency: Split; Split+Join; Unordered
 - Choice
 - If-Then-Else
 - Looping: Repeat-Until; Iterate (non-deterministic)
- Data Flow
 - No explicit variables, no internal data store
 - Predicate "sameValues" to match input of composite service and input of subordinate service
- Less refined than, e.g., WS-BPEL

SWSL

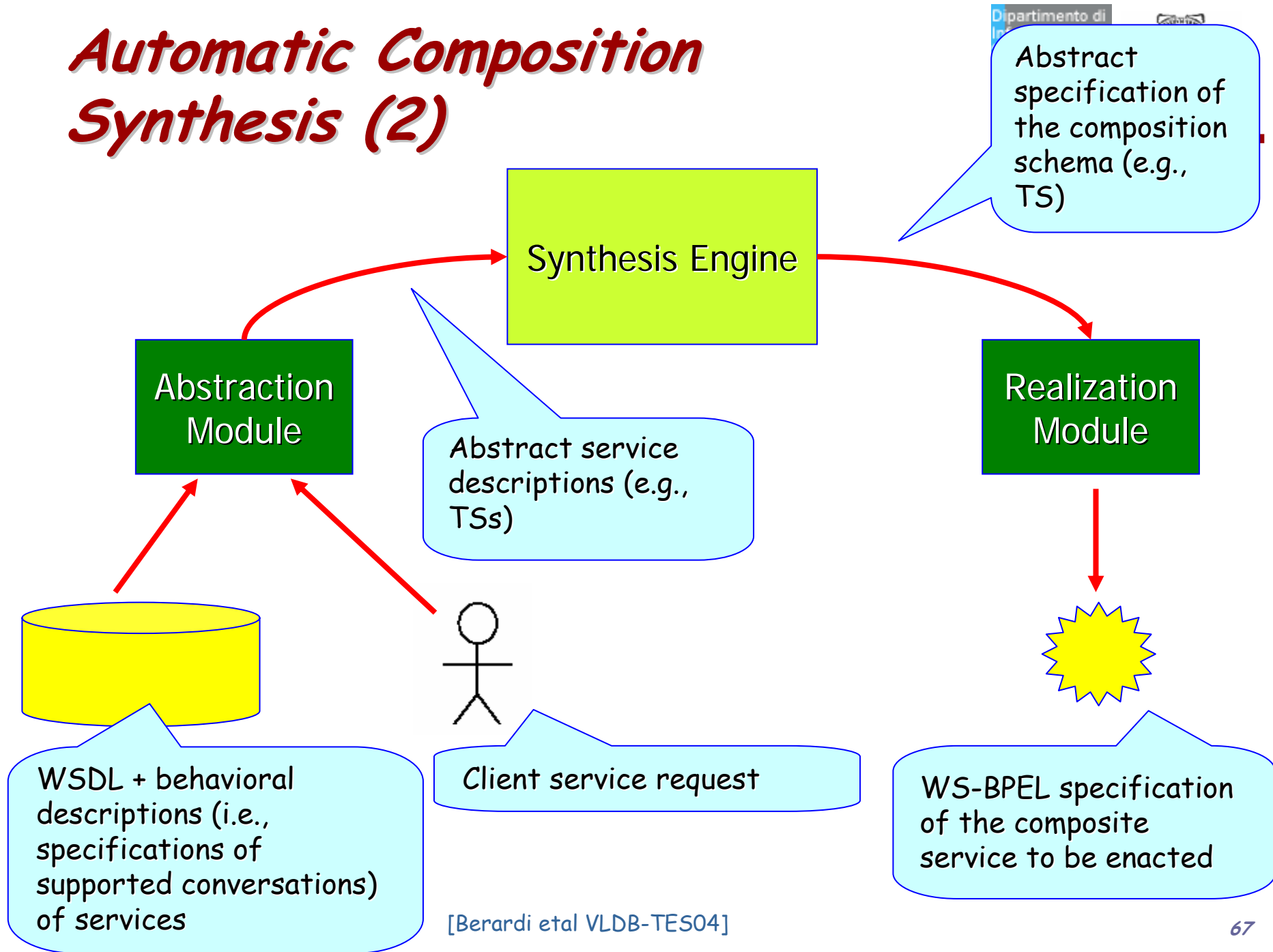
- SWSL (Semantic Web Service Language) is being developed to enhance Service Profile and Service Model. It consists of (as of May 2004):
 - ROSE (Rules and Ontologies for web SErvices) and ?*&! (i.e., CTR++)
 - Logic-based rule languages
 - Focus on security & trust and semantic interoperability (i.e., mapping outputs of one service to input of another service)
 - FLOWS (First-order Logic Ontology for Web Services)
 - Based on a first-order variant of Situation Calculus, i.e., PSL (Process Specification Language, ISO 18629)
 - Focus on "rich" process modeling & semantics
- Work in Progress !!

Introducing Automatic Composition

Automatic Composition Synthesis (1)

- Given:
 - a set (S_1, \dots, S_n) of component services
 - a client service request T
- Automatically build:
 - a composition schema CS that fulfills T by suitably orchestrating (S_1, \dots, S_n)

Automatic Composition Synthesis (2)



Abstracting over Technologies

Modeling Services as
Transition Systems

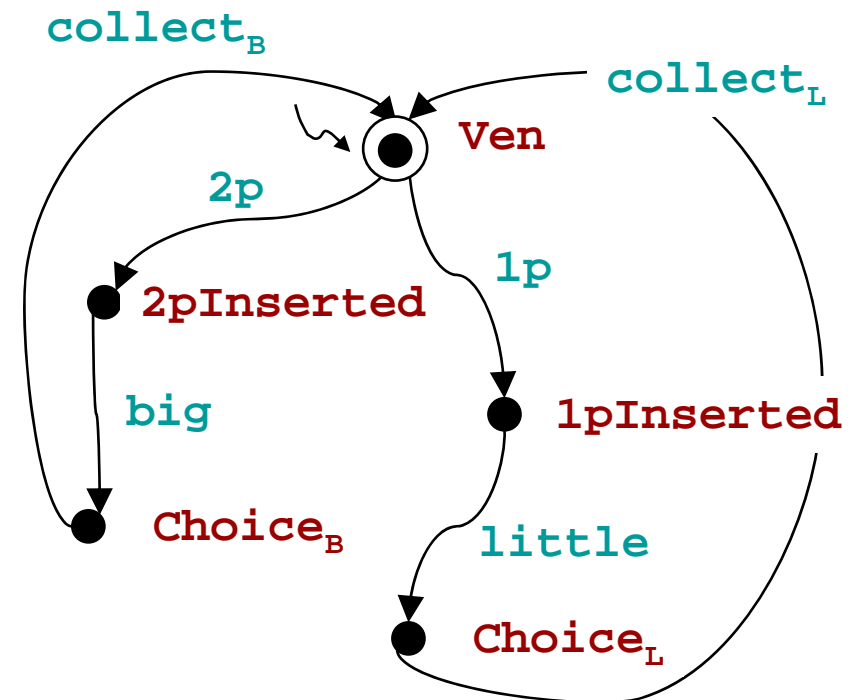
Transition Systems

- A transition system (TS) is a tuple

$$T = \langle A, S, S^0, \delta, F \rangle$$

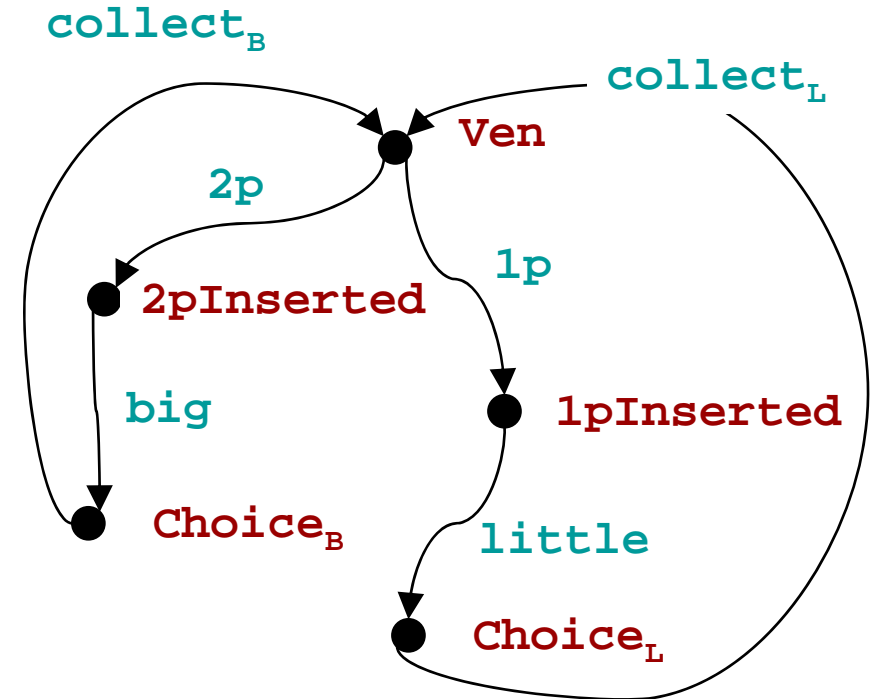
where:

- A is the set of actions
- S is the set of states
- $S^0 \subseteq S$ is the set of initial states
- $\delta \subseteq S \times A \times S$ is the transition relation
- $F \subseteq S$ is the set of final states



Process Algebras and TSs

- Process theory:
 - a process is a term of an algebraic language
 - a transition $E \rightarrow_a F$ means that process E may become F by performing (participating in, or accepting) action a
 - structured rules guide the derivation



- A graph:
 - nodes are process terms
 - labelled directed arcs between nodes

$$\text{Ven} = 2p.2p\text{Inserted} + 1p.1p\text{Inserted}$$

$$2p\text{Inserted} = \text{big}.Choice_B$$

$$1p\text{Inserted} = \text{little}.Choice_L$$

$$Choice_B = \text{collect}_B.\text{Ven}$$

$$Choice_L = \text{collect}_L.\text{Ven}$$

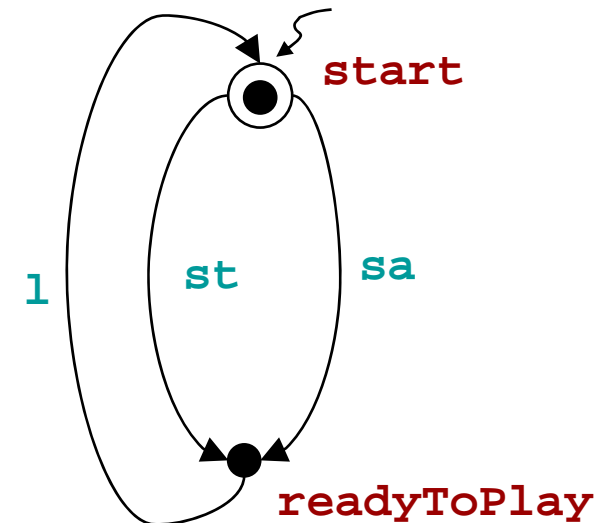
WS-DL is the Set of Actions

- A message exchange pattern (and the related operation) represents an **interaction** with the service client
 - an action that the service can perform by interacting with its client
- Abstracting from formal parameters, we can associate a different symbol to each operation ...
- ... thus obtaining the alphabet of actions

An Example

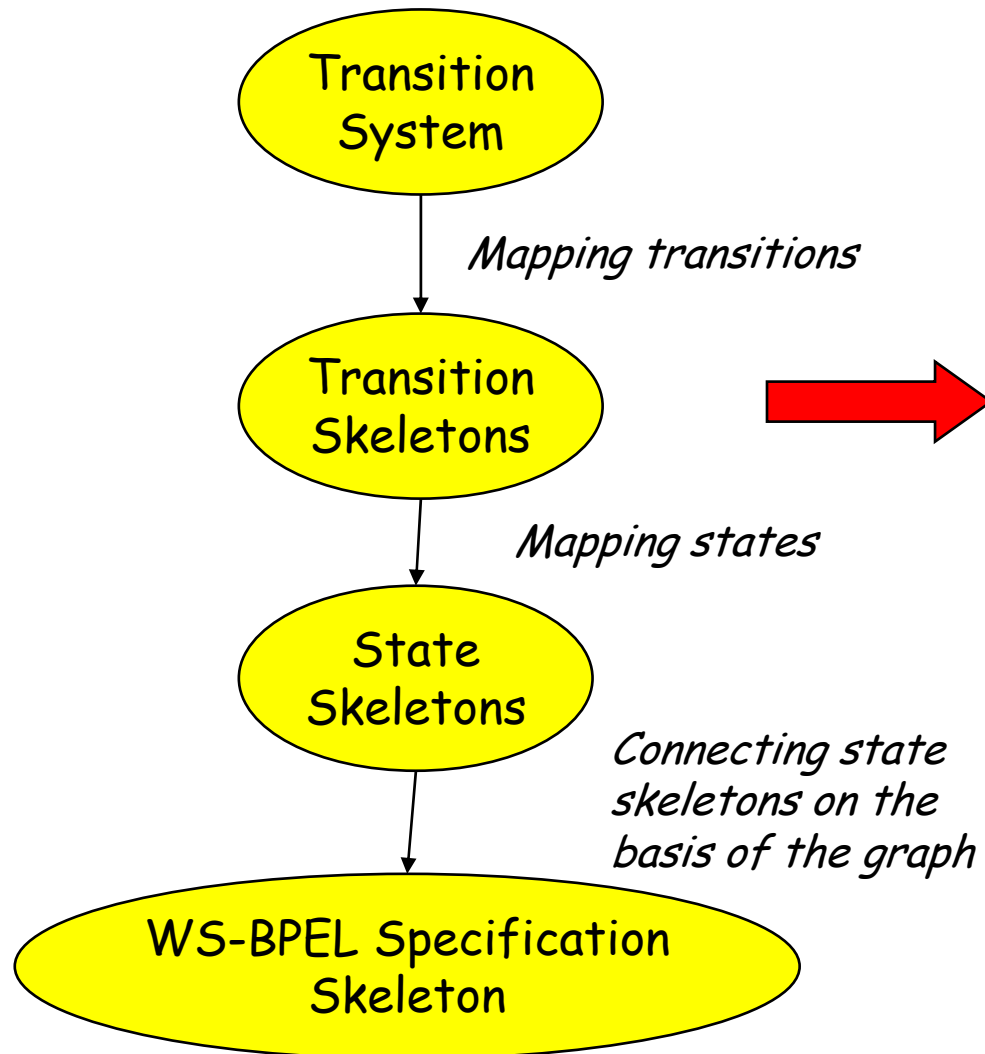
- The MP3ServiceInterface defines 3 actions:

- search_by_title / st
- search_by_author / sa
- listen / l



- Formally $A = \{st, sa, l\}$

From a TS to WS-BPEL (1)



```
<process name = "...">
  <partnerLinks>
    ...
  </partnerLinks>
  <variables>
    ...
  </variables>
  <flow>
    <links>
      ...
    </links>
    <!-- state skel. -->
    ...
    <!-- state skel. -->
  </flow>
</process>
```

From a TS to WS-BPEL (2)

Intuition [Baina etal CAISE04, Berardi etal VLDB-TES04]

1. Each transition corresponds to a WS-BPEL pattern consisting of (i) an `<onMessage>` operation (in order to wait for the input from the client of the composite service), (ii) followed by the effective logic of the transition, and then (iii) a final operation for returning the result to the client. Of course both before the effective logic and before returning the result, messages should be copied forth and back in appropriate variables
2. All the transitions originating from the same state are collected in a `<pick>` operation, having as many `<onMessage>` clauses as transitions originating from the state
3. The WS-BPEL file is built visiting the graph in depth, starting from the initial state and applying the previous rules.

N.B.: (1) and (2) works for in-out interactions (the one shown in the following). Simple modifications are needed for in-only, robust-in-only and in-optional-out. The other kinds of interactions implies a proactive behaviour of the composite service, possibly guarded by `<onAlarm>` blocks.
(3) works for acyclic TS. See later for cycle management.

Transition Skeletons

```
<onMessage ... >  
  <sequence>  
    <assign>  
      <copy>  
        <from variable="input" ... />  
        <to variable="transitionData" ... />  
      </copy>  
    </assign>  
    <!-- logic of the transition -->  
    <assign>  
      <copy>  
        <from variable="transitionData" ... />  
        <to variable="output" ... />  
      </copy>  
    </assign>  
    <reply ... />  
  </sequence>  
</onMessage >
```

State Skeletons

- N transitions from state S_i are mapped onto:

```
<pick name = "Si">  
  <!-- transition #1 -->  
  <onMessage ... >  
  <!-- transition skeleton -->  
  </onMessage>  
  
  ... ..  
  <!-- transition #N -->  
  <onMessage ... >  
  <!-- transition skeleton -->  
  </onMessage>  
</pick>
```

Mapping the TS

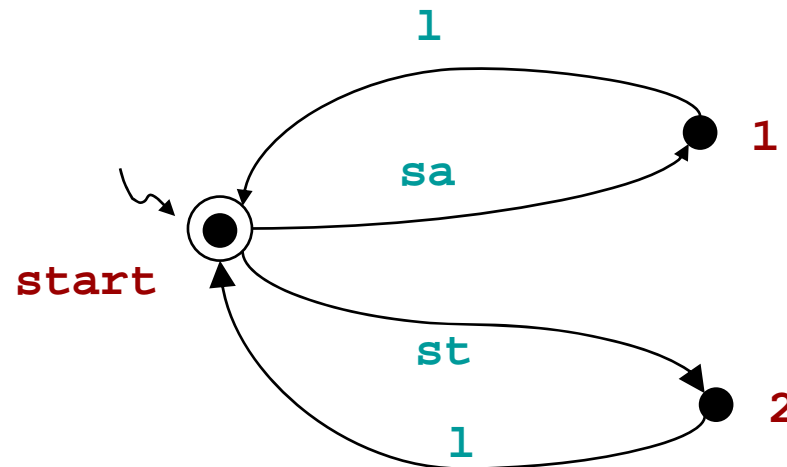
- All the `<pick>` blocks are enclosed in a surrounding `<flow>`; the dependencies are modeled as `<link>s`
 - `<link>s` are controlled by specific variables s_i -to- s_j that are set to TRUE iff the transition $S_i \rightarrow S_j$ is executed
 - Each state skeleton has many outgoing `<link>s` as states connected in output, each going to the appropriate `<pick>` block

Mapping Cyclic TSs

(Intuition)

- Identify all the cycles
- Enclose the involved state skeletons inside a `<while>` block controlled by a condition (`!exit`) (`exit` is a variable defined ad hoc)
 - `exit` is set to `FALSE` by any transition that "goes out" of the cycle
 - The overall `<while>` block is connected to other state skeletons by appropriate `<link>`s
- Special cases:
 - A state `S` with self-transitions can be represented as a `<pick>` block enclosed in a `<while>` block controlled by a condition (`vs`) (the variable `vs` is set to `FALSE` by other non self-transitions)
 - Cycles starting from the initial state should not be considered, as they can be represented as the start of a new instance

An Example (1)



<partnerLinks>

<!-- The "client" role represents the requester of this composite service -->

<partnerLink name="client"

partnerLinkType="tns:Transition"

myRole="MP3ServiceTypeProvider"

partnerRole="MP3ServiceTypeRequester"/>

<partnerLink name="service"

partnerLinkType="nws:MP3CompositeService"

myRole="MP3ServiceTypeRequester"

partnerRole="MP3ServiceTypeProvider"/>

</partnerLinks>

An Example (2)

```
<variables>
  <variable name="input" messageType="tns:listen_request"/>
  <variable name="output" messageType="tns:listen_response"/>
  <variable name="dataIn" messageType="nws:listen_request"/>
  <variable name="dataOut" messageType="nws:listen_response"/>
</variables>

  <pick>
    <onMessage partnerLink="client"
      portType="tns:MP3ServiceType"
      operation="listen"
      variable="input">
      <sequence>
        <assign>
          <copy>
            <from variable="input" part="selectedSong"/>
            <to variable="dataIn" part="selectedSong"/>
          </copy>
        </assign>
        ... ..
        <assign>
          <copy>
            <from variable="dataOut" part="MP3FileURL"/>
            <to variable="output" part="MP3FileURL"/>
          </copy>
        </assign>
        <reply name="replyOutput"
          partnerLink="client"
          portType="tns:MP3ServiceType"
          operation="listen"
          variable="output"/>
      </sequence>
    </onMessage>
    ... ..
  </pick>
```

An Example (3)



```
<process suppressJoinFailure = "no">
```

```
  <flow>
```

```
  <links>
```

```
    <link name="start-to-1"/>
```

```
    <link name="start-to-2"/>
```

```
  </links>
```

```
  <pick createInstance = "yes">
```

```
    <onMessage="sa">
```

```
      <sequence>
```

```
        <copy>...</copy>
```

```
        ... ..
```

```
        <copy>...</copy>
```

```
        <reply ... />
```

```
      </sequence>
```

```
    </onMessage>
```

```
    <onMessage="st">
```

```
      <sequence>
```

```
        <copy>...</copy>
```

```
        ... ..
```

```
        <copy>...</copy>
```

```
        <reply ... />
```

```
      </sequence>
```

```
    </onMessage>
```

```
    <source linkName="start-to-1" transitionCondition = "bpws:getVariableData('start-to-1') = 'TRUE' " />
```

```
    <source linkName="start-to-2" transitionCondition = "bpws:getVariableData('start-to-2') = 'TRUE' " />
```

```
  </pick>
```

A new instance is created in the initial state. This resolve also the presence of the cycles without the need of enclosing <while>

The <sa> transition skeleton should set variables:

```
start-to-1 = TRUE
```

```
start-to-2 = FALSE
```

The <st> transition skeleton should set variables:

```
start-to-1 = FALSE
```

```
start-to-2 = TRUE
```

