



SAPIENZA  
UNIVERSITÀ DI ROMA

---

# *Automatic Web Service Composition*

**Giuseppe De Giacomo  
Massimo Mecella**

*Dipartimento di Informatica e Sistemistica  
"Antonio Ruberti"*

*SAPIENZA -- Università di Roma*

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

based on joint work with Daniela Berardi, Diego Calvanese, Rick Hull,  
Alessandro Iuliani, Maurizio Lenzerini, Damiano Pozzi, Ruggero Russo

# Instructors

(i.e., the Odd Couple)



SAPIENZA  
UNIVERSITÀ DI ROMA

- 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 (1)



- Lecture 1 (Tue. June 5 - 09.15 - 10.30)  
*Introduction to Web Service Technologies*  
(Mecella)
- Lecture 2 (Tue. June 5 - 13.30 - 15.00)  
*Transition-based Composition Synthesis:  
Basic Concepts on Transition Systems*  
(De Giacomo)
- Lecture 3 (Wed. June 6 - 13.30 - 15.00)  
*Transition-based Composition Synthesis: the  
Roman Approach*  
(De Giacomo)

# Outline (2)



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- Lecture 4 (Thu. June 7 - 09.00 - 10.30)  
*Automated Composition: State of the Art*  
(Mecella)
  
- Lecture 5 (Thu. June 7 - 15.15 - 16.45)  
*WSCE - Web Service Composition Engine & Other Stuffs (security, ...)*  
(Mecella)