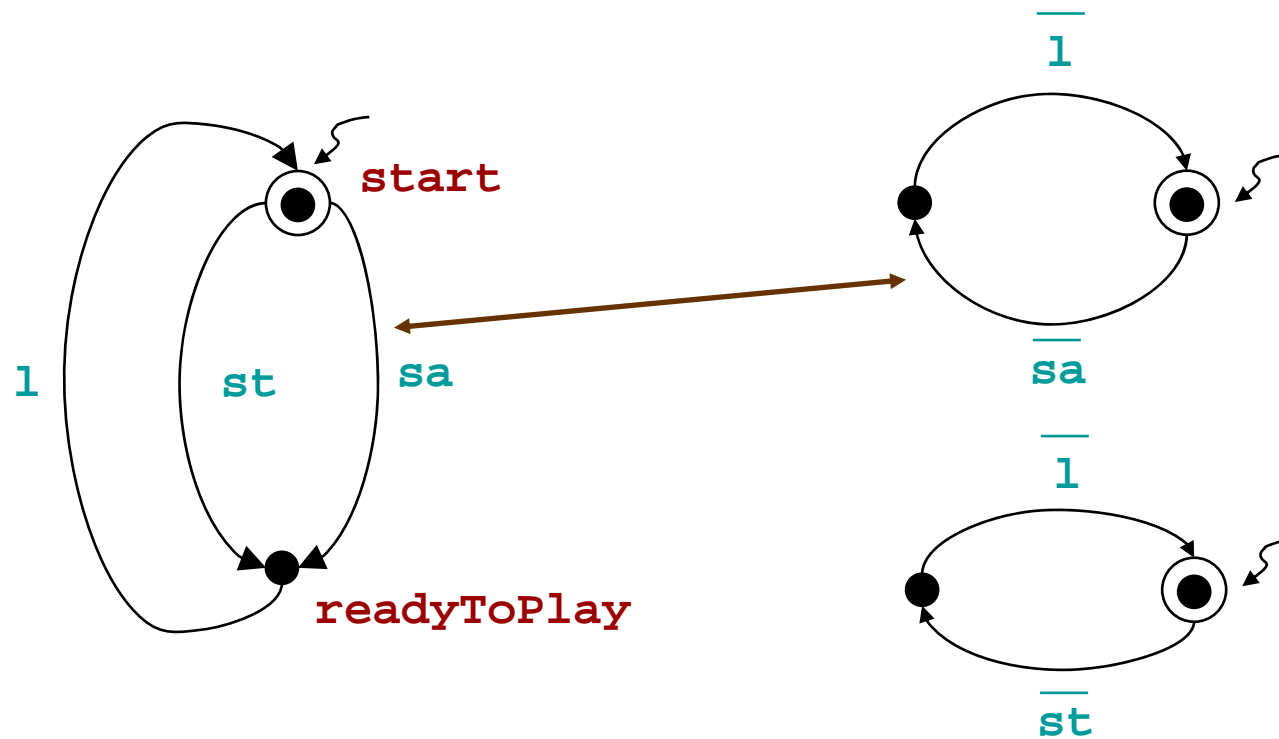
An Example (4)

```
<pick>
  <onMessage="">
    <sequence>
      <copy>...</copy>
      ...
      <copy>...</copy>
      <reply ... />
    </sequence>
  </onMessage>
  <target linkName="start-to-1" />
</pick>
<pick>
  <onMessage="">
    <sequence>
      <copy>...</copy>
      ...
      <copy>...</copy>
      <reply ... />
    </sequence>
  </onMessage>
  <target linkName="start-to-2" />
</pick>
</process>
```

TSs and Choreography

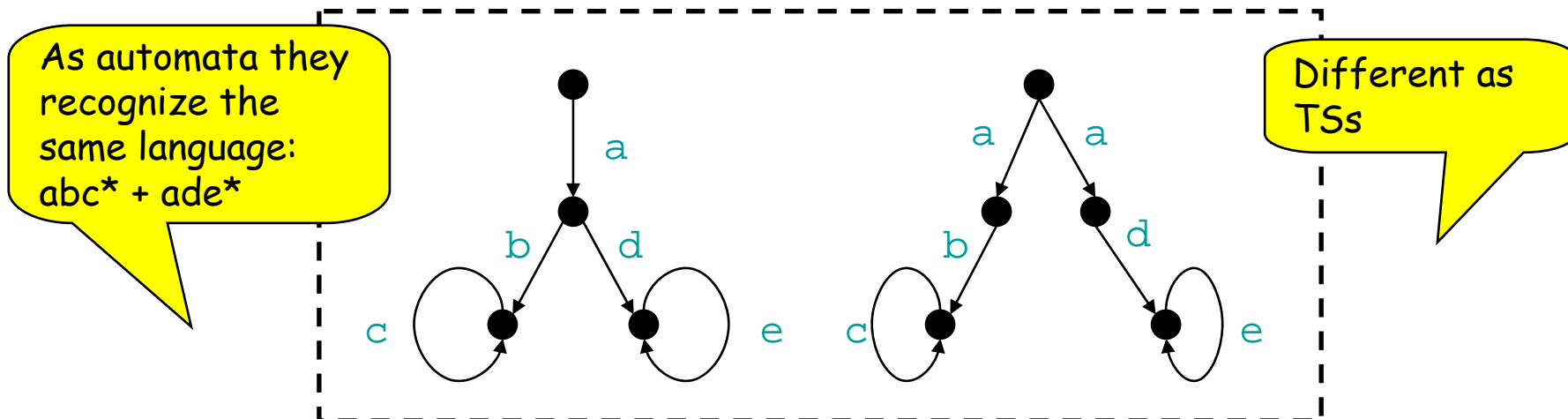
(only an intuition :-)

- A Choreography can be seen as the specification of a set of concurrent peers, each one exposing a TS, that fulfills the global model



Automata vs. Transition Systems

- Automata
 - define sets of runs (or traces or strings): (finite) length sequences of actions
- TSs
 - ... but I can be interested also in the alternatives "encountered" during runs, as they represent client's "choice points"



References

- [ACKM04] - G. Alonso, F. Casati, H. Kuno, V. Machiraju: *Web Services. Concepts, Architectures and Applications*. Springer-Verlag 2004
- [VLDBJ01] - F. Casati, M.C. Shan, D. Georgakopoulos (eds.): *Special Issue on e-Services*. VLDB Journal, 10(1), 2001
Based on the 1st International Workshop on Technologies for e-Services (VLDB-TEs 2001)
- [CACM03] - M.P. Papazoglou, D. Georgakopoulos (eds.): *Special Issue on Service Oriented Computing*. Communications of the ACM 46(10), 2003
- [WSOL] - V. Tomic, B. Pagurek, K. Patel, B. Esfandiari, W. Ma: *Management Applications of the Web Service Offerings Language (WSOL)*. To be published in *Information Systems*, Elsevier, 2004.
An early version of this paper was published in *Proc. of CAiSE'03*, LNCS 2681, pp. 468-484, 2003
- [Berardi et al WSCC04] - D. Berardi, R. Hull, M. Gruninger, S. McIlraith: *Towards a First-Order Ontology for Semantic Web Services*. Proc. W3C International Workshop on Constraints and Capabilities for Web Services (WS-CC), 2004, <http://www.w3.org/2004/06/ws-cc-cfp.html>
- [Benatallah et al IJCIS04] - B. Benatallah, F. Casati, H. Skogsrud, F. Toumani: *Abstracting and Enforcing Web Service Protocols*, International Journal of Cooperative Information Systems (IJCIS), 13(4), 2004

References

- [Baina et al CAISE04] K. Baina, B. Benatallah, F. Casati, F. Toumani: Model-driven Web Service Development, Proc. of CAiSE'04, LNCS 3084, 2004
- [Berardi et al ICSOC03] - D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: Proc. of ICSOC'03, LNCS 2910, 2004
- [Berardi et al VLDB-TES04] - D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: Post-proc. of VLDB-TES'04, to appear
- [Stirling Banff '96] - C. Stirling: Modal and Temporal Logics for Processes. Banff Higher Order Workshop, LNCS 1043, 1996. Available at: <http://homepages.inf.ed.ac.uk/cps/banff.ps>
- [ebpml] - Jean-Jacques Dubray: the ebPML.org Web Site, <http://www.ebpml.org/>
- [DAML-S] - DAML Semantic Web Services, <http://www.daml.org/services>

References

- [WS-Policy] - Web Services Policy Framework (WS-Policy), September 2004, <http://www-106.ibm.com/developerworks/library/specification/ws-polfram/>
- [WSCL] - Web Services Conversation Language (WSCL) 1.0. W3C Note, 14 March 2002, <http://www.w3.org/TR/wscl10/>
- [WSLA] - A. Dan, D. Davis et al: Web Services On Demand: WSLA-driven Automated Management. IBM Systems Journal, 43(1), 2004
- [ebXML] - Electronic Business using eXtensible Markup Language, <http://www.ebxml.org/>
- [OASIS] - Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/home/index.php>
- [WSDL] - R. Chinnici, M. Gudgin, J.J. Moreau, J. Schlimmer, and S. Weerawarana, Web Services Description Language (WSDL) 2.0, Available on line: <http://www.w3.org/TR/wsdl20>, 2003, W3C Working Draft.
- [BPEL4WS] - T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, Business Process Execution Language for Web Services (BPEL4WS) -Version 1.1, <http://www-106.ibm.com/developerworks/library/ws-bpel/>, 2004

References

[WS-CDL] - N. Kavantzias, D. Burdett, G. Ritzinger, Y. Lafon: Web Services Choreography Description Language (WS-CDL) Version 1.0, Available on line at: <http://www.w3.org/TR/ws-cdl-10/>, W3C Working Draft.

[UDDI] - Universal Discovery, Description and Integration, <http://www.uddi.org/>

[WS-C] - Web Services Coordination (WS-C), <http://www-106.ibm.com/developerworks/library/ws-coor/>

[WS-T] - Web Services Transaction (WS-Transaction), <http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/>

[WS-CAF] - Web Services Composite Application Framework, <http://developers.sun.com/techttopics/webservices/wscaf/>

State of the Art on Service Composition

Service Composition

1. Composition Synthesis:

Input:

- client request
- set of available services

Output:

- specification of composite service

2. Orchestration:

Input:

- specification of composite service

Output:

- coordination of available services according to the composition schema
- data flow and control flow monitoring

How to model client request ?

How to model available services ?

How to model the composite service ?

How to orchestrate the composite service ?

1. Basics

- Services export a **view of their behavior**
 - I/O interface
 - Data Query
 - focus on data
 - for information gathering
 - Atomic Actions
 - focus on actions
 - world altering services
 - (complex) Behavioral Descriptions
 - using finite state representation formalisms

**information-
based services**

**services as
atomic actions**

**services as
processes**

1. *Relevant Works*

- Papazoglou & his group
- Bouguettaya & his group
- Composition as Planning
- Traverso & his group
- McIlraith & her group
- Hull & his group
- Knoblock & his group
- The Roman group
- Other approaches proposing interesting conceptual models for services, not targeted towards composition:
 - Vianu & his group
 - Benatallah and Casati & their group

Papazoglou & his group

[Yang & Papazoglou ISO4]

- available services: **I/O interfaces**
 - service component: simple or complex pre-existing service wrapped into a web component
 - they are stored in a service component class library
 - operations offered through a uniform interface
- composite service: **behavioral description**
 - set of service components (from service component class library) "glued" together by composition logics
 - composition logics defines execution order (either sequential or concurrent) of service components within composition, dependencies among input and output parameters, etc.
 - support for **manual** composition: designer specifies composite service using the Service Scheduling Language and the Service Composition Execution Language

Bouguettaya & his group

[Medjahed etal VLDBJ03]

- available services: **atomic actions**
 - semantically described in terms of their I/O interfaces and non-functional properties such as their purpose, their category and their quality
 - Available services stored into an ontology on the basis of their non-functional properties

Bouguettaya & his group

(cont'd) [Medjahed etal VLDBJ03]

- client request:
 - expressed in the Composite Service Specification Language (CSSL): it specifies the sequence of desired operations that the composite service should perform and control flow between operations
- **service composition problem:**
 - **Input:** (i) I/O descr. of available services
(ii) client request expr. in CSSL
 - **Output:** composite service as sequence of operations (**semi-automatically**) **obtained** from the client specification by identifying, for each operation, the operation(s) of available services that matches it, on the basis of their I/O interface and non-functional features

Composition as Planning

- available services: atomic actions
- client request: client (propositional) goal
- **service composition problem**: planning problem
 - Input:
 - (i) client goal (also encodes initial condition)
 - (ii) available services as atomic actions
 - Output: composite service as a (possibly conditional) plan, i.e., sequence of actions that transform the initial state into a state satisfying the goal.
 - *Sirin, Parsia, Wu, Hendler & Nau [Sirin etal ICWS03]*
 - ICAPS 2003 Planning for Web Services workshop [P4WS03]
 - ICAPS 2004 Planning for Web and Grid Services workshop [P4WGS04]
- NOTE: the client has not influence over the control flow of the composite service

Traverso & his group

[Traverso & Pistore ISWC04]

- available services:
 - non-deterministic transition systems characterized by a set of initial states and by a transition relation that defines how the execution of each action leads from one state to a set of states
- client request:
 - it specifies a main execution to follow, plus some side paths that are typically used to resolve exceptional circumstances, e.g., **Do Φ else Try Ψ**
- service composition problem: (extended) planning problem
- **NOTE:** the client has not influence over the control flow of the composite service

McIlraith & her group

[McIlraith et al IEEE01]

- both available and composite service:
behavioral description seen as procedures
invokable by clients
 - Golog procedure, atomically executed, i.e., seen by its client as an atomic Situation Calculus action, presenting an I/O interface
 - each service stored in an OWL-S ontology

McIlraith & her group (cont'd) [McIlraith et al IEEE01]

- client request:
 - skeleton of a Golog procedure expressing also user constraints and preferences
- **service composition problem:**
 - **Input:** (i) OWL-S ontology of services as atomic actions, and (ii) client request
 - **Output:** Golog procedure obtained by **automatically** instantiating the client request with services contained in the ontology, by also taking user preferences and constraints into account
- NOTE: the client has not influence over the control flow of the composite service

Hull & his group

[Bultan et al WWW03]

- both available and composite service (peer):
behavioral description
 - Mealy machine, that exchanges messages with other peers according to a predefined communication topology (channels among peers)
 - peers equipped with (bounded) queue to store messages received but not yet processed
 - Conversation: sequence of messages exchanged by peers
 - At each step, a peer can either (i) send a message, or (ii) receive a message, or (iii) consume a message from the queue, or (iv) perform an empty move, by just changing state

Hull & his group (cont'd)

[Bultan et al WWW03]

- **Choreography mapping problem:**
 - **Input:** (i) a desired global behavior (i.e., set of desired conversations) as a Linear Temporal Logic formula, and (ii) an infrastructure (a set of channels, a set of peer names and a set of messages)
 - **Output:** Mealy machines (**automatically obtained**) for all the peers such that their conversations are compliant with the LTL specification
- **NOTE:** not yet a "jam session style" choreography

Knoblock & his group

[Michalowski et al IEEE04]

- available service: **data query**
 - basic idea: informative services as views over data sources
 - each service described in terms of I/O parameters (of course, the latter being provided by the data source), binding patterns and additional constraints on the source
- client request:
 - data query, expressed in terms of inputs provided by the user and requested outputs

Knoblock & his group

(cont'd) [Michalowski et al IEEE04]

- **service composition problem:**
 - **Input:** (i) available services modeled as data-sources, and (ii) client request as user query
 - **Output:** (automatically obtained) composite service as integration plan for a *generalized* user query, so that all the user queries that differ only for intensional input values can be answered by the same (composite) service. Integration plan as a sequence of source queries, taking binding pattern into account

The Roman group

[Berardi et al ICSOC03]

- available service: **behavioral description**
 - service as an interactive program: at each step it presents the client with a set of actions among which to choose the next one to be executed
 - client choice depends on outcome of previously executed actions, but the rationale behind this choice depends entirely on the client
 - behavior modeled by a finite state transition system, each transition being labeled by a deterministic (atomic) action, seen as the abstraction of the effective input/output messages and operations offered by the service

The Roman group

[Berardi et al ICSOC03]

- client request (target service):
 - set of executions organized in a (finite state) *transition system* of the activities he is interested in doing
- **service composition problem:**
 - **Input:** (i) finite state transition system of available services, and (ii) finite state transition system of target service
 - **Output:** (automatically obtained) composite service that realizes the client request, such that each action of the target service is delegated to at least one available service, in accordance with the behavior of such service.
- NOTE: the client "strongly" influence the composite service control flow

Vianu & his group

[Deutsch etal PODS04]

- available service: data query + behavioral descr.
 - service as a data-driven entity characterized by a database and a tree of web pages
 - At each step, set of input choices presented to client: some generated as queries over the database; specific client data treated as constants. The client chooses one of such inputs, and in response, the service produces as output updates over the service database and/or performs some actions, and makes a transition from a web page to another
- automatic verification of service properties:
 - both over runs (linear setting) and over sets of runs (branching setting)
 - they characterize the complexity of verifying such properties for various classes of services

Benatallah and Casati & their group [Benatallah et al IEEE04]

- available service: **behavioral description**
 - behavior of a service as finite state transition system in terms of message exchanged with the clients (conversations)
 - transitions labeled by messages, and states labeled with the status of the conversation (e.g., effect of the message exchange leading to it, if clearly defined)
- they study how to automatically generate the skeleton of a BPEL4WS spec. starting from the transition system modeling the service behavior
- they also study properties of service behavior in order for two services to correctly interact

References

- [Yang & Papazoglou IS04] - J. Yang and M.P. Papazoglou: Service Components for Managing the Life-cycle of Service Compositions, Information Systems 29 (2004), no. 2, 97 - 125
- [Medjahed etal VLDBJ03] - B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid: Composing Web services on the Semantic Web, Very Large Data Base Journal 12 (2003), no. 4, 333-351
- [Bultan etal WWW03] - T. Bultan, X. Fu, R. Hull, and J. Su: Conversation Specification: A New Approach to Design and Analysis of E-Service Composition, Proceedings of the 12th International World Wide Web Conference (WWW 2003), ACM, 2003, pp. 403-410
- [McIlraith etal IEEE01] - S. McIlraith, T.C. Son, and H. Zeng: Semantic Web Services, IEEE Intelligent Systems 16 (2001), no. 2, pp.46 - 53
- [Michalowski etal IEEE04] M. Michalowski, J.L. Ambite, S. Thakkar, R. Tuchinda, C.A. Knoblock, and S. Minton: Retrieving and semantically integrating heterogeneous data from the web. IEEE Intelligent Systems, 19 (2004), no. 3, pp.72 - 79

References

- [Sirin et al ICWS03] - D. Wu, B. Parsia, E. Sirin, J.A. Hendler, D.S. Nau: Automating DAML-S Web Services Composition Using SHOP2. In Proceedings of the International Semantic Web Conference 2003, pp 195-210
- [P4WS03] - Proceedings of ICAPS 2003 Planning for Web Services workshop
- [P4WGS04] - Proceedings of ICAPS 2004 Planning for Web and Grid Services workshop
- [Traverso & Pistore ISWC04] - P. Traverso, M. Pistore: Automated Composition of Semantic Web Services into Executable Processes. In Proceedings of the Third International Semantic Web Conference (ISWC2004). To appear
- [Berardi et al ICSOC03] - D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella: Automatic Composition of e-Services that Export their Behavior, Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003), LNCS, vol. 2910
- [Deutsch et al PODS04] - A. Deutsch, L. Sui, and V. Vianu: Specification and Verification of Data-driven Web Services, In Proceedings of the 23rd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2004), ACM, 2004, pp. 71-82.
- [Benatallah et al IEEE04] - B. Benatallah, F. Casati, and F. Toumani: Web services conversation modeling: The Cornerstone for E-Business Automation. IEEE Internet Computing, 8 (2004), no. 1, pp.46 - 54

Other References

- D. Berardi *Automatic Service Composition. Models, Techniques and Tools*. Ph.D. thesis, Dipartimento di Informatica e Sistemistica - Università di Roma "La Sapienza", Rome, Italy, 2005.
- D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, *ESC: A Tool for Automatic Composition of e-Services based on Logics of Programs*, Post-proceedings of the 5th VLDB International Workshop on Technologies for e-Services (VLDB-TE04), 2004, To appear.
- D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, *Synthesis of Underspecified Composite e-Services based on Automated Reasoning*. In Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004). To appear.
- W.J. van den Heuvel, J. Yang, and M.P. Papazoglou, *Service Representation, Discovery and Composition for e-Marketplaces*, Proceedings of the 9th International Conference on Cooperative Information Systems (CoopIS 2001), LNCS, vol. 2172, Springer, 2001, pp. 270 - 284.
- R. Hull, M. Benedikt, V. Christophides, and J. Su, *E-Services: A Look Behind the Curtain*, Proceedings of the 22nd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2003), ACM, 2003, pp. 1-14.
- S. McIlraith and T. Son, *Adapting Golog for Composition of Semantic Web Services*, Proceedings of the 8th International Conferences on Principles of Knowledge Representation and Reasoning (KR 2002), Morgan Kaufmann, 2002, pp. 482 - 493.

2. Orchestration

- Two main kinds of orchestration [Hull et al PODS03]:
 - (i) the mediated approach, based on a hub-and-spoke topology, in which one service is given the role of process mediator/delegator, and all the interactions pass through such a service, and
 - (ii) the peer-to-peer approach, in which there is no centralized control

Mediated Orchestration Engines

- *e-Flow [Casati & Shan, IS01]:*
 - Platform for specifying, enacting and monitoring composite service
 - Composite *E-Service* (CES) is a service process engine offered as (meta-) service that performs coordination of services, with some process adaption/evolution mechanisms
 - A provider can offer a value added service as coordination of different services: it registers the new service to the CES and let the CES enact its execution
- *AZTEC [Christophides etal TES01]:*
 - Framework for orchestration of session-oriented, long running telecommunication services is studied. It is based on active flowcharts thus coping with asynchronous events that can happen during active telecom sessions

Mediated Orchestration Engines

- *WISE [Lazcano etal CSSE2000]:*
 - Orchestration engine that coordinates the execution of distributed applications (virtual processes), and a set of brokers enables the interaction with already existing systems that are to be used as building blocks.
 - Process meta-model based on Petri Nets, with the possibility to add Event-Condition-Action (ECA) rules
- *MENTOR-lite [Shegalov etal VLDBJ01]:*
 - Workow management system based on a XML mediator for coordinating services which are distributed among different organizations and deployed on heterogeneous platforms
 - Process meta-model is based on a specific statechart dialect

Peer-to-Peer Orchestration Engines

- *Self-Serv [Benatallah et al IEEE03]:*
 - Platform for composing services and executing new composed services in a decentralized way, through peer-to-peer interactions
 - Composite service modeled as an activity diagram
 - Its enactment carried out through the coordination of different state coordinators (one for each service involved in the specification and one for the composite service itself)
- *PARIDE Orchestrator [Mecella et al VLDB-TES02]:*
 - A composition schema, modeled as a specific Coloured Petri Net, is orchestrated by a set of organizations, which moves it (as a "token") along the execution
 - Separation between the responsibility of the orchestration and the providing of services (suitable in specific scenarios)
 - Services can be substituted with other compatibles

References

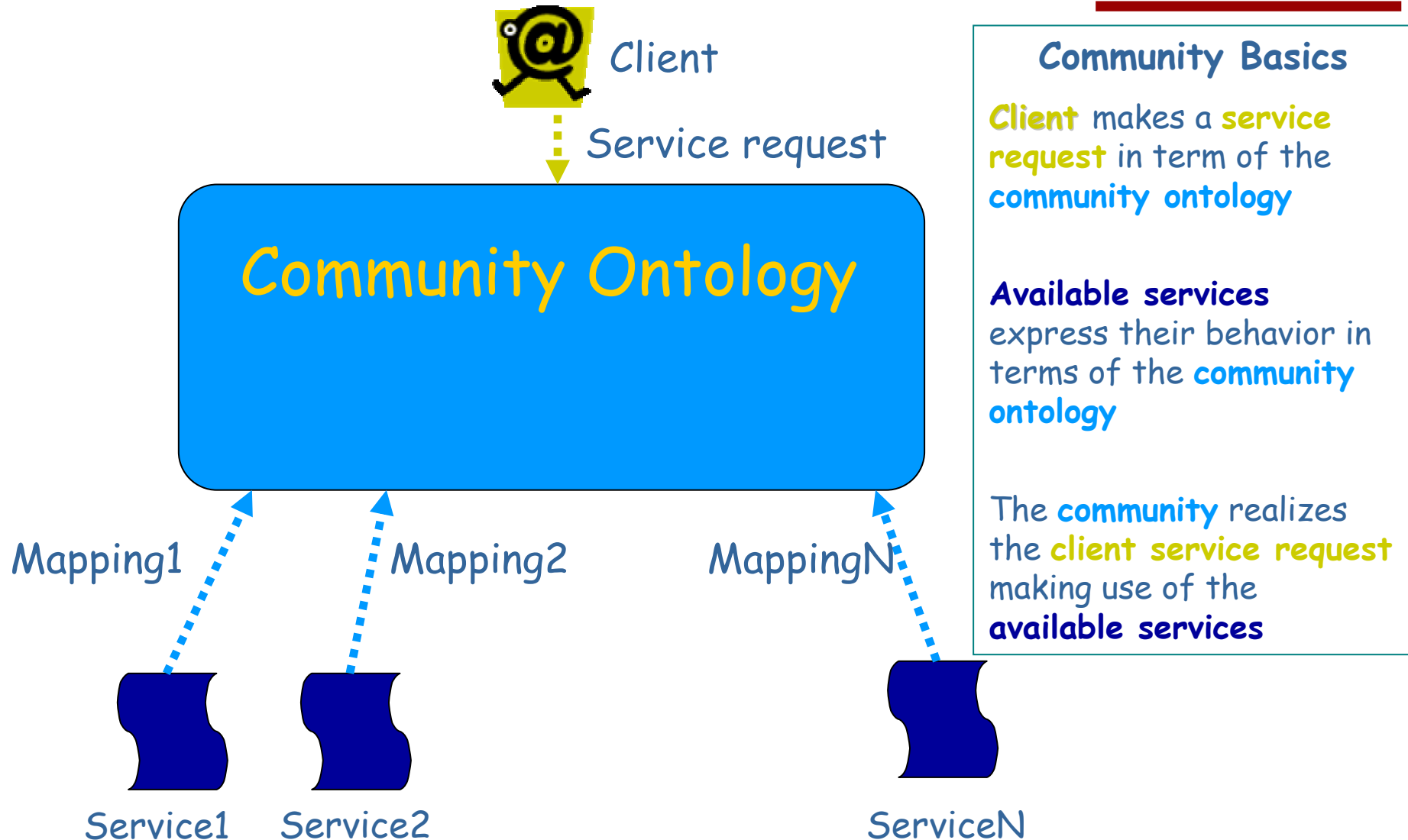
- [Casati & Shan, IS01] - F. Casati and M.C. Shan, Dynamic and Adaptive Composition of e-Services, *Information Systems* 6 (2001), no. 3, 143 - 163.
- [Christophides et al TES01] - V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong. Beyond Discrete e-Services: Composing Session-oriented Services in Telecommunications. In *Proc. of VLDB-TES, 2001*.
- [Lazcano et al CSSE2000] - A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler, The WISE approach to Electronic Commerce, *International Journal of Computer Systems Science & Engineering* 15 (2000), no. 5
- [Shegalov et al VLDBJ01] - G. Shegalov, M. Gillmann, and G. Weikum, XML-enabled Workflow Management for e- Services across Heterogeneous Platforms, *Very Large Data Base Journal* 10 (2001), no. 1, 91-103.
- [Benatallah et al IEEE03] - B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40-48, 2003
- [Mecella et al VLDB-TES02] - M. Mecella, F. Parisi Presicce, B. Pernici: Modeling e-Service Orchestration Through Petri Nets. *Proc. VLDB-TES 2002, LNCS 2444*. An extended version as M. Mecella, B. Pernici: Building Flexible and Cooperative Applications Based on eServices, Technical Report 21-02, DIS Univ. Roma "La Sapienza", 2002

Automatic Service Composition: A Conceptual Perspective

Basic Approaches to Composition

1. Information Oriented Services
2. Services as Atomic Actions
3. Services as Processes

Service Community Basics



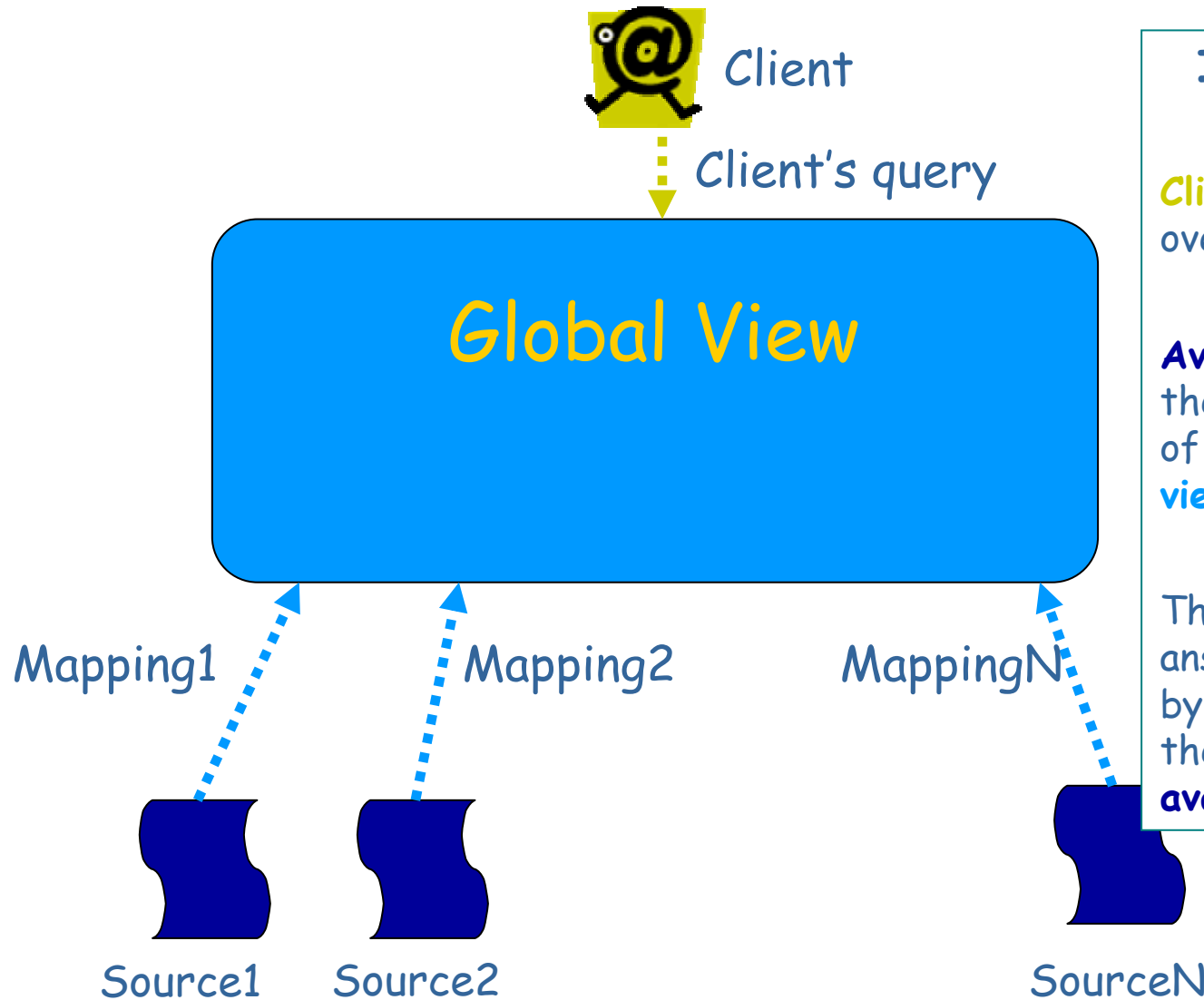
1. Information Oriented Services



1. Information Oriented Services



Data Integration System



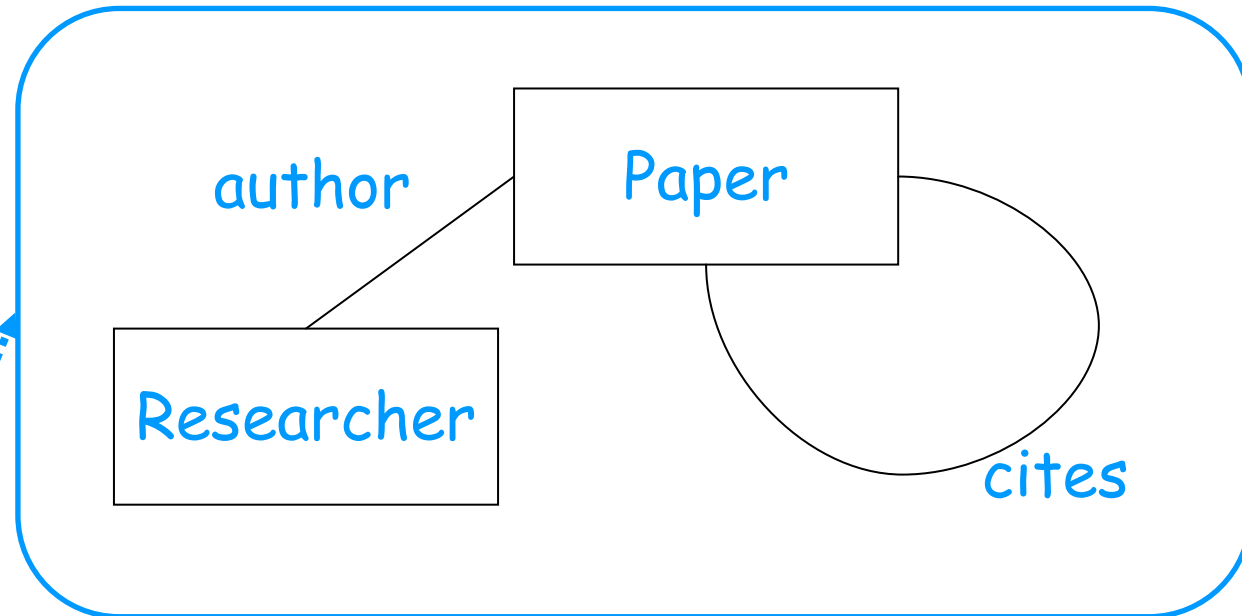
Integration System Basics

Client's request: **query** over the **global view**

Available sources express their information in terms of a **query** over the **global view**

The **integration system** answers the **client's query** by **rewriting** it in terms of the information in the **available sources**

Example



$$\text{Selfcitation}(x) \rightsquigarrow \exists z, y. \text{cite}(x,y) \wedge \text{author}(z,x) \wedge \text{author}(z,y)$$

Selfcitation

Selfcitation: contains papers that cite (other) papers by the same authors

...

GAV and LAV Mappings in Data Integration

- In **data integration**, we can distinguish two approaches in defining the mapping:
 - **GAV** (Global As View): terms of the Global View are mapped to queries over the sources
 - **LAV** (Local As View): sources are described by mapping them to a query over the Global View (cf. previous example)

GAV and LAV Mappings in Service Communities

- Following data integration, we can distinguish two approaches in defining the mapping in our **service community**:
 - **GAV** (Global As View): terms of the Community Ontology are mapped to complex expressions involving services
 - **LAV** (Local As View): services are described by mapping them to complex expressions over the Community Ontology

GAV vs LAV

- **GAV:**
 - Adopted in early Data Integration Systems
 - Typical setting
 - Sources are relational DBs
 - Global View is a relational schema
 - Mapping associate relations in the global view with a relational query over the sources
 - **Query Answering** is performed by
 - "unfolding" (substituting) each relation in the client's query with the corresponding query over the sources (the mapping),
c.f., computing the composition
 - the evaluating the resulting query
c.f., executing the composition
 - If constrains are present in the Global View, QA becomes more involved

GAV vs LAV

- LAV:
 - Recent research on data integration favors the LAV approach over the GAV approach
 - **Better support of dynamic changes in the system:** sources (services) can be added and deleted without restructuring the global view (community ontology), and hence without impacting the clients
 - **Query Answering** is a challenge because it needs to deal with incomplete information
 - QA is conceptually performed by
 - first, "**rewriting**" the client query to an "equivalent" query over the sources,
c.f., computing the composition
 - then, evaluating the resulting query
c.f., executing the composition
 - Often rewriting is not obvious and/or may require query languages that are different from the one used by the client and the mappings

Impact on Service Composition

- Work in data integration as a **direct impact on information-based service composition systems**
 - Techniques developed there can be often used off-the-shelf or with minor adaptation for information-based services
- More generally data integration research has deeply looked at systems that share many conceptual notions with service composition systems:
 - **Insights** in data integration systems can be applied to service composition systems
 - Examples:
 - The distinction between **GAV** and **LAV**
 - The distinction between **query evaluation** and **query rewriting** (**execution time vs. composition time**)
- However Data Integration has **not** looked at the **procedural aspects** typical of services (except for binding patterns)

References

- **Read and exploit data integration literature!**

- Survey on data integration**

- [Halevy VLDBJ01] A. Y. Halevy: Answering queries using views: A survey. VLDB J. 10(4): 270-294 (2001)

- [Lenzerini PODS02] M. Lenzerini: Data Integration: A Theoretical Perspective. PODS 2002: 233-246

- [Ullman ICDT97] J. D. Ullman: Information Integration Using Logical Views. ICDT 1997: 19-40

- Seminal papers**

- [Levy etal PODS05] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava: Answering Queries Using Views. PODS 1995: 95-104

- [Abiteboul etal PODS98] S. Abiteboul, O. M. Duschka: Complexity of Answering Queries Using Materialized Views. PODS 1998: 254-263

- [Duschka etal PODS97] O. M. Duschka, M. R. Genesereth: Answering Recursive Queries Using Views. PODS 1997: 109-116

- [Calvanese etal JCSS02] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Y. Vardi: Rewriting of Regular Expressions and Regular Path Queries. J. Comput. Syst. Sci. 64(3): 443-465 (2002)

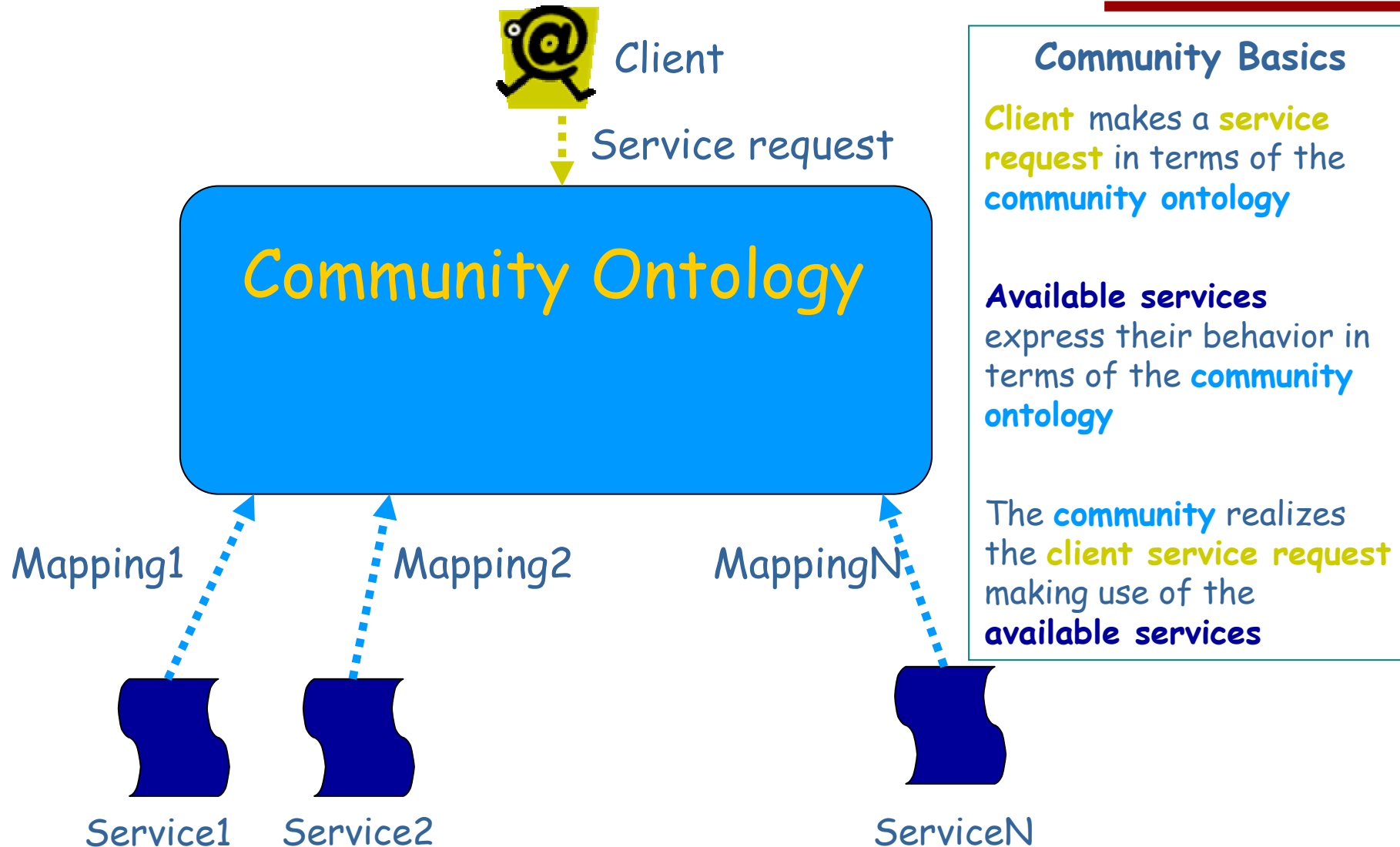
- [Rajaraman etal PODS95] A. Rajaraman, Y. Sagiv, J. D. Ullman: Answering Queries Using Templates with Binding Patterns. PODS 1995: 105-112

- **See how other service-researchers have used it!**

- [Ghandeharizadeh etal ICWS03] S. Ghandeharizadeh, C. A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. L. Ambite, M. Cai, C. Chen, P. Pol, R. R. Schmidt, S. Song, S. Thakkar, R. Zhou: Proteus: A System for Dynamically Composing and Intelligently Executing Web Services. ICWS 2003: 17-21

- [Thakkar etal P4WGS] S. Thakkar, J. L. Ambite, C. A. Knoblock: A Data Integration Approach to Automatically Composing and Optimizing Web Services. P4WGS -ICAPS WS 2004: 86-93

2. Services as Atomic Actions



2. Services as Atomic Actions



2. *Services as Atomic Actions*

- Services:
 - Seen as atomic - only **I/O behavior modeled** (no entry points other than the start and the end of the computation)
- Community Ontology:
 - **Propositions/Formulas**: facts that are known to be true
 - **Actions**: change the truth-value of the propositions
- Mappings:
 - Services are mapped towards the Community Ontology as **operators/actions**, with **preconditions** and **postconditions**
- Client Service Request:
 - **Constraints on the sequence of operators/actions to be performed**, which are expressed in terms of the Community Ontology (via pre's and post's)

Example (1)

- Services + Mappings
 - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
 - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
 - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Community Ontology:
 - $\ddot{\text{TravelSettledUp}} \equiv$
 $\text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
 $\text{CommutingSettled} \equiv$
 $\text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
- Client Service Request:
 - **Find a composition of the actions (i.e., a sequence, a program using such actions as basic instructions) such that a given property is fulfilled**

Example (2)

- Services + Mappings
 - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
 - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
 - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Community Ontology:
 - \dots
 $\text{TravelSettledUp} \equiv$
 $\text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
 $\text{CommutingSettled} \equiv$
 $\text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
- Client Service Request:
 - Starting from:**
 $\neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked} \wedge \neg \text{CommutingSettled}$
 - achieve:**
 $\text{TravelSettledUp} \wedge \text{CommutingSettled}$

Example (3)

- Services + Mappings
 - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
 - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
 - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Community Ontology:
 - $\ddot{\text{TravelSettledUp}} \equiv$
 $\text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
 - $\text{CommutingSettled} \equiv$
 $\text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
- Client Service Request:
 - Starting from:**
 $\neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked} \wedge \neg \text{CommutingSettled}$
 - achieve:**
 $\text{TravelSettledUp} \wedge \text{CommutingSettled}$
- Compositions:
 - $S_1:\text{bookFlight}; S_1:\text{bookLimo}; S_2:\text{bookHotel}; S_3:\text{bookEvent}$
 - $S_3:\text{bookEvent}; S_2:\text{bookHotel}; S_1:\text{bookFlight}; S_2:\text{bookShuttle}$

Another Example (1)

- Services + Mappings
 - S_1 : Registered \rightarrow { S_1 :bookFlight} FlightBooked
 \neg Registered \rightarrow { S_1 :register} Registered
 - S_2 : True \rightarrow { S_2 :bookHotel} HotelBooked
 HotelBooked \rightarrow { S_2 :bookShuttle} ShuttleBooked
 - S_3 : True \rightarrow { S_3 :bookEvent} EventBooked
- Community Ontology:
 - $\ddot{\text{TravelSettedUp}} \equiv$
 FlightBooked \wedge HotelBooked \wedge EventBooked
- Client Service Request:
 - Starting from:
 - \neg FlightBooked \wedge \neg HotelBooked \wedge \neg EventBooked
 - achieve:
 - TravelSettedUp

Another Example (2)

Client Service Request:

- Starting from $\neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked}$ achieve: TravelSettedUp

What about Registered?