**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.



## *Lecture 1*

1. Basic Technologies
2. Abstracting Service Behaviors

# *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)*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

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)*

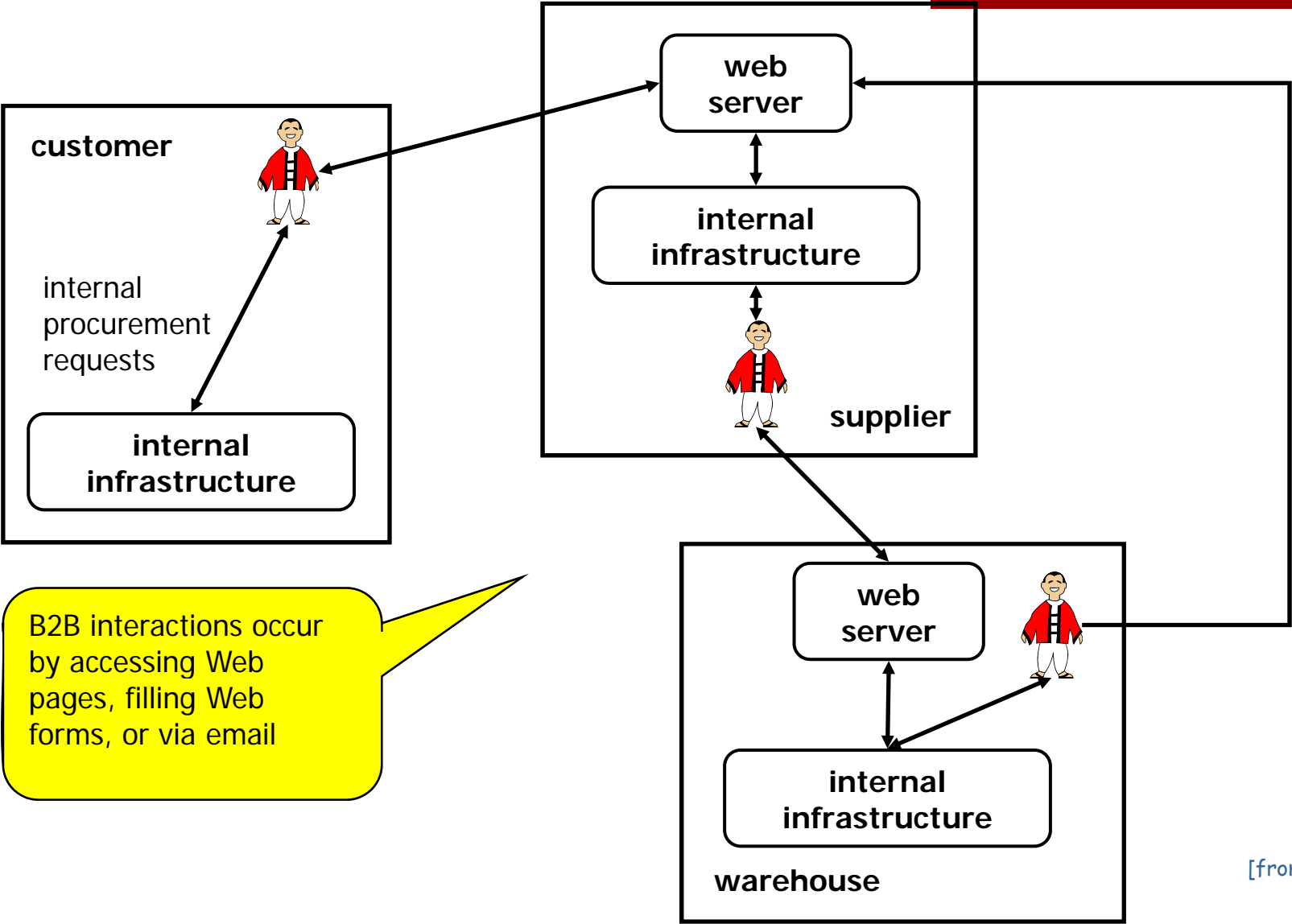


- **Services are self-describing, open components that support rapid, low-cost composition of distributed applications.** Services are offered by service providers — organizations that procure the service implementations, supply their service descriptions, and provide related technical and business support. 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 [CACM03]*



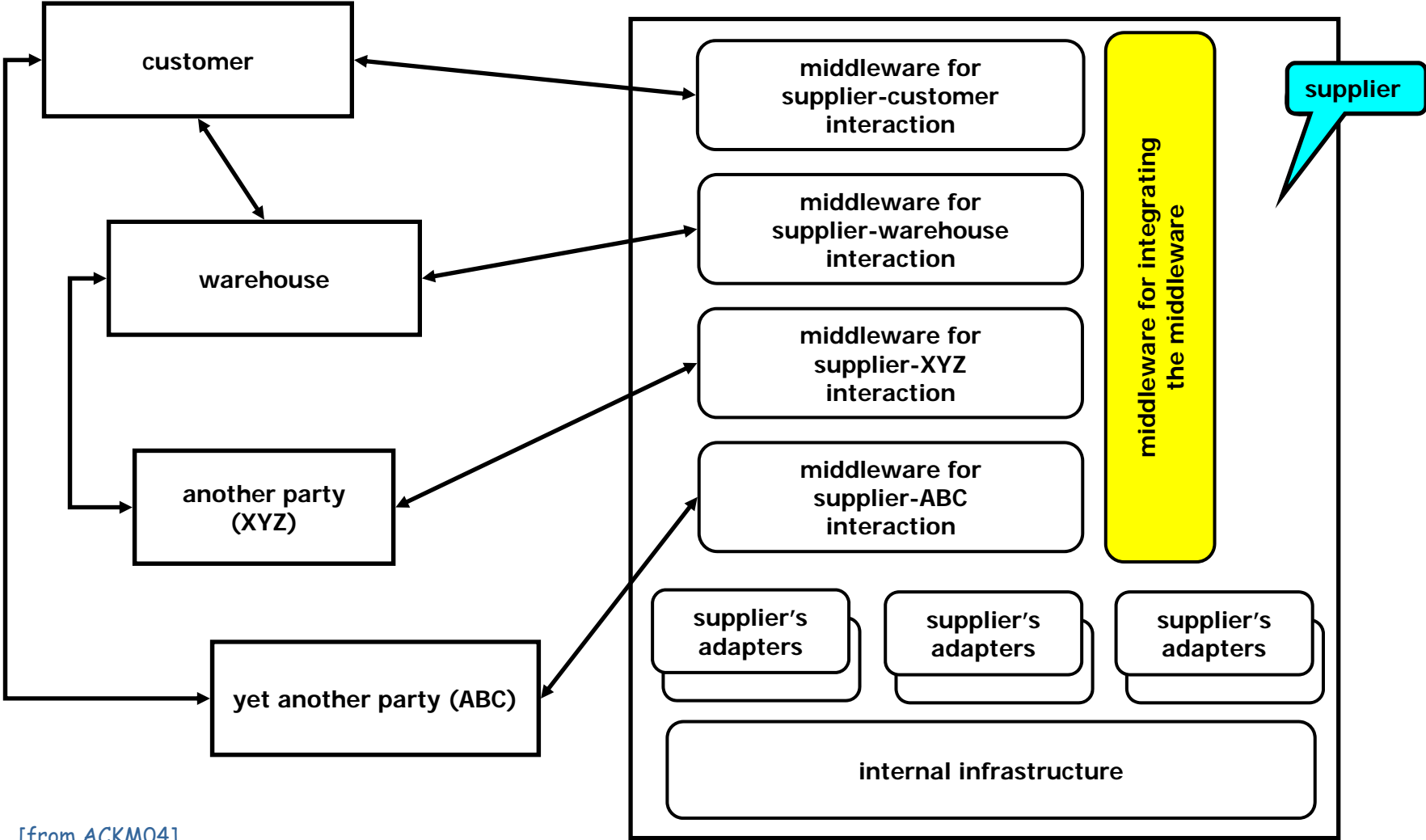
# *(naive) Business-to-Business Integration*