The client does not know whether he/she/it is registered or not.

The composition must resolve this at runtime:

```
if ( $\neg$ Registered) {  
    S1:register;  
}  
S1:bookFlight;  
S2:bookHotel;  
S3:bookEvent
```

Planning is a Rich Area!!!

- Sequential Planning (plans are sequences of actions)
- Conditional Planning (plans are programs with if's and while's)
- Conformant Planning (plans the work in spite of incomplete -non observable- information)
- Knowledge Producing Actions/Sensing (distinction between truth and knowledge)
- Plan Monitoring
- Interleaving Deliberation and Execution
- Form of the Goals:
 - Achieve something
 - Achieve something while keeping something else
 - Temporal goals
 - Main goal + exception handling

GAV and LAV in Planning based Approaches

- *GAV: Community Ontology built on the basis of the actual available services (actions)*
 - Difficult to express client service request
 - Easy to express services in terms of actions with Pre's and Post's expressed in the community ontology
- *LAV: Community Ontology independent of actual available services (actions)*
 - Easy to express client service request
 - It may be difficult to abstract services as actions with Pre's and Post's expressed in the Community ontology

Main Point of Planning-based Approaches

- Client Service Request specifies **sequences of actions**
- **No further control from the client** (i.e. no further choice selections) is considered
- In other words, the **control flow is completely delegated to the community**
- Notice that instead the **control flow** can be influenced by the **environment** in certain forms of planning!

References

- **Read and exploit planning and reasoning about actions literature!**

Books

Chapters on Planning and on Reasoning about Actions in any Artificial Intelligence textbook.

[GNT04] M. Ghallab, D. Nau, P. Traverso. Automated Planning: Theory and Practice. Morgan Kaufmann, 2004.

[Reiter02] R.Reiter: Knowledge in Action. MIT Press, 2002.

Interesting papers

[Levesque AAI/IAAI96] H. J. Levesque: What Is Planning in the Presence of Sensing? AAI/IAAI, Vol. 2 1996: 1139-1146

[Bacchus&Kabanza AAI/IAAI96] F. Bacchus, F. Kabanza: Planning for Temporally Extended Goals. AAI/IAAI, Vol. 2 1996: 1215-1222

[Giunchiglia&Traverso ECP99] F. Giunchiglia, P. Traverso: Planning as Model Checking. ECP 1999: 1-20

[Calvanese etal KR02] D. Calvanese, G. De Giacomo, M. Y. Vardi: Reasoning about Actions and Planning in LTL Action Theories. KR 2002: 593-602

[De Giacomo&Vardi ECP99] G. De Giacomo, M. Y. Vardi: Automata-Theoretic Approach to Planning for Temporally Extended Goals. ECP 1999: 226-238

[Bylander IJCAI91] Tom Bylander: Complexity Results for Planning. IJCAI 1991: 274-279

- **See how other service-researchers have used it!**

- Proceedings of P4WGS - ICAPS Workshop 2004
- Proceedings of P4WS - ICAPS Workshop 2003

3. Exposing Complex Behavior



3. Exposing Complex Behavior



3. *Exposing Complex Behavior*

- Services:
 - Seen as transition systems

This research is the most specific to service composition

Several approaches:

- McIlraith & her group
- Hull & his group
- Traverso & his group
- The Roman group

McIlraith & her group Approach

- Services:
 - Seen as (possibly infinite) transition systems
- Common ontology:
 - Situation Calculus Theory and service-names
- Mapping:
 - each service name in the common ontology is mapped to a service seen as a procedure in Golog/Congolog SitCalc based high level programming language these languages describe (possibly infinite transition system)
- Client Service Request
 - Golog/Congolog program having service name as atomic actions with the understanding that it specifies acceptable sequenced of actions for the client (as in planning) and not a transition system that the client want to realize (see later)

McIlraith & her group Approach

- It's a *GAV* approach:
 - if service change the client request are must be rephrased.
- The setting described is *very general* (SitCalc is a Second Order Logic)
 - It is based on *theorem proving*
- Practical realizations of the approach are based on reducing to *formula evaluation* (in Prolog) instead of theorem proving
- It synthesize compositions that essentially are supposed to run *batch* - i.e., *control flow cannot be further influenced by the client*

References

Background

- [McCarthy IFIP62] J. L. McCarthy: Towards a Mathematical Science of Computation. IFIP Congress 1962: 21-28
- [McCarthy&Hayes MI69] J. L. McCarthy and P. C. Hayes: Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence* 4, 1969
- [Reiter 2002] R. Reiter: *Knowledge in Action*. MIT Press, 2002.
- [Levesque etal JLP2000] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. B. Scherl: *GOLOG: A Logic Programming Language for Dynamic Domains*. *J. Log. Program.* 31(1-3): 59-83 (1997)
- [De Giacomo etal AIJ2000] G. De Giacomo, Y. Lespérance, H. J. Levesque: *ConGolog, a concurrent programming language based on the situation calculus*. *Artif. Intell.* 121(1-2): 109-169 (2000)
- [De Giacomo etal KR02] G. De Giacomo, Y. Lespérance, H. J. Levesque, S. Sardiña: *On the Semantics of Deliberation in IndiGolog: From Theory to Implementation*. *KR 2002*: 603-614
- [Scherl&Levesque AIJ03] R. B. Scherl, H. J. Levesque: *Knowledge, action, and the frame problem*. *Artif. Intell.* 144(1-2): 1-39 (2003)

Papers

- [McIlraith etal IEEE01] S. A. McIlraith, T. Cao Son, H. Zeng: *Semantic Web Services*. *IEEE Intelligent Systems* 16(2): 46-53 (2001)
- [Narayanan&McIlraith WWW02] S. Narayanan, S. A. McIlraith: *Simulation, verification and automated composition of web services*. *WWW 2002*:
- [McIlraith&Son KR02] S. A. McIlraith, T. Cao Son: *Adapting Golog for Composition of Semantic Web Services*. *KR 2002*: 482-496
- [Burstein etal ISWC02] M. H. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, K. P. Sycara: *DAML-S: Web Service Description for the Semantic Web*. *International Semantic Web Conference 2002*: 348-363
- [Narayanan&McIlraith CN03] S. Narayanan, Sheila A. McIlraith: *Analysis and simulation of Web services*. *Computer Networks* 42(5): 675-693 (2003)
- [McIlraith&Martin IEEE03] S. A. McIlraith, D. L. Martin: *Bringing Semantics to Web Services*. *IEEE Intelligent Systems* 18(1): 90-93 (2003)

Hull & his group Approach

*Note that this is not the work on conversations,
which, as previously discussed, is more about choreography,
but the work at ICSOC04*

- Services:
 - Seen as (finite) transition systems
- Common ontology:
 - Alphabet of atomic actions
- Mapping:
 - Service is mapped to the community ontology as a transition system using the alphabet of the community
- Client Service Request:
 - Realize a set of sequences of actions specified by a finite automata.

Hull & his group Approach

- It's a **LAV** approach
- Inspired by the Roman Approach but with **profound differences** on the kind of client requests
- Client requests are essentially **languages**: computing the composition is based on finite automata of strings
- It synthesizes compositions as labeling of sequences of actions that essentially are supposed to run **batch** - i.e., **client cannot further influence the control flow**

References

Paper analyzed

[Gerede et al ICSOC04] C. E. Gerede, R. Hull, O. H. Ibarra, J. Su:
Automated Composition of E-services: Lookaheads. ICSOC
2004

Other papers

[Hull et al PODS03] R. Hull, M. Benedikt, V. Christophides, J. Su:
E-services: a look behind the curtain. PODS 2003: 1-14

[Hull et al SIGMOD03] R. Hull, J. Su: Tools for Design of
Composite Web Services. SIGMOD Conference 2004: 958-961

[Bultan et al WWW03] T. Bultan, X. Fu, R. Hull, J. Su:
Conversation specification: a new approach to design and
analysis of e-service composition. WWW 2003: 403-410

Traverso & his group Approach

- Services:
 - Seen as (finite) transition systems
- Common ontology:
 - Atomic actions and propositions, as in Planning
- Mapping:
 - Service is mapped to the community ontology as a transition system using the alphabet of the community and defining how transitions affect the propositions
- Client Service Request:
 - try to find a sequence of actions to achieve $Goal_1$ (main computation), with guarantees that upon failure $Goal_2$ is reached (exception handling)

Traverso & his group Approach

- It's a **LAV** approach
- Has a richer ontology that allow the client to specify its requests in a declarative way
- Client request are specific formulas of mu-calculus
- Composition techniques are based on (advanced form of) model checking (see work by the authors on planning via model checking)
- Once the "plan" is synthesized, it is supposed to run **batch** - i.e., **client cannot further influence the control flow**

References

Papers on Planning as Model Checking

[Giunchiglia&Traverso ECP99] F. Giunchiglia, P. Traverso: Planning as Model Checking. ECP 1999: 1-20

[Pistore&Traverso IJCAI01] M. Pistore, P. Traverso: Planning as Model Checking for Extended Goals in Non-deterministic Domains. IJCAI 2001: 479-486

[Bertoli et al IJCAI01] P. Bertoli, A. Cimatti, M. Roveri, P. Traverso: Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. IJCAI 2001: 473-478

[Dal Lago et al AAI/IAAI02] U. Dal Lago, M. Pistore, P. Traverso: Planning with a Language for Extended Goals. AAI/IAAI 2002: 447-454

[Cimatti et al AIJ03] A. Cimatti, M. Pistore, M. Roveri, P. Traverso: Weak, strong, and strong cyclic planning via symbolic model checking. Artif. Intell. 147(1-2): 35-84 (2003)

[Bertoli et al ICAPS03] P. Bertoli, A. Cimatti, M. Pistore, P. Traverso: A Framework for Planning with Extended Goals under Partial Observability. ICAPS 2003: 215-225

Papers on Service Composition

[Pistore&Traverso ISWC04] M. Pistore, P. Traverso: Automated Composition of Semantic Web Services into Executable Processes. ISWC2004.

[Pistore et al P4WGS04] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso: Planning and Monitoring Web Service Composition. P4WGS - ICAPS WS 2004

[Pistore et al AIMS04] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso: Planning and Monitoring Web Service Composition. AIMS04: 106-115

The Roman group Approach

- Services:
 - Seen as (finite) transition systems
- Common ontology:
 - Alphabet of atomic actions
- Mapping:
 - Service is mapped to the community ontology as a transition system using the alphabet of the community
- Client Service Request:
 - Desired service behavior, i.e., a (finite) **TS** using the **common set of actions** of the community

The Roman group Approach

- It's a **LAV** approach
- It is characterized by the form of client service request:
 - **Desired service behavior, i.e., a (finite) TS using the common set of actions of the community**
- The composition, once realized, is **not** supposed to work **batch** but **the client has full control on the control flow, i.e., the composite service acts as a (virtual) service available to the client**
- Such kind of client request tell apart this approach from previous work on planning
- More later ...

References

Papers

- [Berardi et al ICSOC03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: Automatic Composition of E-services That Export Their Behavior. ICSOC 2003: 43-58
- [Berardi et al ICSOC04] D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella, D. Calvanese: Synthesis of Underspecified Composite e-Services based on Automated Reasoning. ICSOC 2004
- [Berardi et al WES03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: A Foundational Vision of e-Services. WES 2003: 28-40
- [Berardi et al P4WS03] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella: Composing e-Services by Reasoning about Actions, ICAPS 2003 Workshop on Planning for Web Services (P4WS03).
- [Berardi et al DL03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: e-Service Composition by Description Logics Based Reasoning. Description Logics 2003
- [Berardi et al P4WGS04] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: Synthesis of Composite e-Services based on Automated Reasoning. ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services (P4WGS04).
- [Berardi et al TES04] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: ESC: A Tool for Automatic Composition of e-Services based on Logics of Programs, VLDB-TES 2004

Transition Systems

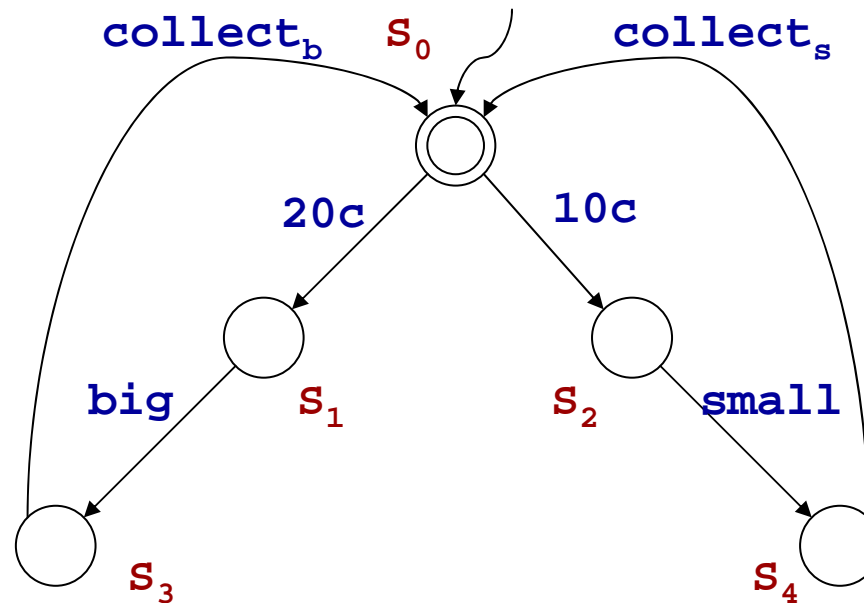
Transition Systems

- A transition system TS is a tuple $T = \langle A, S, S^0, \delta, F \rangle$ where:
 - A is the set of actions
 - S is the set of states
 - $S^0 \subseteq S$ is the set of initial states
 - $\delta \subseteq S \times A \times S$ is the transition relation
 - $F \subseteq S$ is the set of final states

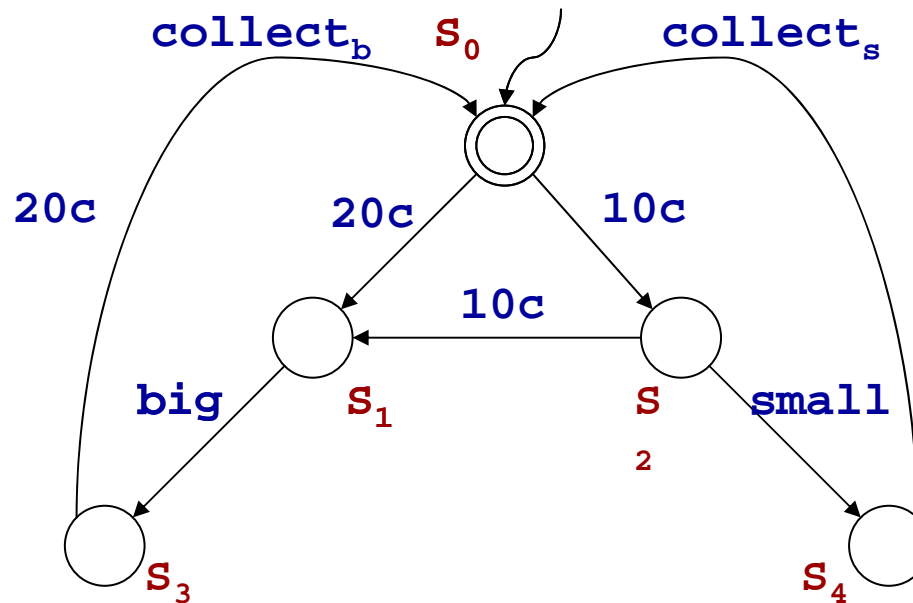
(c.f. Kripke Structure)

- Variants:
 - No initial states
 - Single initial state
 - Deterministic actions
 - States labeled by propositions other than Final/ \neg Final

Example (Vending Machine)

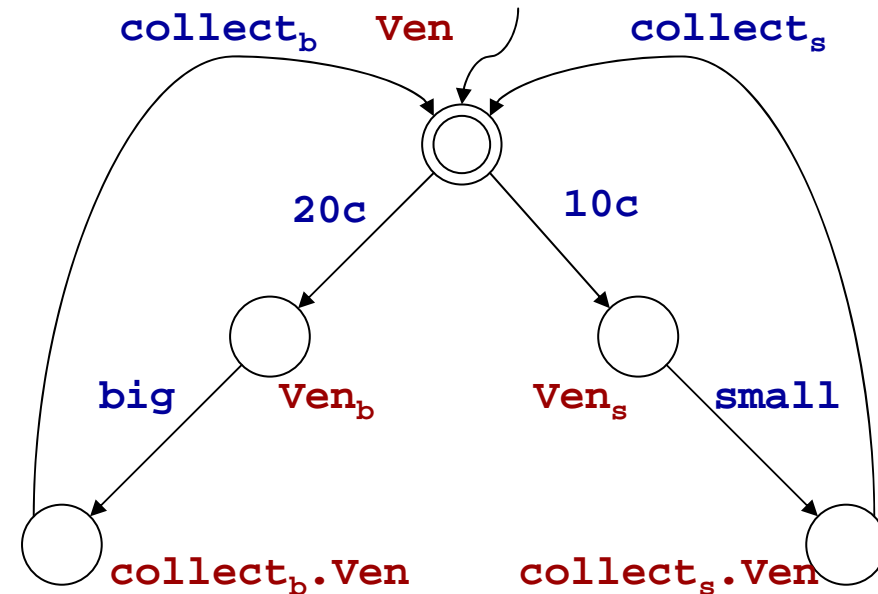


Example (Another Vending Machine)



Process Algebras are Formalisms for Describing TS

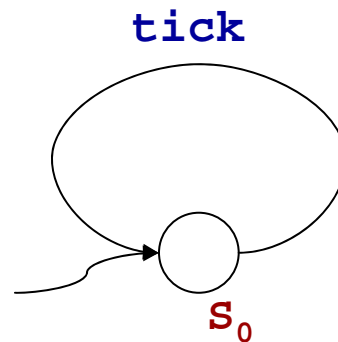
- Trans (a la CCS)
 - $Ven = 20c.Ven_b + 10c.Ven_s$
 - $Ven_b = big.collect_b.Ven$
 - $Ven_l = small.collect_s.Ven$
- Final
 - $\checkmark Ven$



- *TS may have infinite states - e.g., this happens when generated by process algebras involving iterated concurrency*
- *However we have good formal tools to deal only with finite states TS*

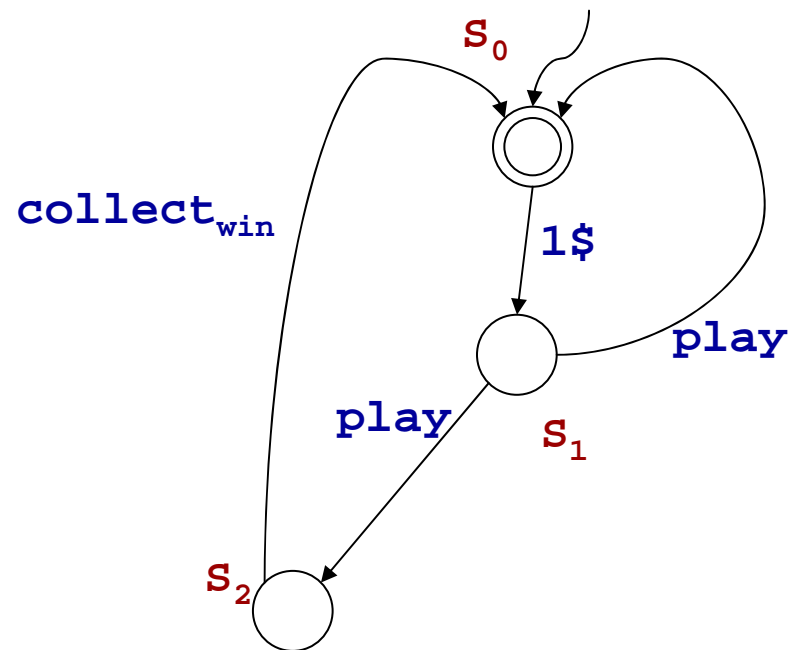
Example (Clock)

TS may describe (legal) nonterminating processes

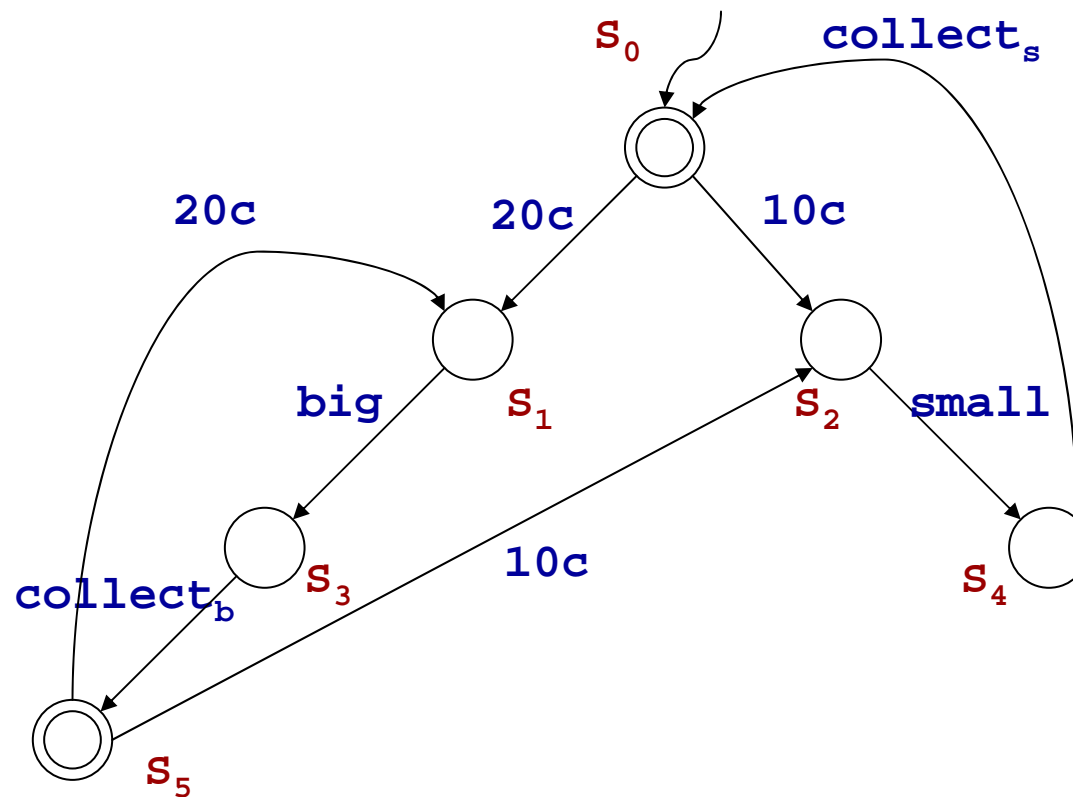


Example (Slot Machine)

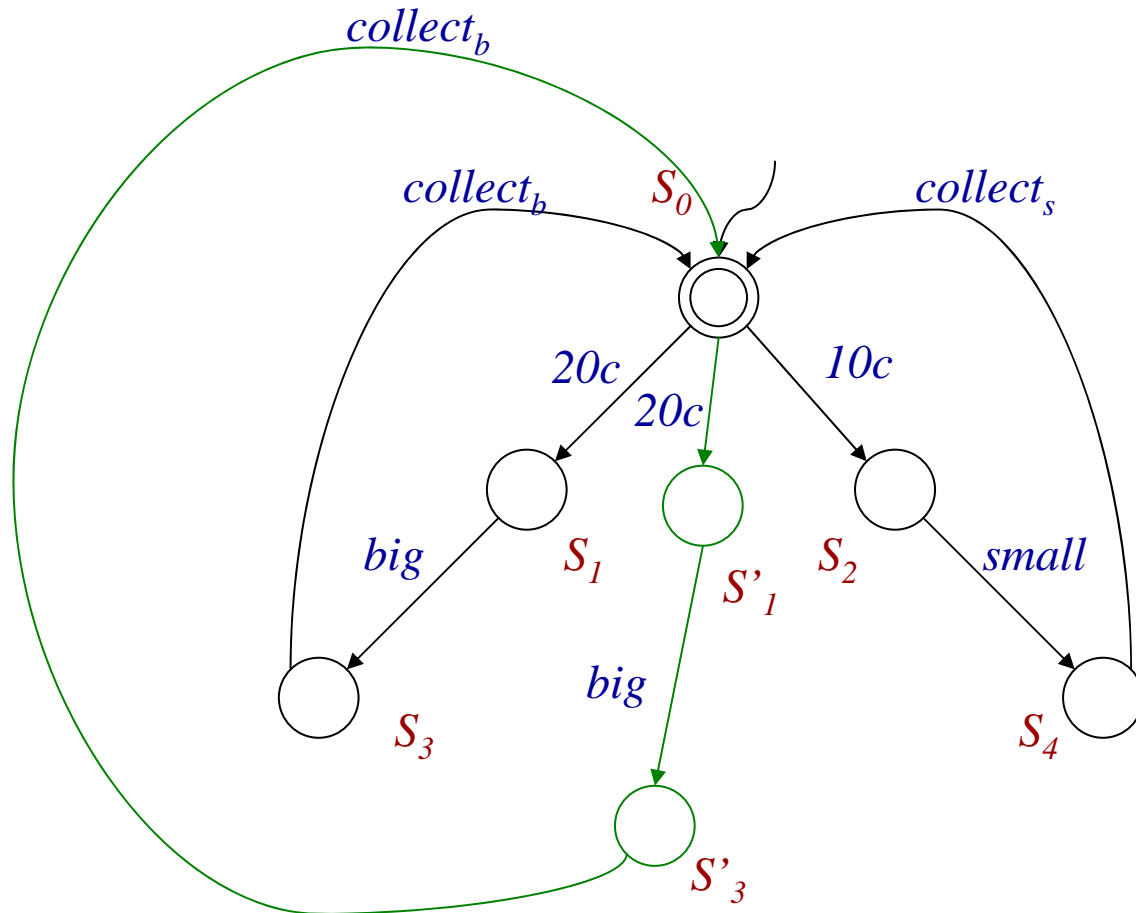
Nondereminisic transitions express
choice that is not under the control of clients



Example (Vending Machine - Variant 1)



Example (Vending Machine - Variant 2)



Bisimulation

- A binary relation R is a **bisimulation** iff:

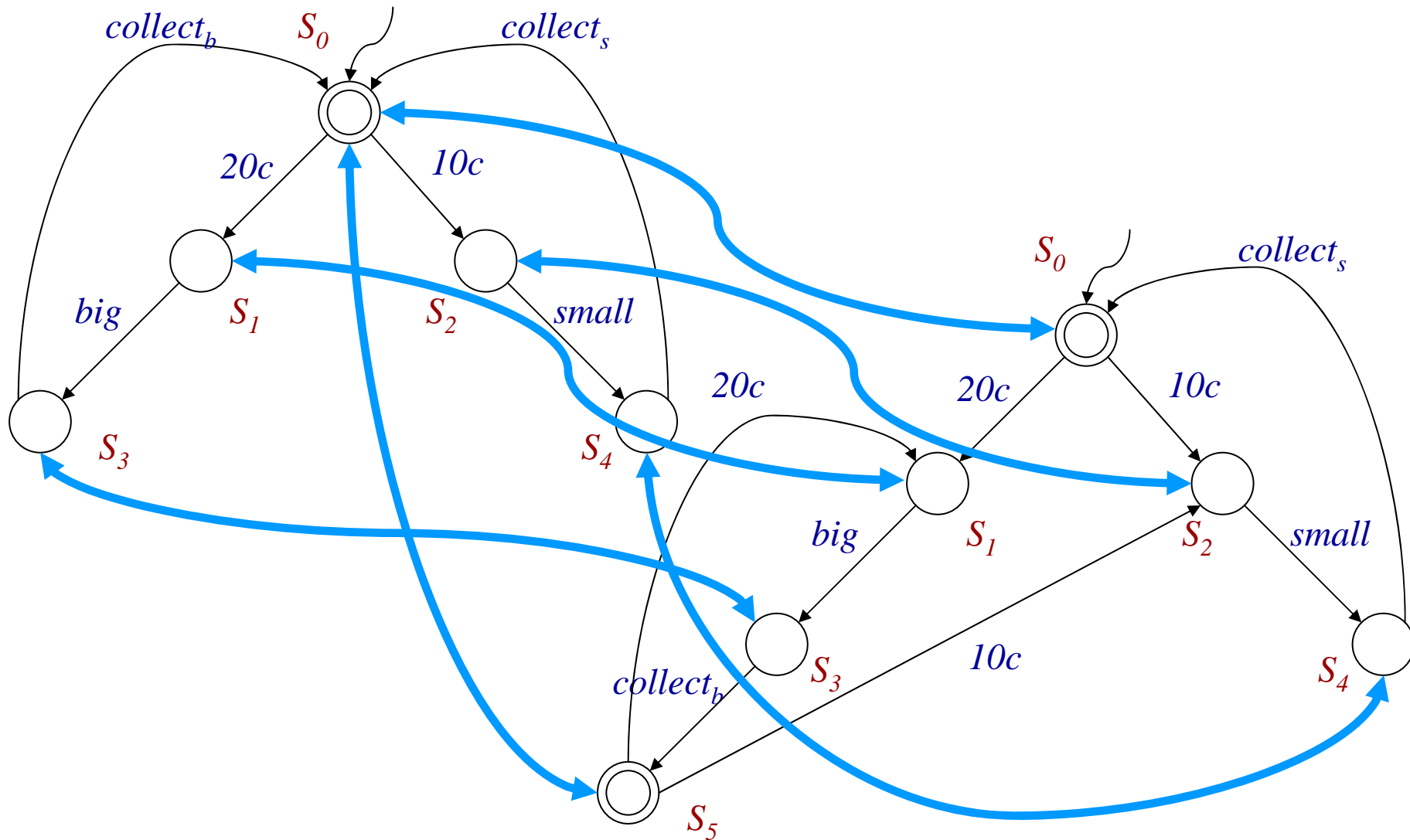
$(s,t) \in R$ implies that

- s is *final* iff t is *final*
- for all actions a
 - if $s \rightarrow_a s'$ then $\exists t' . t \rightarrow_a t'$ and $(s',t') \in R$
 - if $t \rightarrow_a t'$ then $\exists s' . s \rightarrow_a s'$ and $(s',t') \in R$

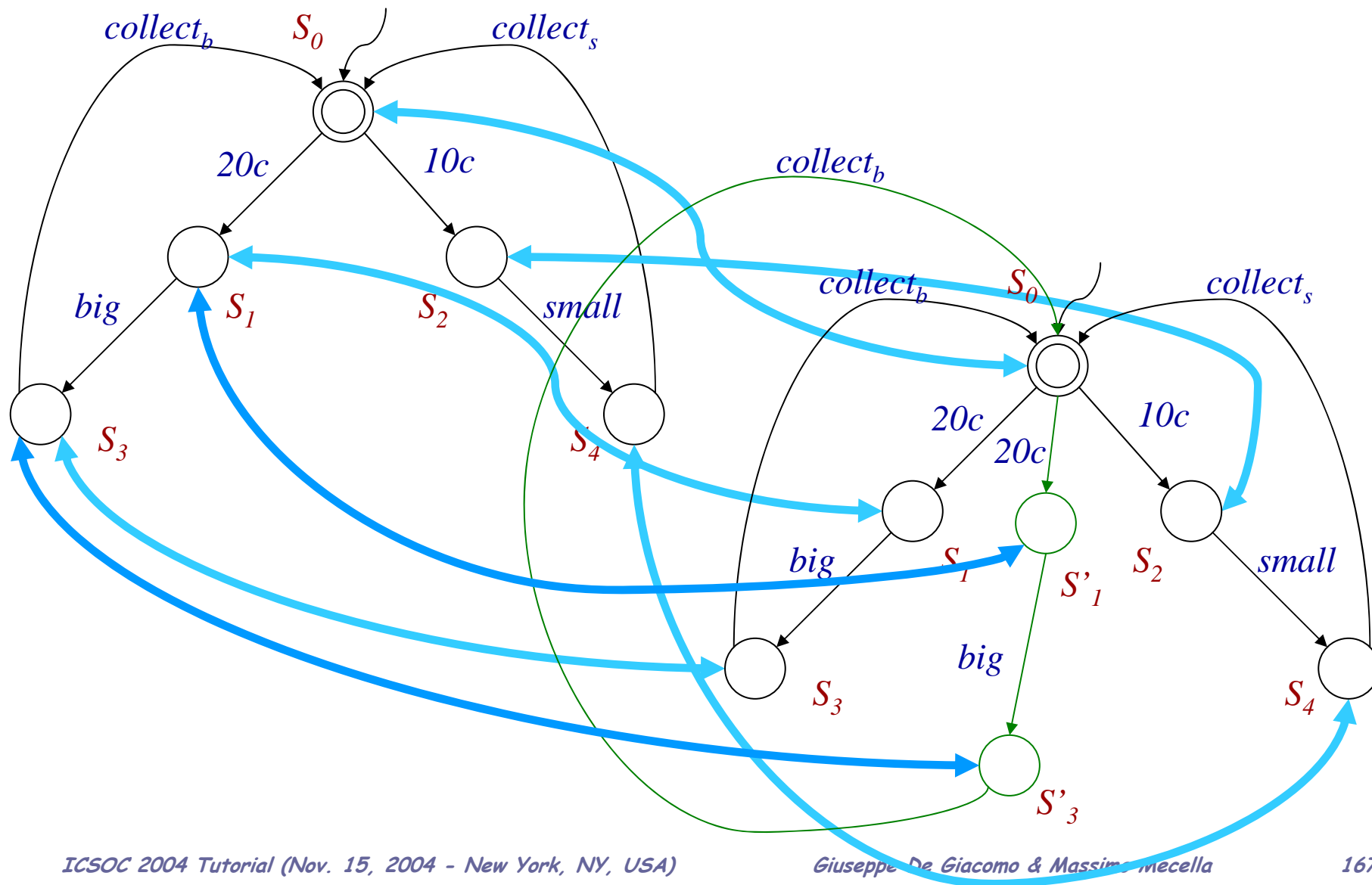
Note it is a co-inductive definition!

- A state s_0 of transition system S is **equivalent** to a state t_0 of transition system T iff there **exists** a **bisimulation** between the initial states s_0 and t_0 .

Example of Bisimulation

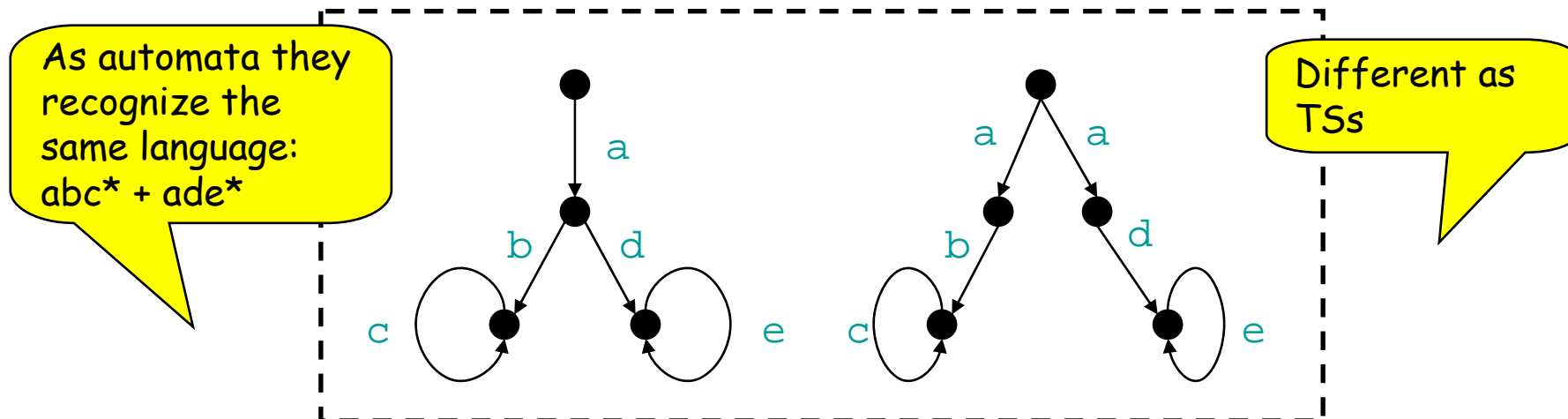


Example of Bisimulation



Automata vs. Transition Systems

- Automata
 - define sets of runs (or traces or strings): (finite) length sequences of actions
- TSs
 - ... but I can be interested also in the alternatives "encountered" during runs, as they represent client's "choice points"



Logics of Programs

- Are modal logics that allow to describe properties of transition systems
- Examples:
 - HennessyMilner Logic
 - Propositional Dynamic Logics
 - Modal (Propositional) Mu-calculus
- Perfectly suited for describing transition systems: they can tell apart transition systems modulo bisimulation

HennessyMilner Logic

- $\Phi := P$ | *(atomic propositions)*
 $\neg \Phi$ | $\Phi_1 \wedge \Phi_2$ | $\Phi_1 \vee \Phi_2$ | *(closed under boolean operators)*
 $[a]\Phi$ | $\langle a \rangle \Phi$ *(modal operators)*
- Propositions are used to denote final states
- $\langle a \rangle \Phi$ means there *exists* an a -transition that leads to a *state where Φ holds*; i.e., expresses the capability of executing action a bringing about Φ
- $[a]\Phi$ means that *all* a -transitions lead to states where Φ holds; i.e., express that executing action a brings about Φ

Logics of Programs: Examples

- Usefull abbreviation:
 - $\langle \text{any} \rangle \Phi$ stands for $\langle a_1 \rangle \Phi \vee \dots \vee \langle a_n \rangle \Phi$
 - $[\text{any}] \Phi$ stands for $[a_1] \Phi \wedge \dots \wedge [a_n] \Phi$
 - $\langle \text{any} - a_1 \rangle \Phi$ stands for $\langle a_2 \rangle \Phi \vee \dots \vee \langle a_v \rangle \Phi$
 - $[\text{any} - a_1] \Phi$ stands for $[a_2] \Phi \wedge \dots \wedge [a_v] \Phi$
- Examples:
 - $\langle a \rangle \text{true}$ *cabability of performing action a*
 - $[a] \text{false}$ *inability of performing action a*
 - $\neg \text{Final} \langle \text{any} \rangle \text{true} \wedge [\text{any} - a] \text{false}$
*necessity/inevitability of performing action a
i.e., action a is the only action possible*
 - $\neg \text{Final} \wedge [\text{any}] \text{false}$ *deadlock!*

Propositional Dynamic Logic

- $\Phi := P$ | *(atomic propositions)*
 $\neg \Phi$ | $\Phi_1 \wedge \Phi_2$ | $\Phi_1 \vee \Phi_2$ | *(closed under boolean operators)*
 $[r]\Phi$ | $\langle r \rangle \Phi$ | *(modal operators)*
- $r := a$ | $r_1 + r_2$ | $r_1; r_2$ | r^* | $P?$ | *(complex actions as regular expressions)*
- Essentially add the capability of expressing partial correctness assertions via formulas of the form
 - $\Phi_1 \rightarrow [r]\Phi_2$ | *under the conditions Φ_1 , all possible executions of r that terminate reach a state of the TS where Φ holds*
- Also add the ability of asserting that a property holds in all nodes of the transition system
 - $[(a_1 + \dots + a_n)^*]\Phi$ | *in every reachable state of the TS Φ holds*
- Useful abbreviations:
 - any stands for $(a_1 + \dots + a_n)$ | *- observe that + can be expressed in HM Logic*
 - u stands for any* | *- this is the so called master/universal modality*

Modal Mu-Calculus

- $\Phi := P$ | *(atomic propositions)*
 $\neg \Phi$ | $\Phi_1 \wedge \Phi_2$ | $\Phi_1 \vee \Phi_2$ | *(closed under boolean operators)*
 $[r]\Phi$ | $\langle r \rangle \Phi$ | *(modal operators)*
 $\mu X. \Phi(X)$ | $\nu X. \Phi(X)$ | *(fixpoint operators)*
- It is the most expressive logic of the family of logics of programs.
- It subsumes
 - PDL (modalities involving complex actions are translated into formulas involving fixpoints)
 - LTL (linear time temporal logic),
 - CTS, CTS* (branching time temporal logics)
- Examples:
- $[any^*]\Phi$ can be expressed as $\nu X. \Phi \wedge [any]X$
- $\mu X. \Phi \vee [any]X$ | *along all runs eventually Φ*
- $\mu X. \Phi \vee \langle any \rangle X$ | *along some run eventually Φ*
- $\nu X. [a](\mu Y. \langle any \rangle true \wedge [any-b]Y) \wedge X$ | *every run that contains a contains later b*

Model Checking

- Model checking is polynomial in the size of the TS for
 - HennessyMilner Logic
 - PDL
 - Mu-Calculus
- Also model checking is wrt the formula
 - Polynomial for HennessyMiner Logic
 - Polynomial for PDL
 - Polynomial for Mu-Calculus with bounded alternation of fixpoints and $NP \cap coNP$ in general

Model Checking

- Given a TS T , one of its states s , and a formula Φ verify whether the formula holds in s . Formally:

$$T, s \models \Phi$$

- Examples (TS is our vending machine):
 - $S_0 \models \text{Final}$
 - $S_0 \models \langle 10c \rangle \text{true}$ *capability of performing action 10c*
 - $S_2 \models [\text{big}] \text{false}$ *inability of performing action big*
 - $S_0 \models [10c][\text{big}] \text{false}$ *after 10c cannot execute big*
 - $S_i \models \mu X. \text{Final} \vee [\text{any}] X$ *eventually a final state is reached*
 - $S_0 \models \nu Z. (\mu X. \text{Final} \vee [\text{any}] X) \wedge [\text{any}] Z$ *or equivalently*
 $S_0 \models [\text{any}^*](\mu X. \text{Final} \vee [\text{any}] X)$ *from everywhere eventually final*

Planning as Model Checking

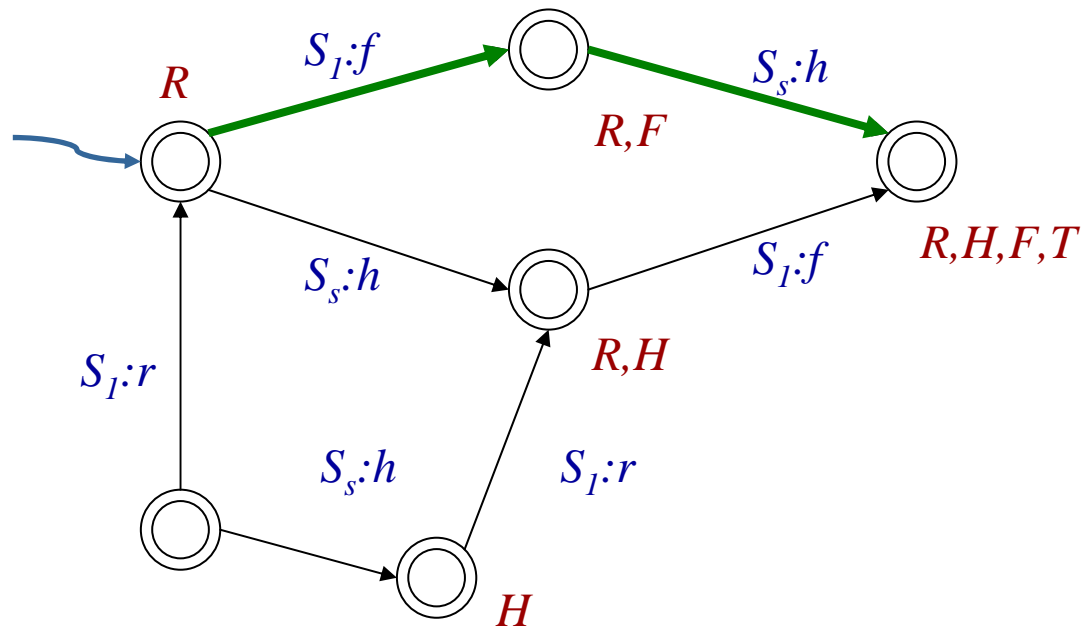
- **Build the TS of the domain:**
 - Consider the set of states formed all possible truth value of the propositions (this works only for propositional setting).
 - Use Pre's and Post of actions for determining the transitions