B2B interactions occur by accessing Web pages, filling Web forms, or via email

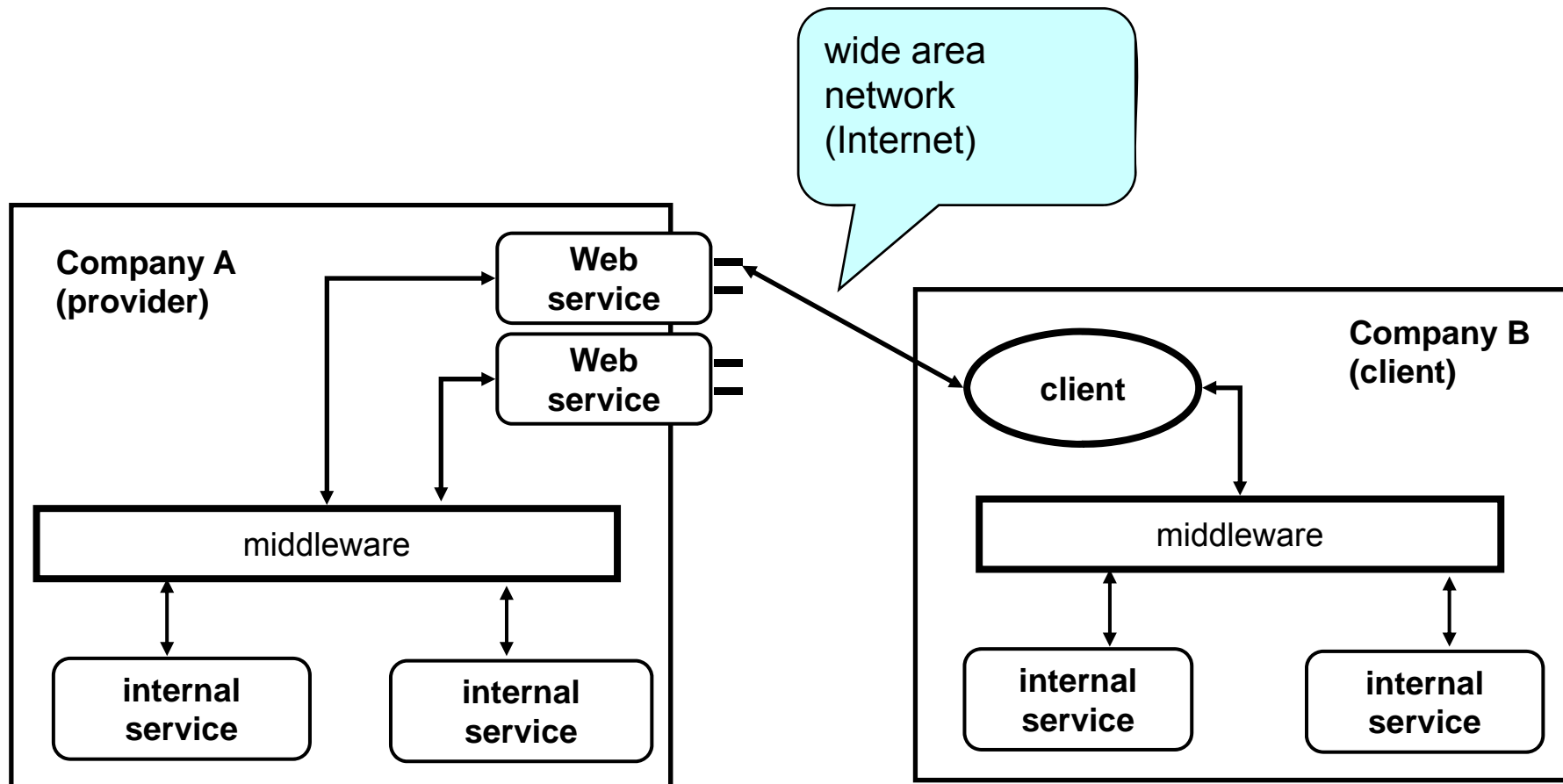
[from ACKM04]