Note: the TS is exponential in the size of the description.
- **Write the goal in a logic of program**
 - typically a single least fixpoint formula of Mu-Calculus
- **Planning:**
 - model check the formula on the TS starting from the given initial state.
 - use the path (paths) used in the above model checking for returning the plan.
- *This basic technique works only when we have complete information (or at least total observability on state):*
 - *Sequential plans if initial state known and actions are deterministic*
 - *Conditional plans if many possible initial states and/or actions are nondeterministic*

Example

- Operators (Services + Mappings)
 - $\text{Registered} \wedge \neg \text{FlightBooked} \rightarrow [S_1:\text{bookFlight}] \text{FlightBooked}$
 - $\neg \text{Registered} \rightarrow [S_1:\text{register}] \text{Registered}$
 - $\neg \text{HotelBooked} \rightarrow [S_2:\text{bookHotel}] \text{HotelBooked}$
- Additional constraints (Community Ontology):
 - $\text{TravelSettledUp} \equiv \text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
- Goals (Client Service Requests):
 - Starting from state
 $\text{Registered} \wedge \neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked}$
check $\langle \text{any}^* \rangle \text{TravelSettledUp}$
 - Starting from all states such that
 $\neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked}$
check $\langle \text{any}^* \rangle \text{TravelSettledUp}$

Example

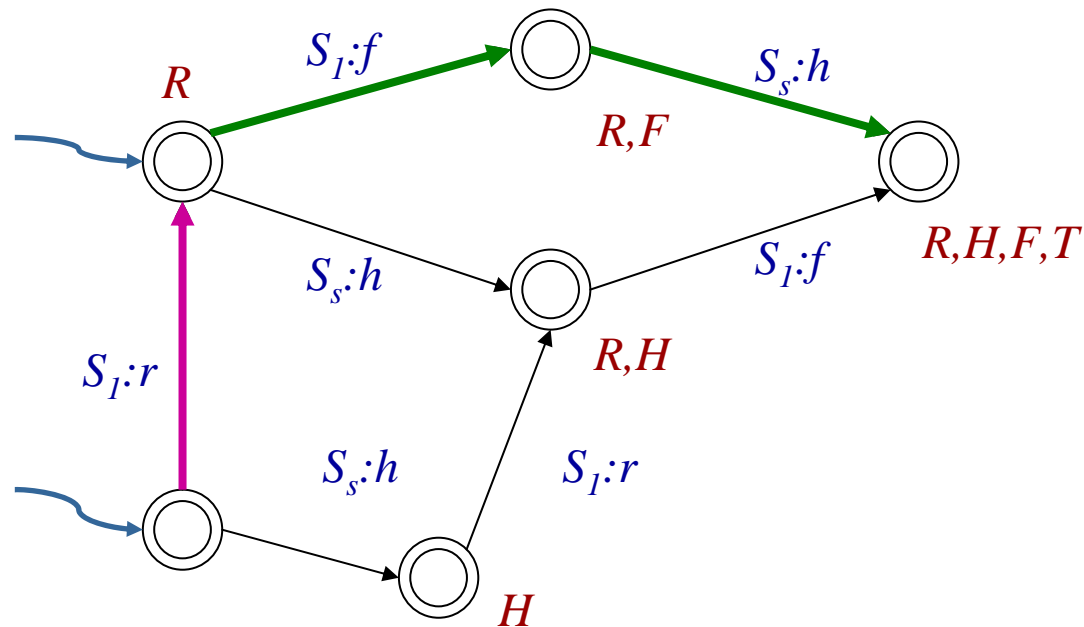


Plan:

S_1 :bookFlight;
 S_2 :bookHotel

Starting from state
 $\text{Registered} \wedge \neg \text{FlightBooked} \wedge \neg \text{HotelBooked} \wedge \neg \text{EventBooked}$
check
 $\langle \text{any}^* \rangle \text{TravelSettledUp}$

Example



Plan:

```

if( $\neg$ Registered) {
  S1:register;
}
S1:bookFlight;
S2:bookHotel
    
```

Starting from states where
 \neg FlightBooked \wedge \neg HotelBooked \wedge \neg EventBooked
 check
 \langle any* \rangle TravelSettledUp

Satisfiability

- Observe that a formula Φ may be used to select among all TS T those such that for a given state s we have that $T, s \models \Phi$
- **SATISFIABILITY**: Given a formula Φ verify whether there exists a TS T and a state s such that. Formally:

check whether exists T, s such that $T, s \models \Phi$

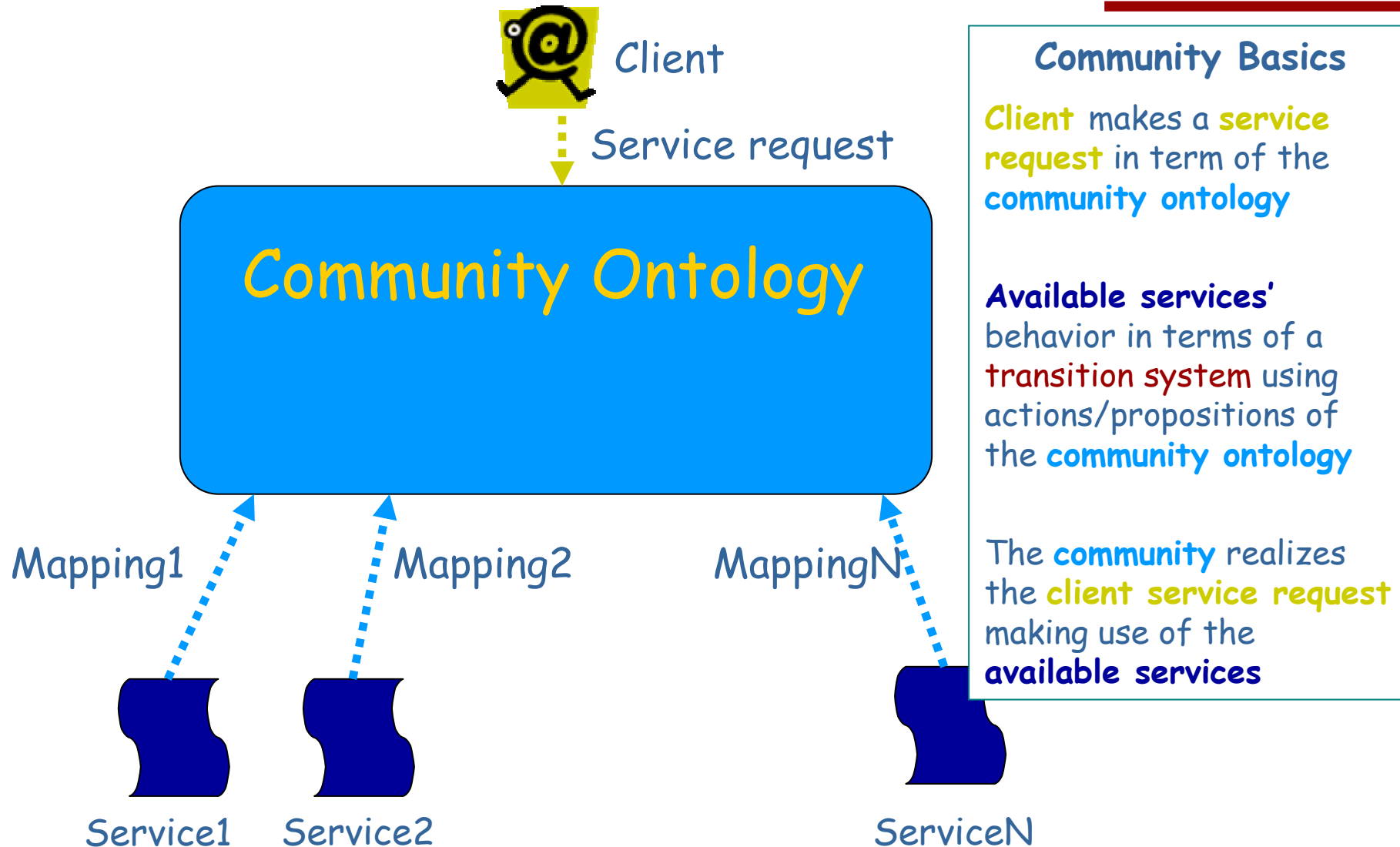
- Satisfiability is:
 - PSPACE for HennesyMilner Logic
 - EXPTIME for PDL
 - EXPTIME for Mu-Calculus

References

- [Stirling Banff96] C. Stirling: Modal and temporal logics for processes. Banff Higher Order Workshop LNCS 1043, 149-237, Springer 1996
- [Bradfield&Stirling HPA01] J. Bradfield, C. Stirling: Modal logics and mu-calculi. Handbook of Process Algebra, 293-332, Elsevier, 2001.
- [Stirling 2001] C. Stirling: Modal and Temporal Properties of Processes. Texts in Computer Science, Springer 2001
- [Kozen&Tiuryn HTCS90] D. Kozen, J. Tiuryn: Logics of programs. Handbook of Theoretical Computer Science, Vol. B, 789-840. North Holland, 1990.
- [HKT2000] D. Harel, D. Kozen, J. Tiuryn: Dynamic Logic. MIT Press, 2000.
- [Clarke& Schlingloff HAR01] E. M. Clarke, B. Schlingloff: Model Checking. Handbook of Automated Reasoning 2001: 1635-1790
- [CGP 2000] E.M. Clarke, O. Grumberg, D. Peled: Model Checking. MIT Press, 2000.
- [Emerson HTCS90] E. A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science, Vol B: 995-1072. North Holland, 1990.
- [Emerson Banff96] E. A. Emerson. Automated Temporal Reasoning about Reactive Systems. Banff Higher Order Workshop, LNCS 1043, 111-120, Springer 1996
- [Vardi CST] M. Vardi: Alternating automata and program verification. Computer Science Today - Recent Trends and Developments, LNCS Vol. 1000, Springer, 1995.
- [Vardi etal CAV94] M. Vardi, O. Kupferman and P. Wolper: An Automata-Theoretic Approach to Branching-Time Model Checking (full version of CAV'94 paper).

Composition: the Roman Way

Exposing Complex Behavior



Community of Services

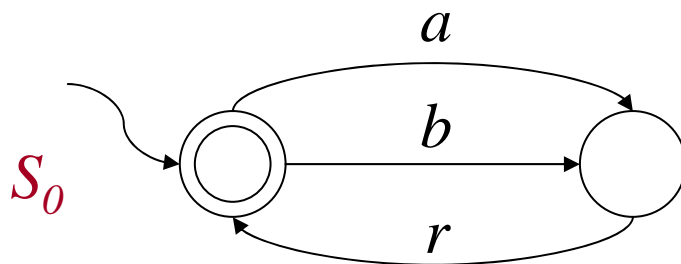
- A **community** of Services is
 - a **set** of services ...
 - ... that share implicitly a *common understanding* on a **common set of actions** (common ontology limited to the alphabet of actions)...
 - ... and export their **behavior** using (finite) **TS** over this **common set of actions**
- A **client** specifies needs as a service behavior, i.e, a (finite) **TS** using the **common set of actions** of the community

Service Behavior

- We model *services* as finite TS $T = (\Sigma, S, s^0, \delta, F)$ with
 - **single initial state** (s^0)
 - **deterministic transitions** (i.e., δ is a partial function from $S \times \Sigma$ to S)

Note: In this way the client entirely controls/chooses the transition to execute

Example:



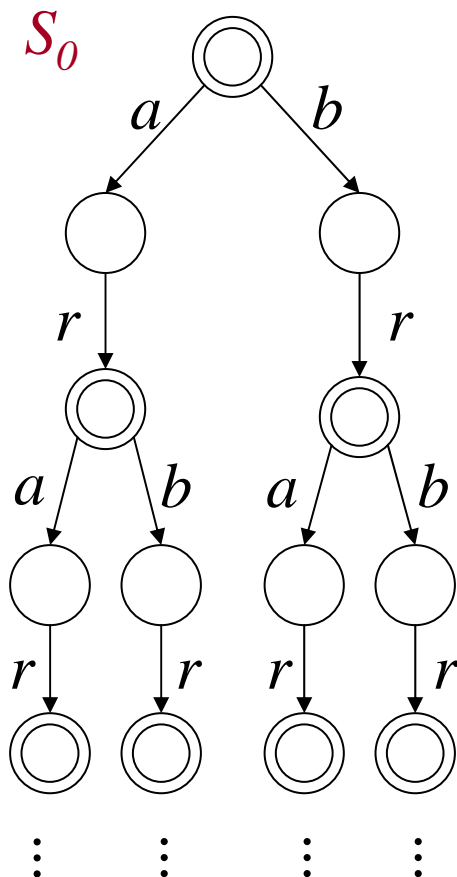
a: "search by author (and select)"

b: "search by title (and select)"

r: "listen (the selected song)"

Execution Tree

By "unfolding" a (finite) TS one gets an (infinite) **execution tree**



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

Formalizing Service Composition

Composition:

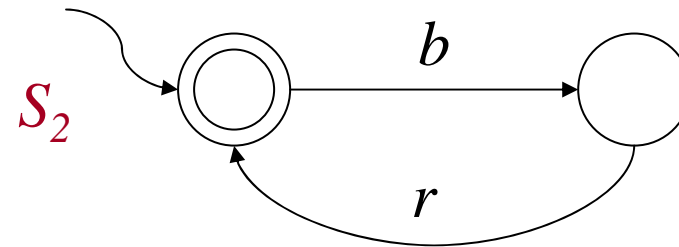
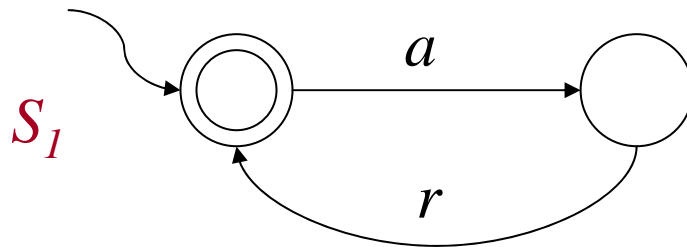
- coordinating program ...
- ... that realizes the target service ...
- ... by suitably coordinating available services

⇒ Composition can be formalized as:

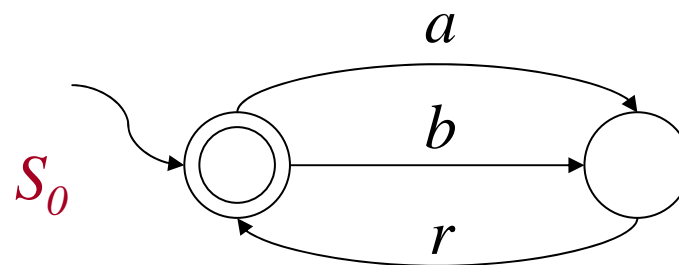
- a labeling of the execution tree of the target service such that ...
- ... each action in the execution tree is labeled by the community service that executes it ...
- ... and each possible sequence of actions on the target service execution tree corresponds to possible sequences of actions on the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

Example of Composition (1)

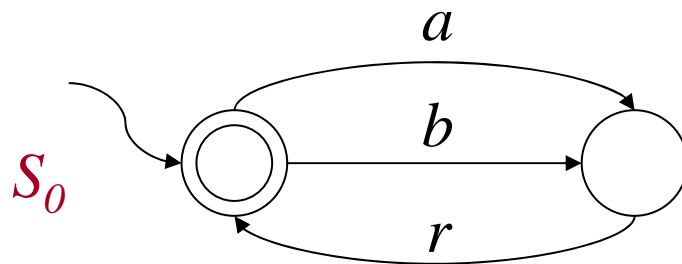
- Community services



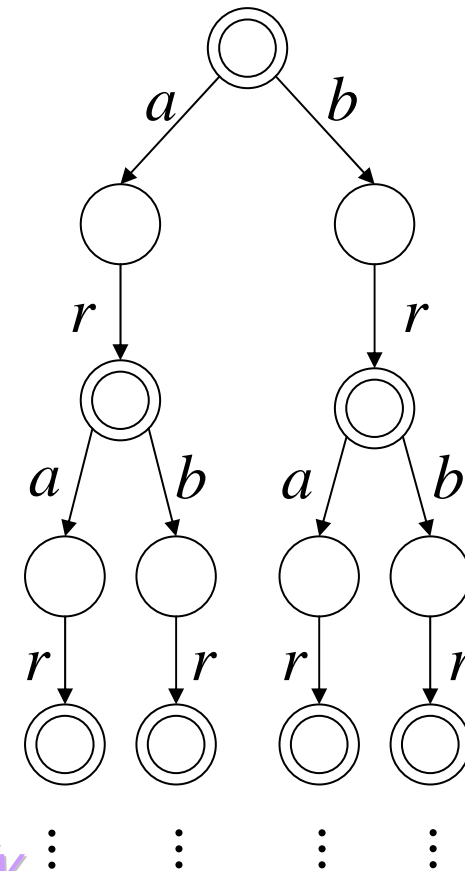
- Target service



Example of Composition (2)



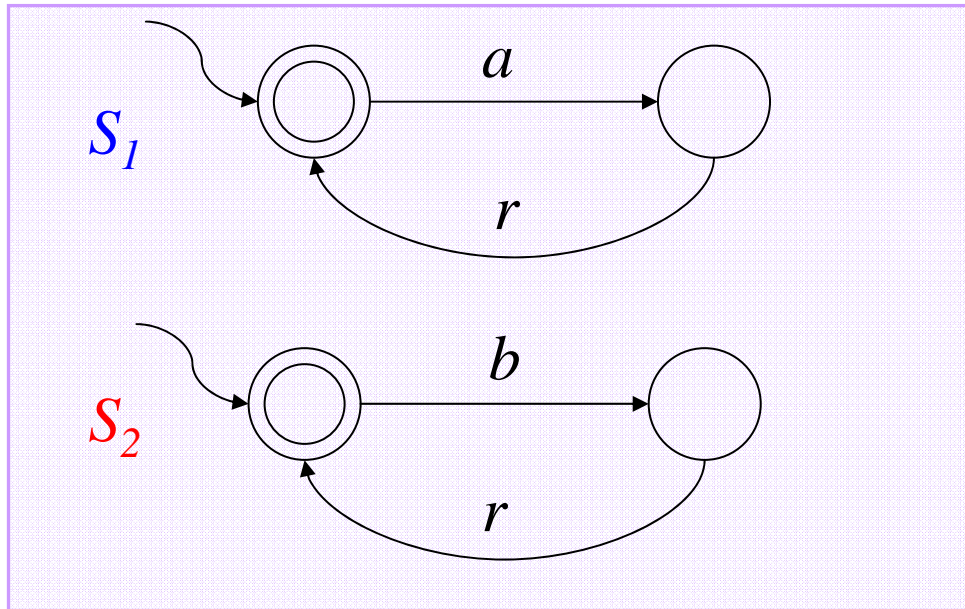
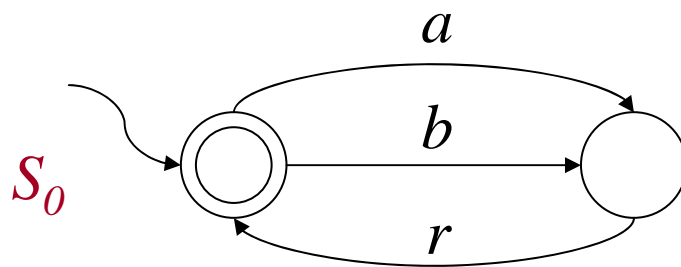
Execution tree of S_0



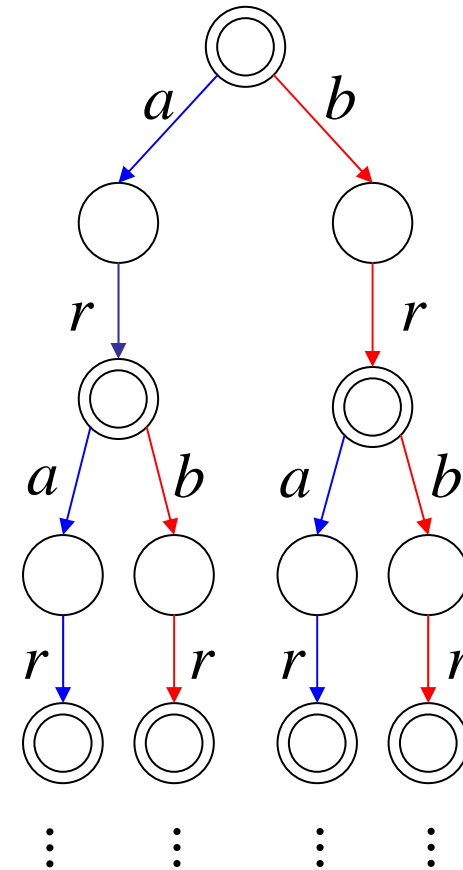
Note: we cannot label the target service TS directly ...