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



[from ACKM04]

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

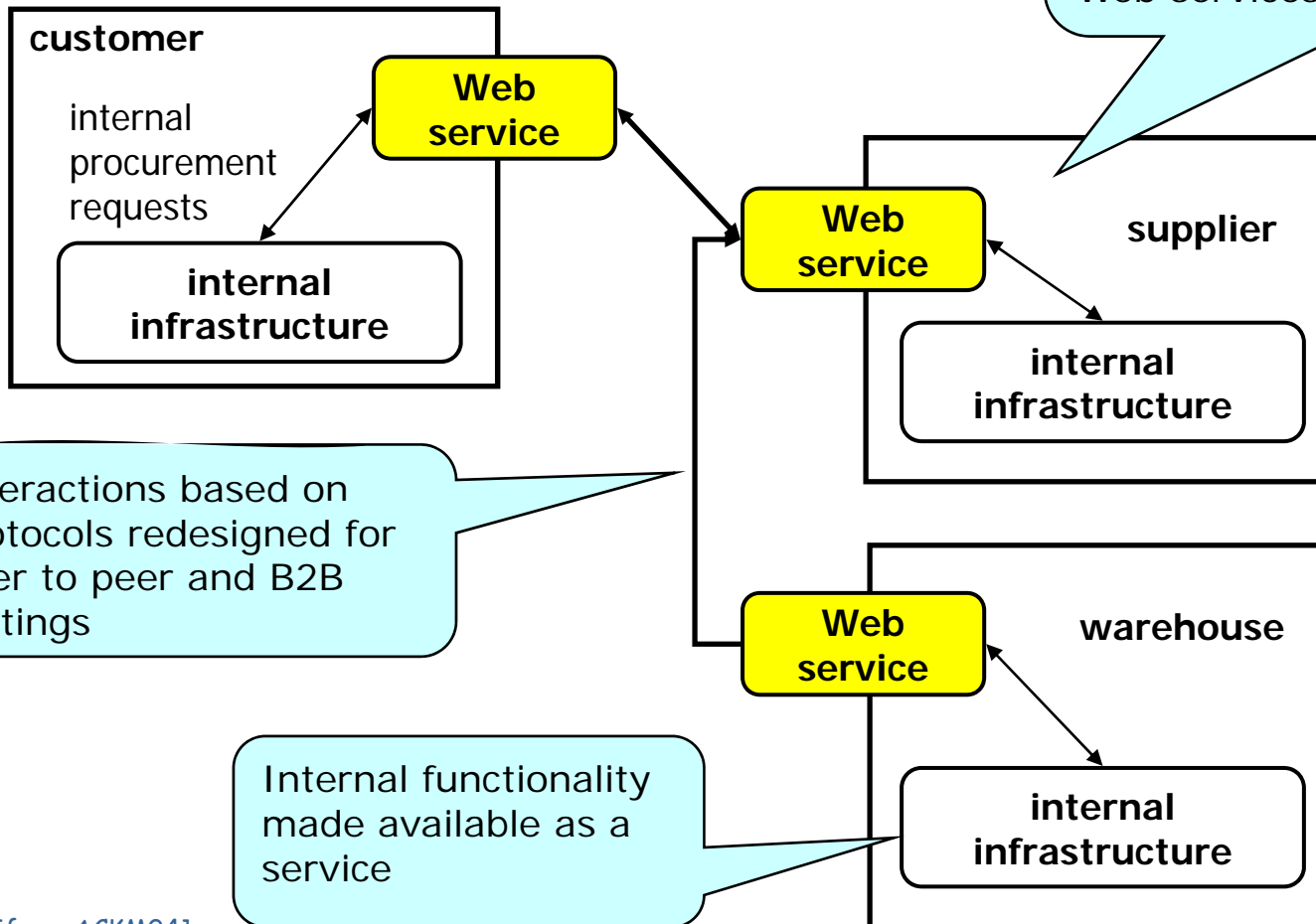


[from ACKM04]

# *(WS-based) Business-to-Business Integration*



Standardized languages and protocols, eliminating the need for many different middleware infrastructures (need only the Web services middleware)



Interactions based on protocols redesigned for peer to peer and B2B settings

Internal functionality made available as a service

[from ACKM04]

# *When Web Services Should Be Applied ?*



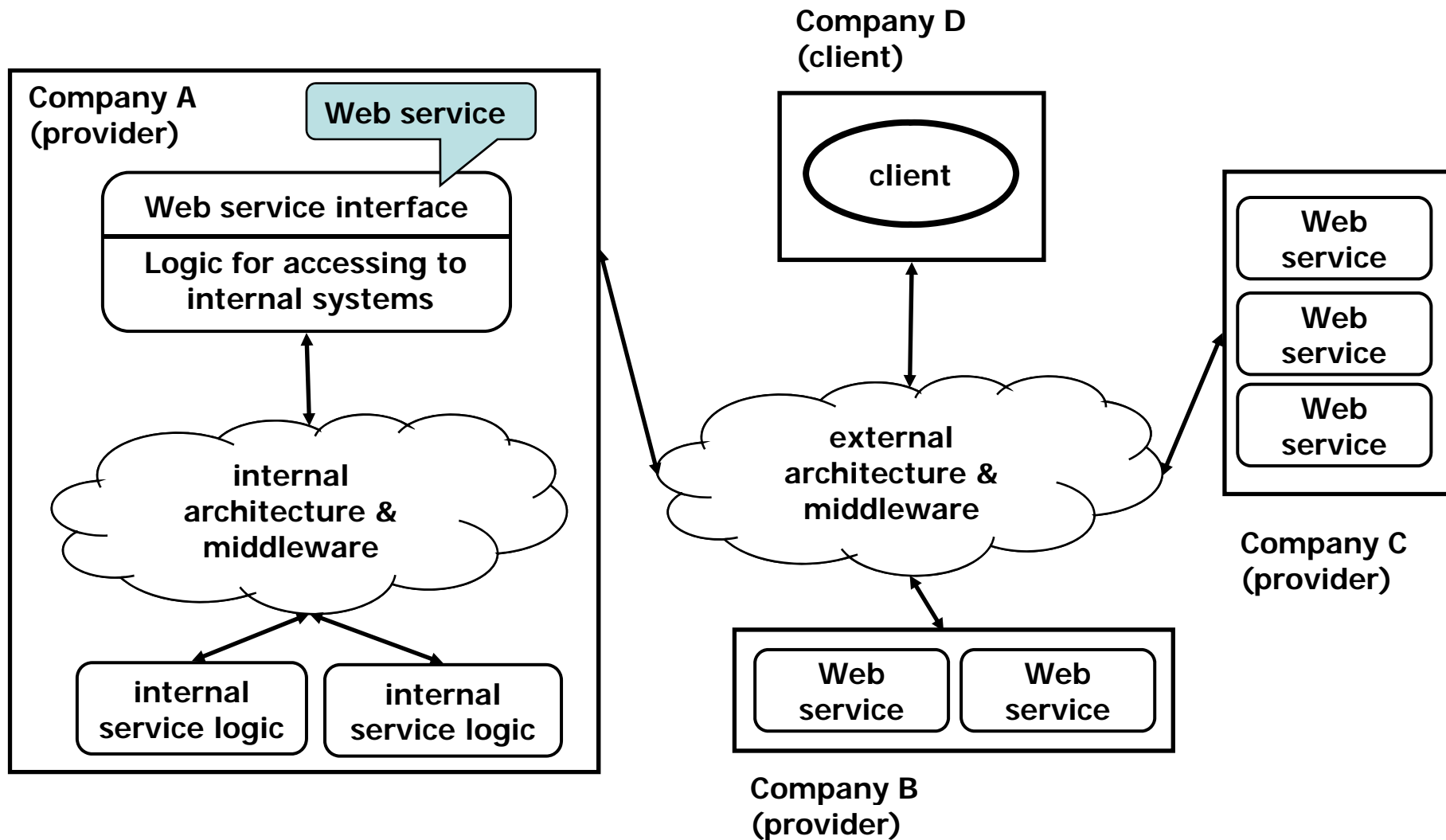
SAPIENZA  
UNIVERSITÀ DI ROMA

---

- When it is no 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

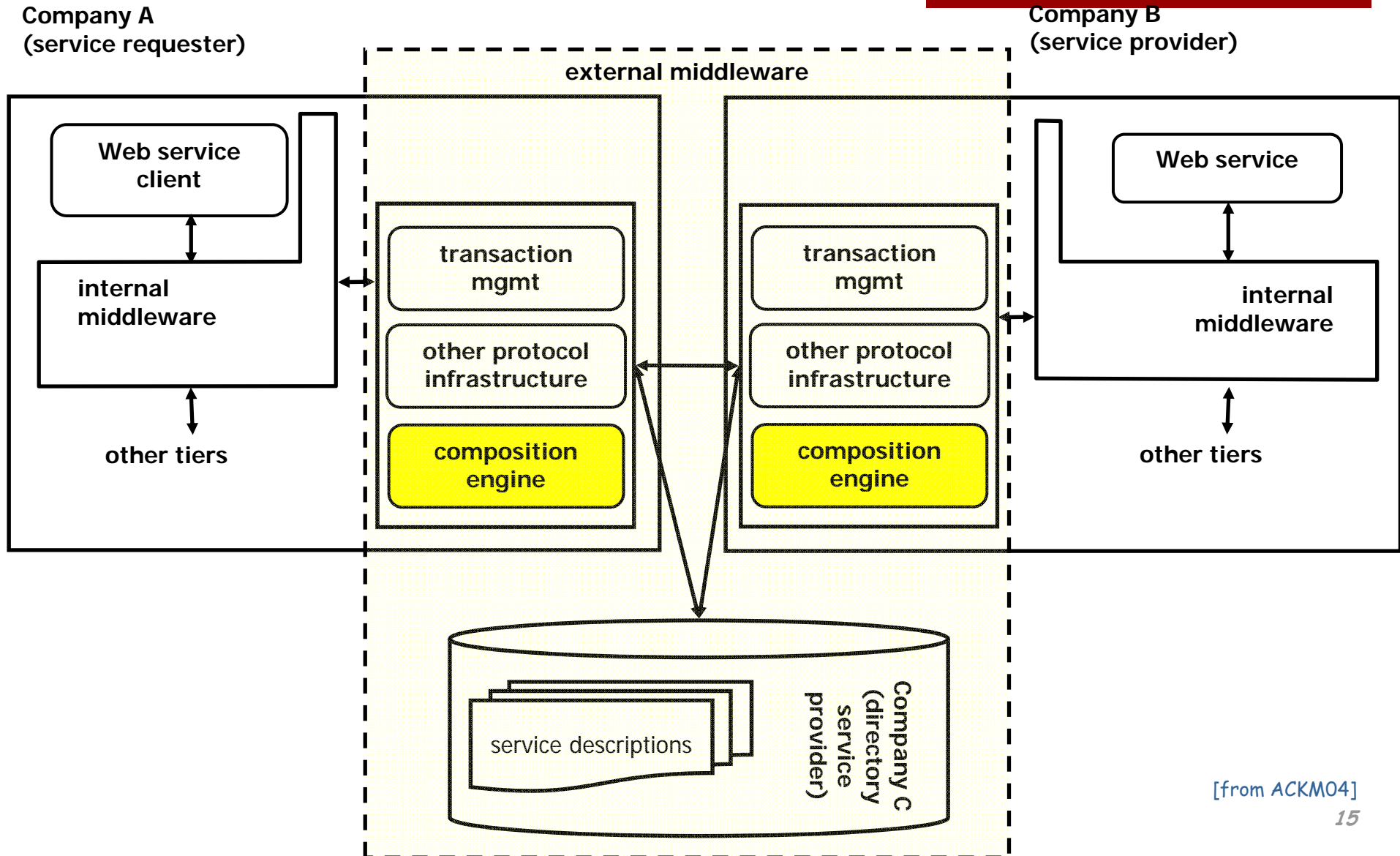
*Web Services Architecture,  
W3C Working Group Note, 11 Feb. 2004,  
<http://www.w3.org/TR/ws-arch/>*