... we need to label the execution tree

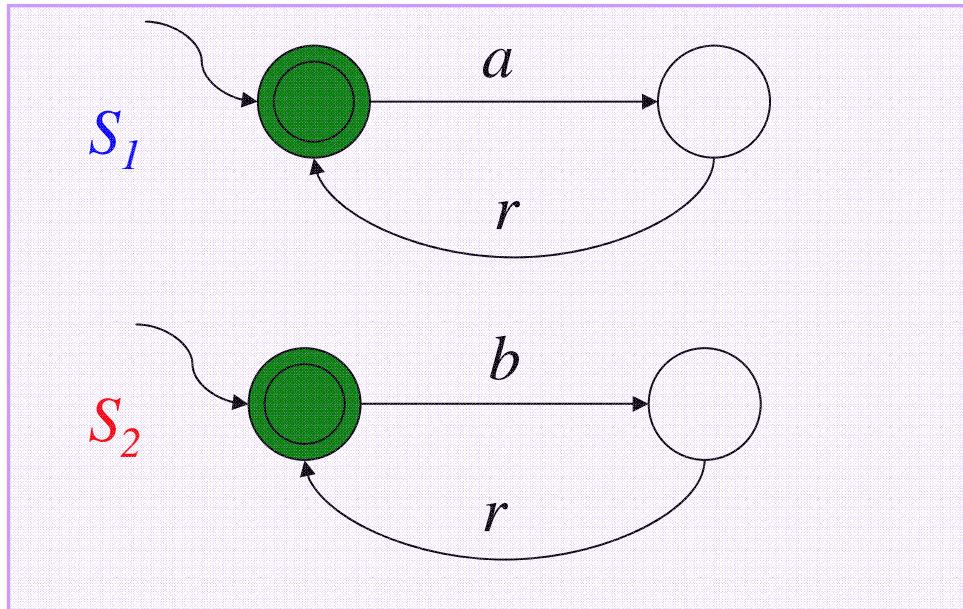
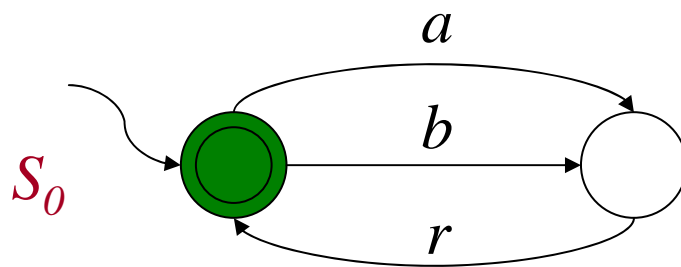
Example of Composition (3)



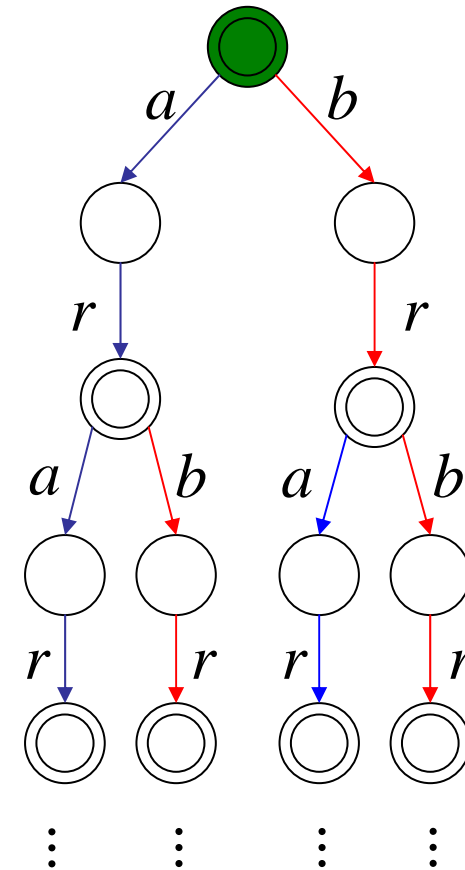
$$S_0 = \text{orch}(S_1 \parallel S_2)$$



Example of Composition (4)

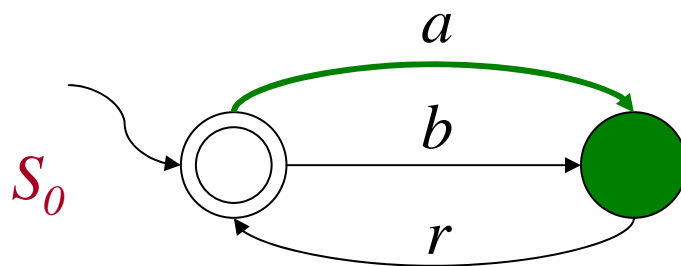


$$S_0 = \text{orch}(S_1 \parallel S_2)$$

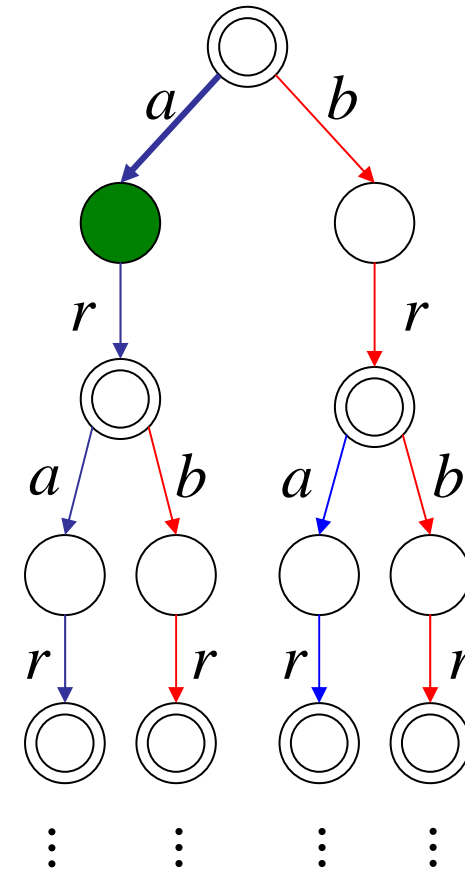
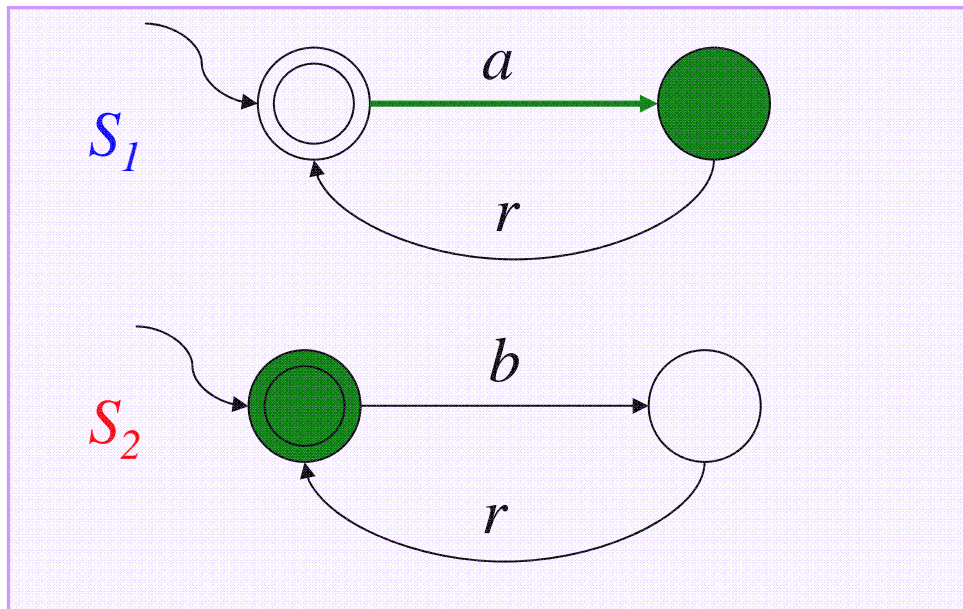


All services start from their starting state

Example of Composition (5)

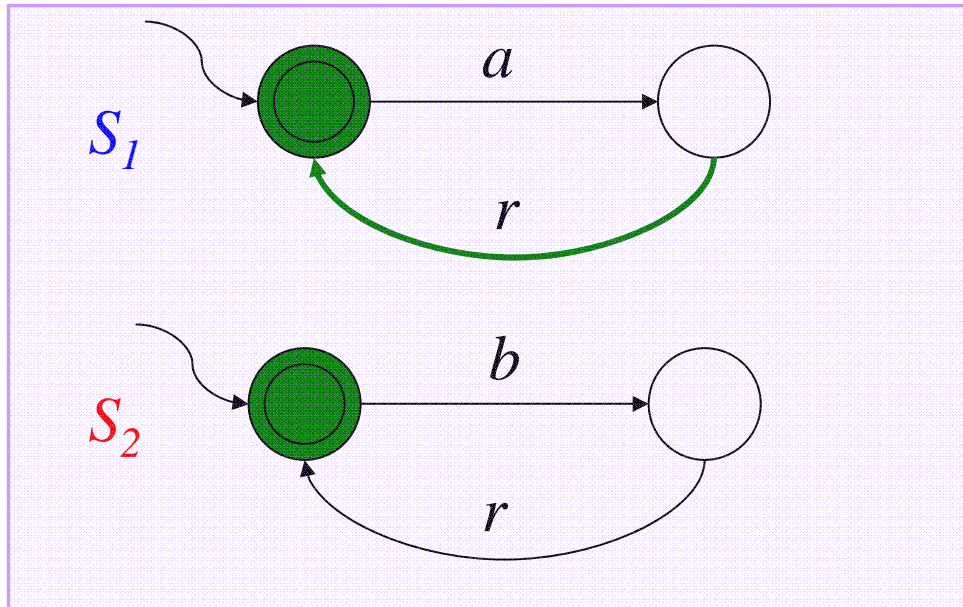
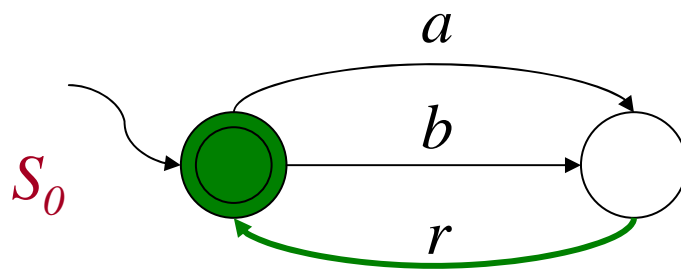


$$S_0 = \text{orch}(S_1 \parallel S_2)$$

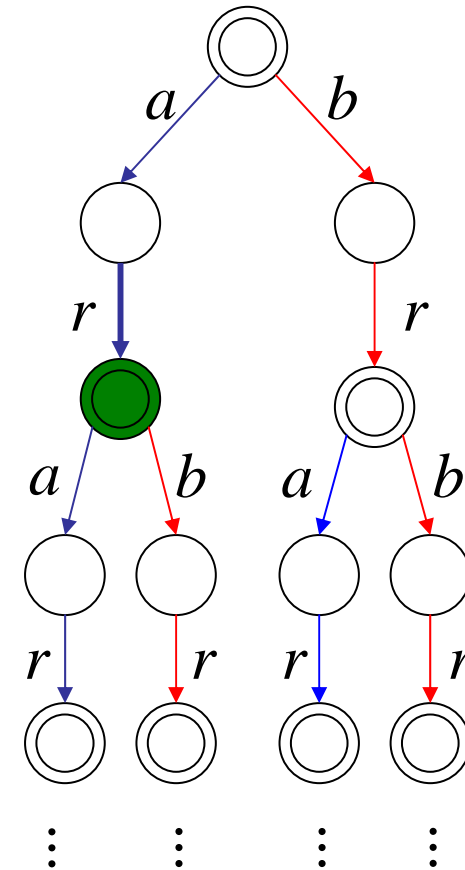


Each action of the target service is executed by at least one of the component services

Example of composition (6)

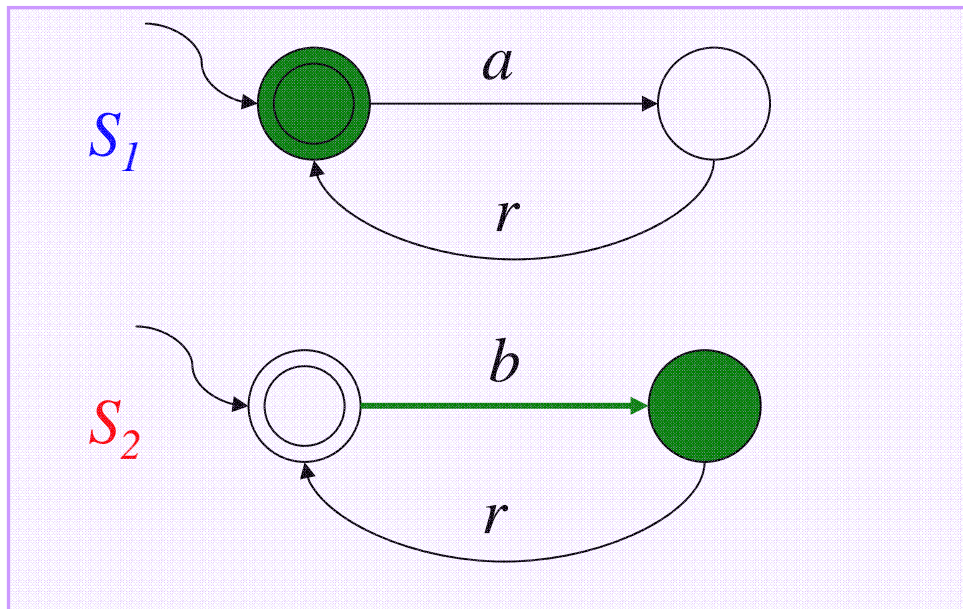
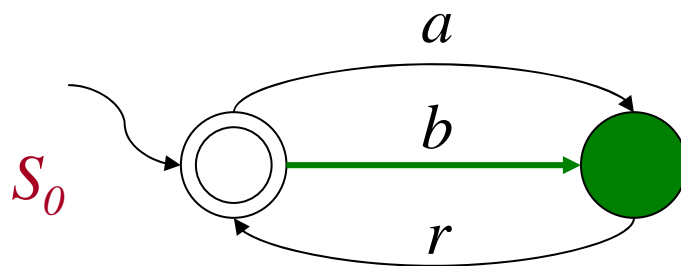


$$S_0 = \text{orch}(S_1 \parallel S_2)$$

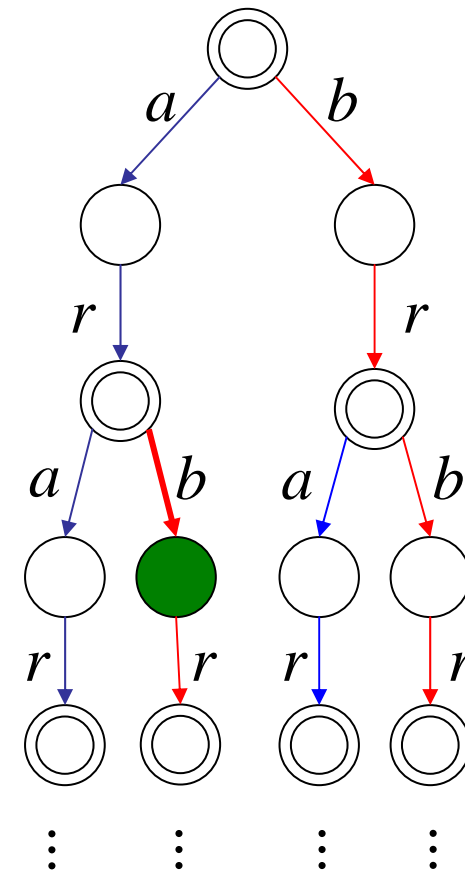


When the target service can be left, then all component services must be in a final state

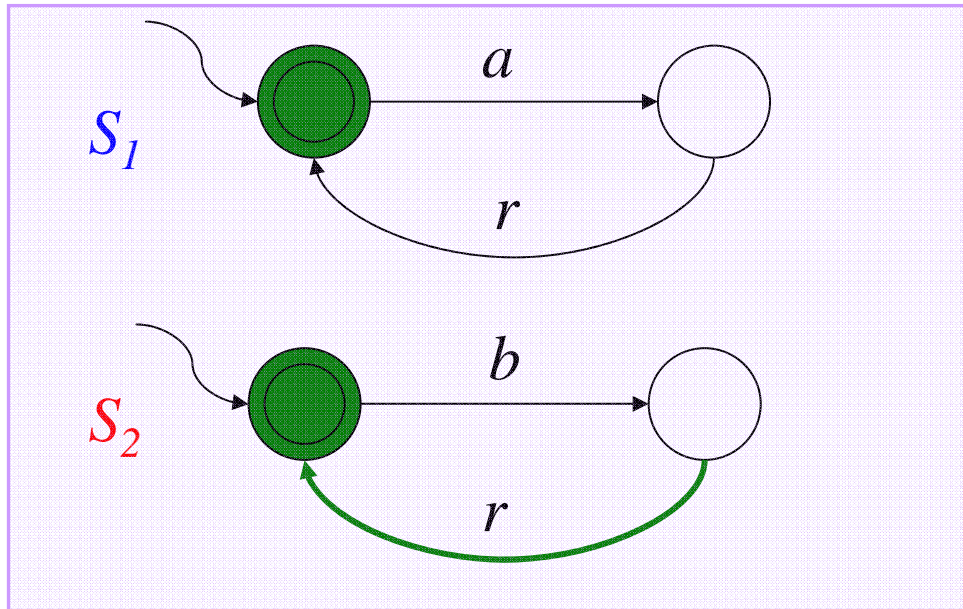
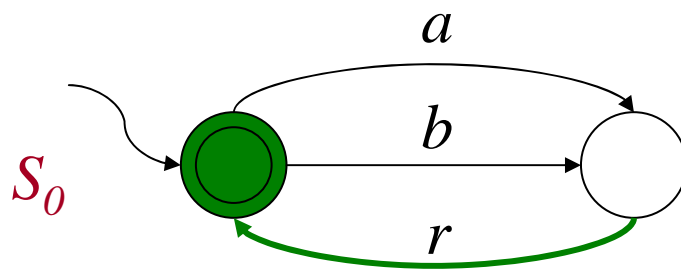
Example of composition (7)



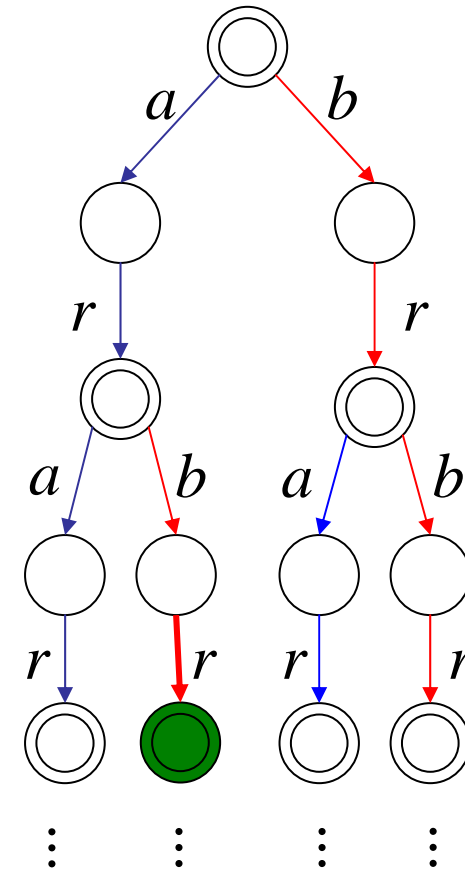
$$S_0 = \text{orch}(S_1 \parallel S_2)$$



Example of composition (8)

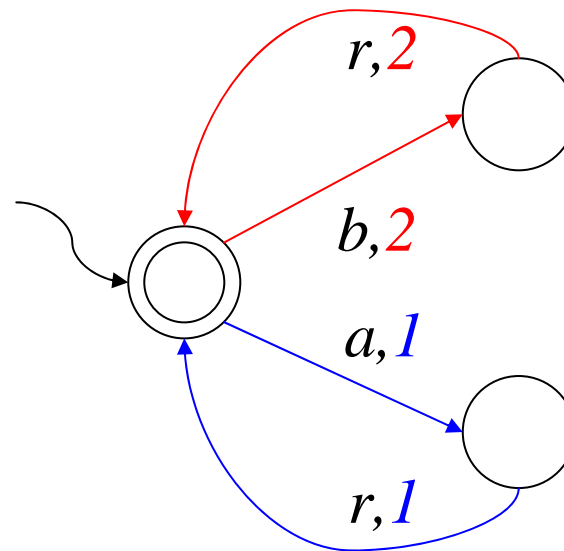


$$S_0 = \text{orch}(S_1 \parallel S_2)$$



Observation

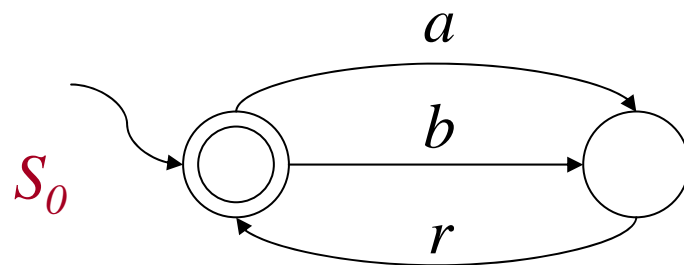
- This labeled execution tree has a finite representation as a finite TS ...
- ...with transitions labeled by an **action** and the **service** performing the action



Is this always the case when we deal with services expressible as finite TS? See later...

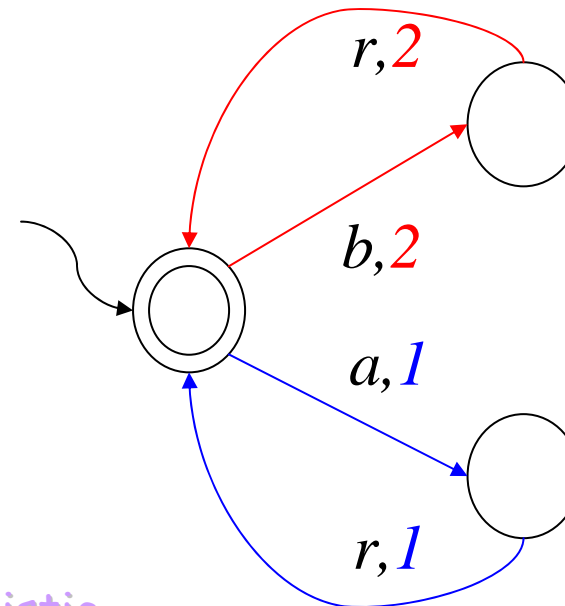
TS for Services and TS for Composition

Finite TS for services



- Deterministic
- Transitions labeled by actions
- Output on state to signal when final

Finite TS for composition



- Deterministic
- Transitions labeled by actions and services
- Output on transition to signal which service

Questions

Assume services of community and target service are finite TSs

- Can we always check composition existence?
- If a composition exists there exists one which is a finite TS?
- If yes, how can a finite TS composition be computed?

To answer we exploit PDL SAT

Answers

Reduce service composition synthesis to satisfiability in (deterministic) PDL

- Can we always check composition existence?
Yes, SAT in PDL is decidable in EXPTIME
- If a composition exists there exists one which is a finite TS?
Yes, by the small model property of PDL
- How can a finite TS composition be computed?
From a (small) model of the corresponding PDL formula

Structure of the PDL Encoding

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

Initial states of all
services

PDL encoding of i -
th component
service

PDL encoding of
target service

PDL additional
domain-
independent
conditions

PDL encoding is polynomial in the size of the service TSSs

PDL Encoding

- Target service $S_0 = (\Sigma, S_0, s_0^0, \delta_0, F_0)$ in PDL we define Φ_0 as the conjunction of:
 - $s \rightarrow \neg s'$ for all pairs of distinct states in S_0
service states are pair-wise disjoint
 - $s \rightarrow \langle a \rangle T \wedge [a]s'$ for each $s' = \delta_0(s, a)$
target service can do an a-transition going to state s'
 - $s \rightarrow [a] \perp$ for each $\delta_0(s, a)$ undef.
target service cannot do an a-transition
 - $F_0 \equiv \bigvee_{s \in F_0} s$
denotes target service final states
- ...