# Two Architectures (and Middlewares) (1)



[from ACKM04]

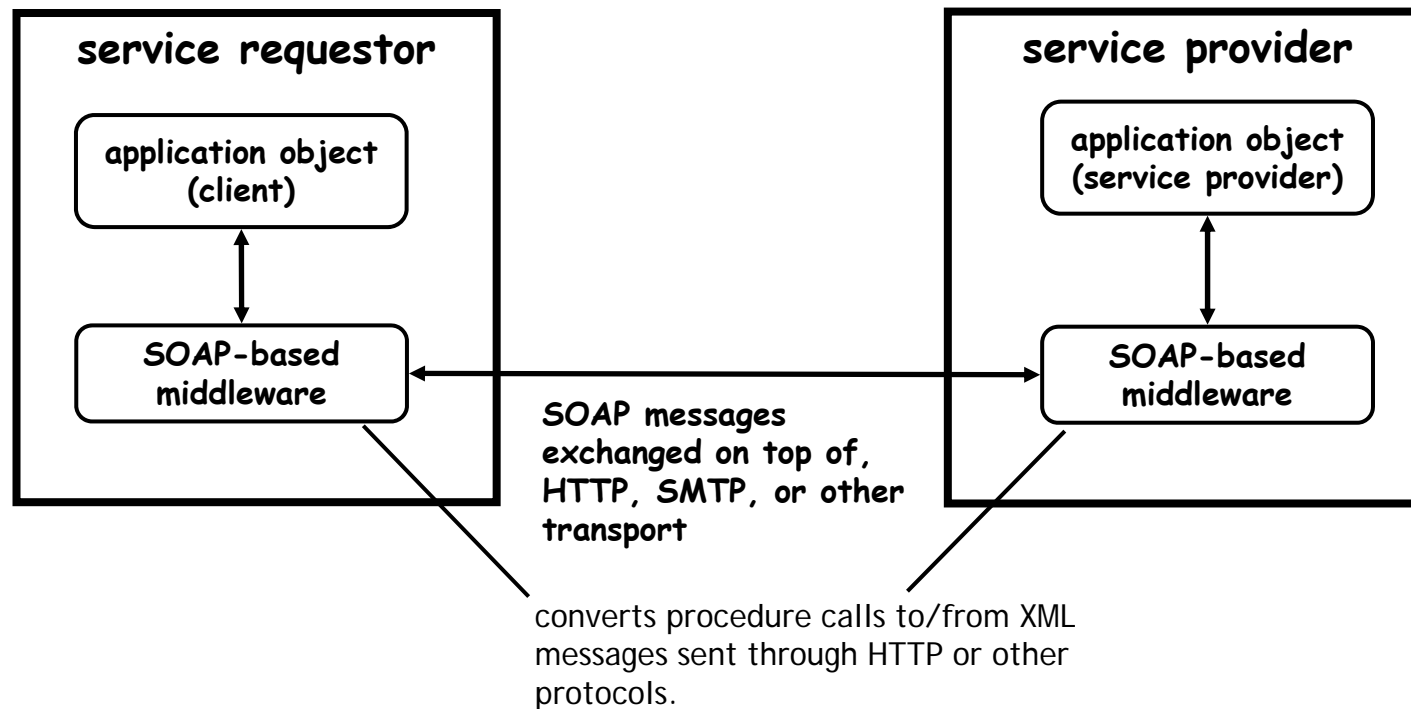
# Two Architectures (and Middlewares) (2)