PDL Encoding (cont. d)

- Community services $S_i = (\Sigma, S_i, s^0_i, \delta_i, F_i)$ in PDL we define Φ_i as the conjunction of:
 - $s \rightarrow \neg s'$ for all pairs of distinct states in S_i
Service states are pair-wise disjoint
 - $s \rightarrow [a](\text{moved}_i \wedge s' \vee \neg \text{moved}_i \wedge s)$ for each $s' = \delta_i(s, a)$
if service moved then new state, otherwise old state
 - $s \rightarrow [a](\neg \text{moved}_i \wedge s)$ for each $\delta_i(s, a)$ undef.
if service cannot do a, and a is performed then it did not move
 - $F_i \equiv \bigvee_{s \in F_i} s$
denotes community service final states
- ...

PDL Encoding (cont. d)

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

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

Results

Thm: Composition exists iff PDL formula Φ SAT

From composition labeling of the target service one can build a tree model of the PDL formula and viceversa

Information on the labeling is encoded in predicates moved;

\Rightarrow Composition existence of services expressible as finite TS is decidable in EXPTIME

Results on TS Composition

Thm: If composition exists then finite TS composition exists.

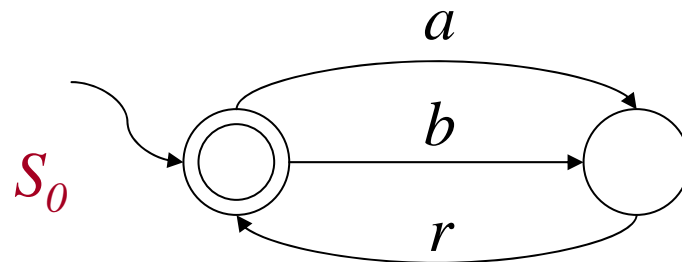
*From a small model of the PDL formula Φ ,
one can build a finite TS machine*

*Information on the output function of the machine is encoded in
predicates moved;*

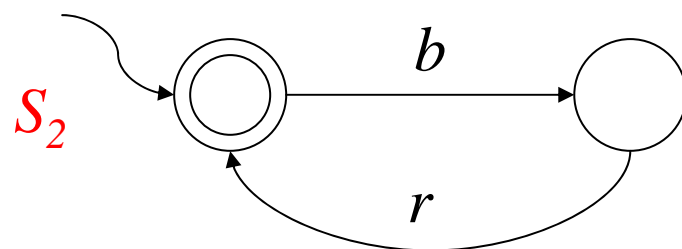
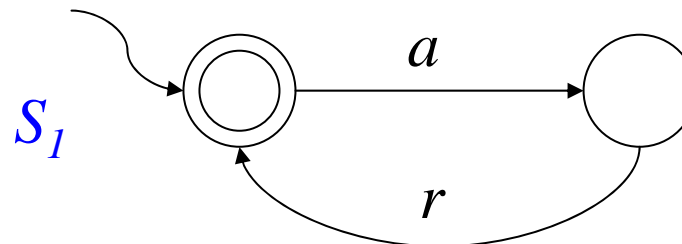
⇒ finite TS composition existence of services
expressible as finite TS is decidable in EXPTIME

Example (1)

Target service



Community services



DPDL

...
...
...

$$s_0^0 \wedge s_1^0 \wedge s_2^0$$

$$\langle a \rangle T \rightarrow [a] (\text{moved}_1 \vee \text{moved}_2)$$

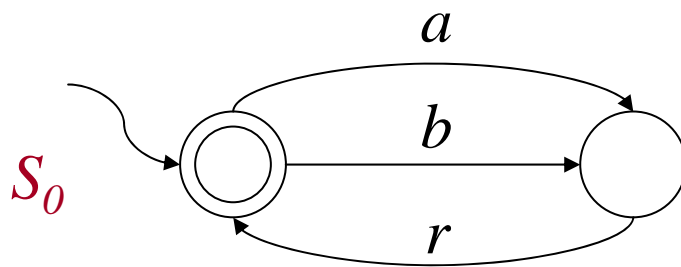
$$\langle b \rangle T \rightarrow [b] (\text{moved}_1 \vee \text{moved}_2)$$

$$\langle r \rangle T \rightarrow [r] (\text{moved}_1 \vee \text{moved}_2)$$

$$F_0 \rightarrow F_1 \wedge F_2$$

Example (2)

Target service



$$s_0^0 \rightarrow \neg s_0^1$$

$$s_0^0 \rightarrow \langle a \rangle \top \wedge [a] s_0^1$$

$$s_0^0 \rightarrow \langle b \rangle \top \wedge [b] s_0^1$$

$$s_0^1 \rightarrow \langle r \rangle \top \wedge [r] s_0^0$$

$$s_0^0 \rightarrow [r] \perp \wedge [r] s_0^0$$

$$s_0^1 \rightarrow [a] \perp$$

$$s_0^1 \rightarrow [b] \perp$$

$$F_0 \equiv s_0^0$$

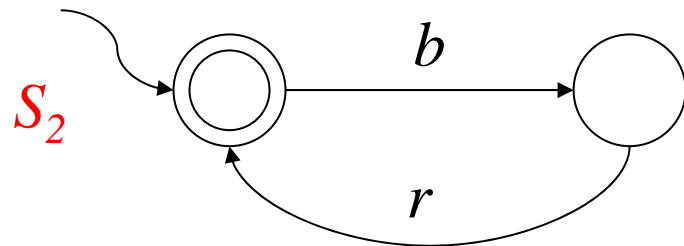
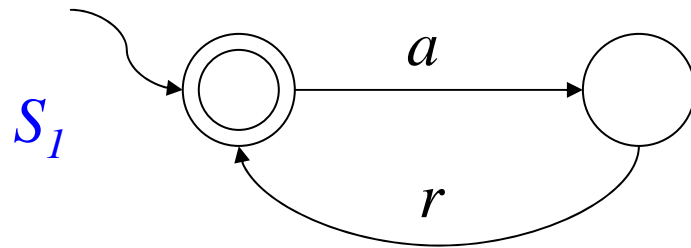
...

...

...

Example (3)

Community services



...

$$s_1^0 \rightarrow \neg s_1^1$$

$$s_1^0 \rightarrow [a] (\text{moved}_1 \wedge s_1^1 \vee \neg \text{moved}_1 \wedge s_1^0)$$

$$s_1^0 \rightarrow [r] \neg \text{moved}_1 \wedge s_1^0$$

$$s_1^1 \rightarrow [b] \neg \text{moved}_1 \wedge s_1^1$$

$$s_1^1 \rightarrow [a] \neg \text{moved}_1 \wedge s_1^1$$

$$s_1^1 \rightarrow [b] \neg \text{moved}_1 \wedge s_1^1$$

$$s_1^1 \rightarrow [r] (\text{moved}_1 \wedge s_1^0 \vee \neg \text{moved}_1 \wedge s_1^0)$$

$$F_1 \equiv s_1^0$$

$$s_2^0 \rightarrow \neg s_2^1$$

$$s_2^0 \rightarrow [b] (\text{moved}_2 \wedge s_2^1 \vee \neg \text{moved}_2 \wedge s_2^0)$$

$$s_2^0 \rightarrow [r] \neg \text{moved}_2 \wedge s_2^0$$

$$s_2^0 \rightarrow [a] \neg \text{moved}_2 \wedge s_2^0$$

$$s_2^1 \rightarrow [b] \neg \text{moved}_2 \wedge s_2^1$$

$$s_2^1 \rightarrow [a] \neg \text{moved}_2 \wedge s_2^1$$

$$s_2^1 \rightarrow [r] (\text{moved}_2 \wedge s_2^0 \vee \neg \text{moved}_2 \wedge s_2^0)$$

$$F_2 \equiv s_2^0$$

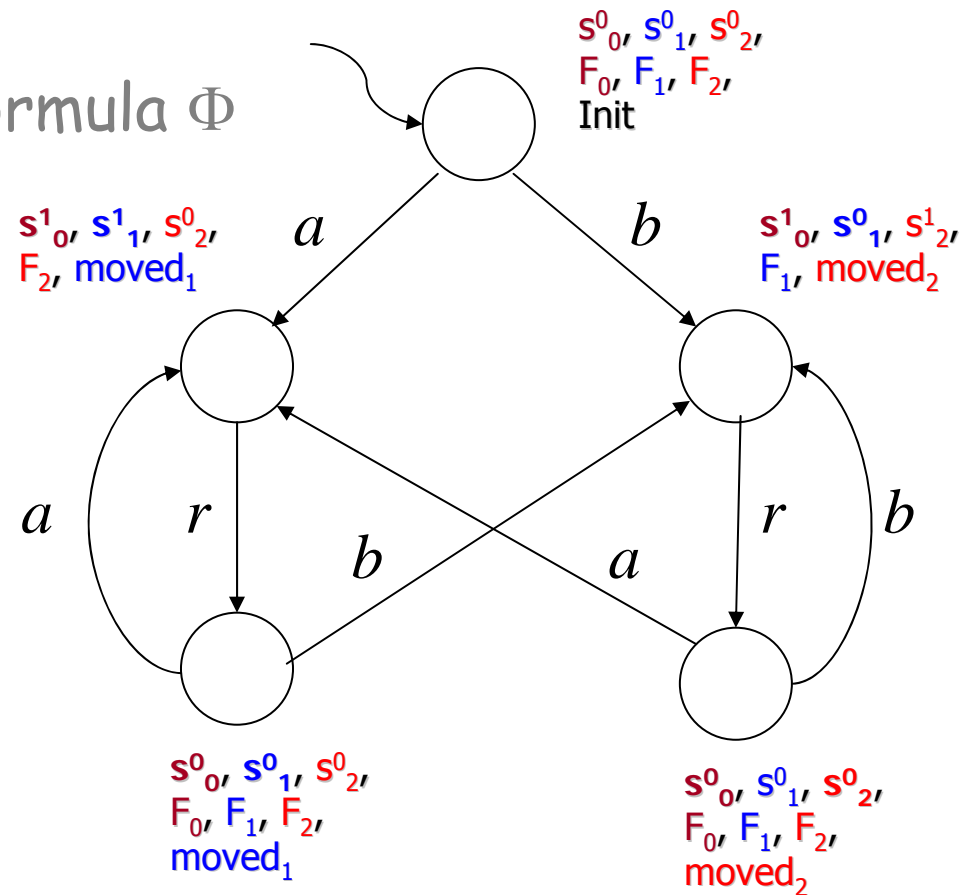
Example (4)



Check: run SAT on PDL formula Φ

Example

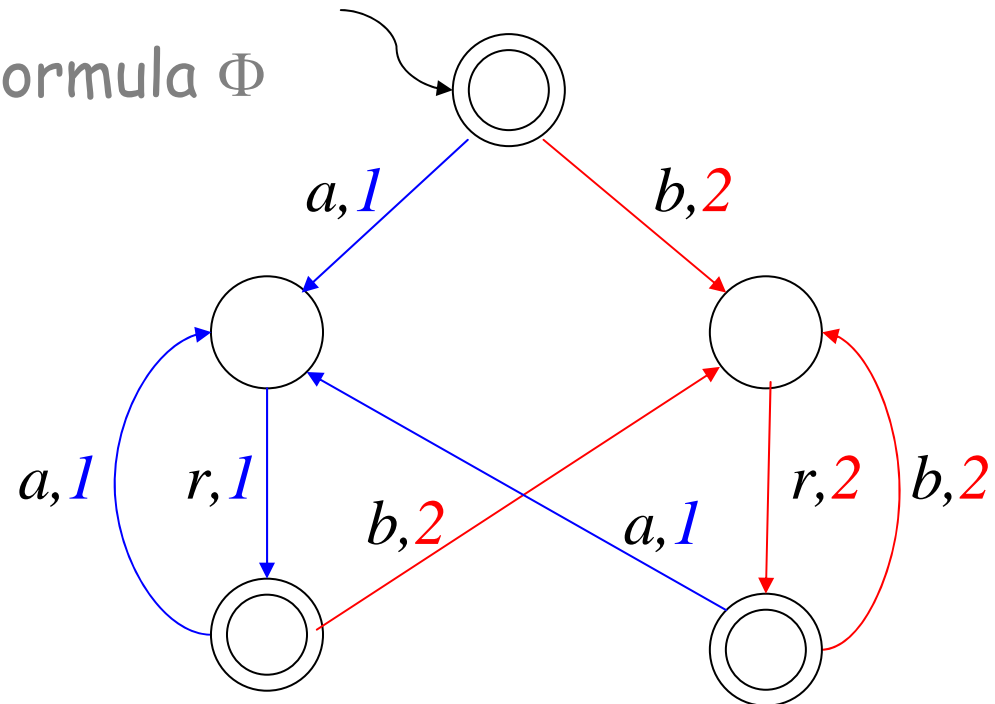
Check: run SAT on PDL formula Φ
Yes \Rightarrow (small) model



Example

Check: run SAT on PDL formula Φ
Yes \Rightarrow (small) model

\Rightarrow extract finite TS



Example

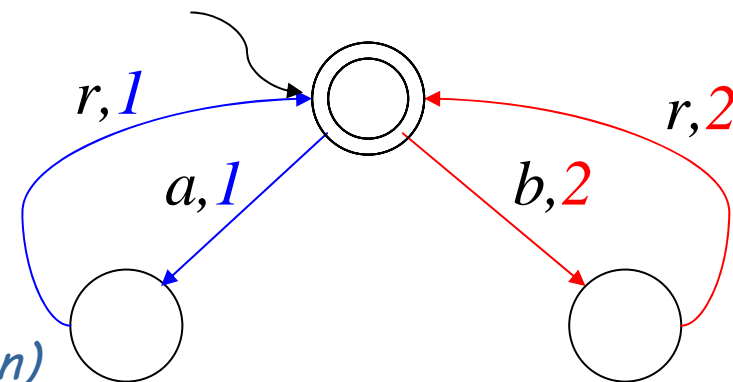
Check: run SAT on PDL formula Φ

Yes \Rightarrow (small) model

\Rightarrow extract finite TS

\Rightarrow minimize finite TS

(similar to Mealy machine minimization)



Results on Synthesizing Composition

- Using PDL reasoning algorithms based on model construction (cf. tableaux), build a (small) model
Exponential in the size of the PDL encoding/services finite TS
- From this model extract a corresponding finite TS
Polynomial in the size of the model
- Minimize such a finite TS using standard techniques (opt.)
Polynomial in the size of the TS

Note: finite TS extracted from the model is not minimal because encodes output in properties of individuals/states

Tools for Synthesizing Composition

- In fact we use only a fragment of PDL in particular we use fixpoint (transitive closure) only to get the universal modality ...
- ... thanks to a tight correspondence between PDLs and Description Logics (DLs), we can use current highly optimized DL reasoning systems to do synthesis ...
- ... when the ability or returning models will be added ...
- ... meanwhile we have developed a prototype tool on this idea (see ESC - E-Service Composer:
<http://sourceforge.net/projects/paride>)

Extensions

- Loose specification of target service
 - target service "under-specified"
 - the client delegates the composer to resolve certain choices

Note: angelic nondeterminism

- Interaction between services without client involvement
 - A component service may initiate an interaction with another component service to get to a certain state
 - Client specifies when during the computation this is allowed and to what extent

Note: again angelic nondeterminism

See ICSOC04

Extensions

- Incomplete specification of services of the community
 - Component services export their behavior as nondeterministic TS (also related to data acquisition) ...

Note: diabolic nondeterminism
 - ...the transitions are not completely under the control of the client nor of the orchestrator ...
 - ... composition has to work in spite of choices made by component services
 - Two possible assumptions
 1. Orchestrator can observe state reached by component service
 2. Orchestrator cannot fully observe state reached by component service

For 1, variants of the technique shown here still apply
- work in progress

For 2, most likely techniques based on automata on infinite trees are needed – see Vardi's work on process synthesis

References

Roman Group

- *See above*

Process Synthesis

[Vardi&Kupferman ICTL97] M. Vardi, O. Kupferman: Synthesis with Incomplete Information (full version of ICTL'97).

[Vardi CAV95] M. Vardi, An Automata-Theoretic Approach to Fair Realizability and Synthesis. CAV 1995.

[Vardi&Kupferman MFCS2000] M. Vardi, O. Kupferman: Mu-calculus Synthesis. MFCS 2000

[Kupferman & Vardi LCS01] Orna Kupferman, Moshe Y. Vardi: Synthesizing Distributed Systems. Logic in Computer Science, 2001.

Description Logics and relationship with Logic of Programs

[BCMNS2003] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider: The Description Logic Handbook: Theory, Implementation, and Applications Cambridge University Press 2003

[Calvanese et al HAR01] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi: Reasoning in Expressive Description Logics. Handbook of Automated Reasoning 2001: 1581-1634

Conclusions

Summary

- Review of relevant technologies
- Distinction between orchestration and choreography
- Relevance of transition systems for abstracting over technologies
- State of the art in service composition
- Automatic composition: a conceptual perspective
- Transition systems and formal systems for verification and synthesis
- In-depth analysis of a particular approach

What to Bring Home

- Composition is not Choreography
 - Composition: we are starting to understand*
 - Choreography: still needs to be developed*
- Relevance of Transition Systems
 - Don't confuse with language-theoretic automata*
- Relevance of Data Integration
 - If you are involved with information-based services, study data integration literature*
 - If you are not involved, look at it anyway, e.g., GAV and LAV*
- Relevance of Planning
 - If you can afford to have clients not driving the control flow, planning has a lot to say*
 - If not, planning doesn't directly apply, but has a lot to say anyway (it is a rich area)*
- Relevance of Process Verification and Synthesis
 - Full potential, still largely unexplored*

Having clients driving the control flow is a specific feature of service composition

"... say no more"



Additional Slides

Composition by Simulation

Simulation

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

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

- if s is *final* then t is *final*
- for all actions a
 - if $s \rightarrow_a s'$ then $\exists t' . t \rightarrow_a t'$ and $(s',t') \in R$

Note it is a co-inductive definition!

Essentially is one direction of the bisimulation!

- A transition system S **simulates** a transition system T iff there **exists a simulation** between the initial states s_0 and t_0 .

Potential behavior of the community

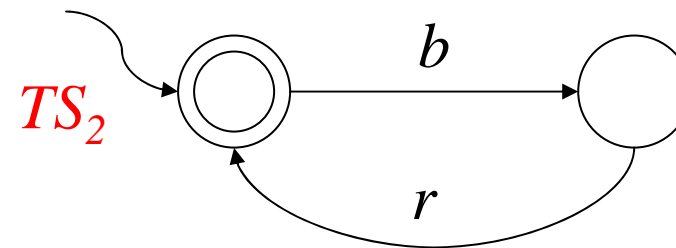
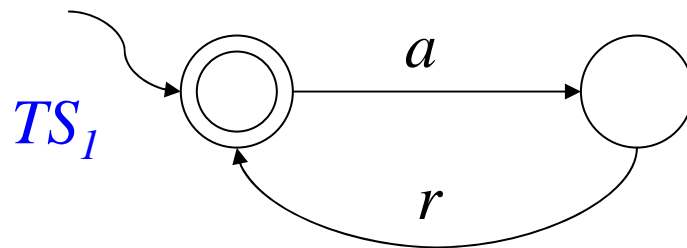
- Let TS_1, \dots, TS_n be the TSs of the component services.
- The Community Big TS is defined as $TS_c = \langle A, S_c, S_c^0, \delta_c, F_c \rangle$ where:
 - A is the set of actions
 - $S_c = S_1 \times \dots \times S_n$
 - $S_c^0 = \{(s_1^0, \dots, s_n^0)\}$
 - $F \subseteq F_1 \times \dots \times F_n$
 - $\delta_c \subseteq S_c \times A \times S_c$ is defined as follows:
 - $(s_1 \times \dots \times s_n) \rightarrow_a (s'_1 \times \dots \times s'_n)$ iff
 - 1. $\exists i. s_i \rightarrow_a s'_i \in \delta_i$
 - 2. $\forall j. s_j \rightarrow_a s'_j \in \delta_j \vee s'_j = s_j$

Composition by Simulation

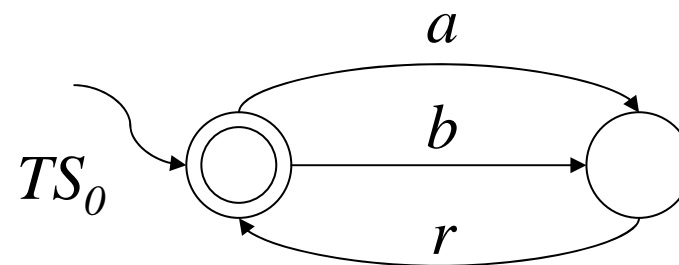
- **Thm:** A **composition exists** iff there exists a **simulation** from (the initial state of) TS_c to (the initial state of) TS_+
- Given a **simulation** R from TS_c to TS_+ (which include the initial states), we can build a **finite composition** as follows:
 $TS = \langle A_r, S_r, S_r^0, \delta_r, F_r \rangle$ with
 - $A_r = A \times 2^{[n]}$
 - $S_c = S_1 \times \dots \times S_n \times S_+$
 - $S_c^0 = \{ (s_1^0, \dots, s_n^0, s_+^0) \}$
 - $F \subseteq \{ (s_1 \times \dots \times s_n \times s) \mid s \in F_+ \}$
 - $\delta_r \subseteq S_r \times A_r \times S_r$ is defined as follows:
 - $(s_1 \times \dots \times s_n, s) \rightarrow_{a, I} (s'_1 \times \dots \times s'_n, s')$ iff
 1. $s \rightarrow_a s'$
 2. $(s_1 \times \dots \times s_n) \rightarrow_{a, I} (s'_1 \times \dots \times s'_n)$
 3. $((s'_1 \times \dots \times s'_n), s) \in R$
 4. $I = \{ j \mid s_j \rightarrow_a s'_j \in \delta_j \}$

Example of Composition

- Community Services



- Target Service



Example of Composition

Community Big Service