[from ACKM04]



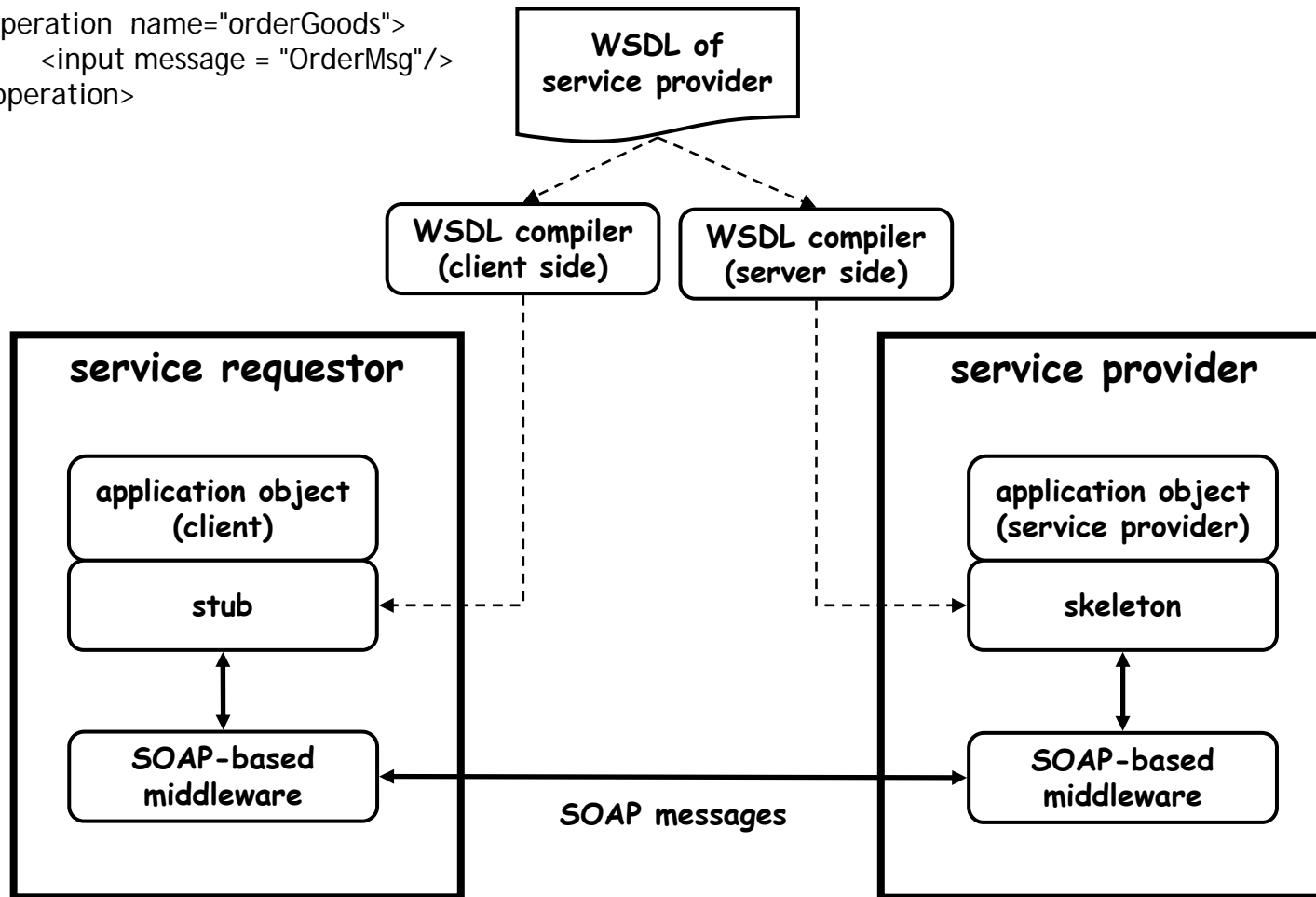
# *A Minimalist Infrastructure for Web Service*



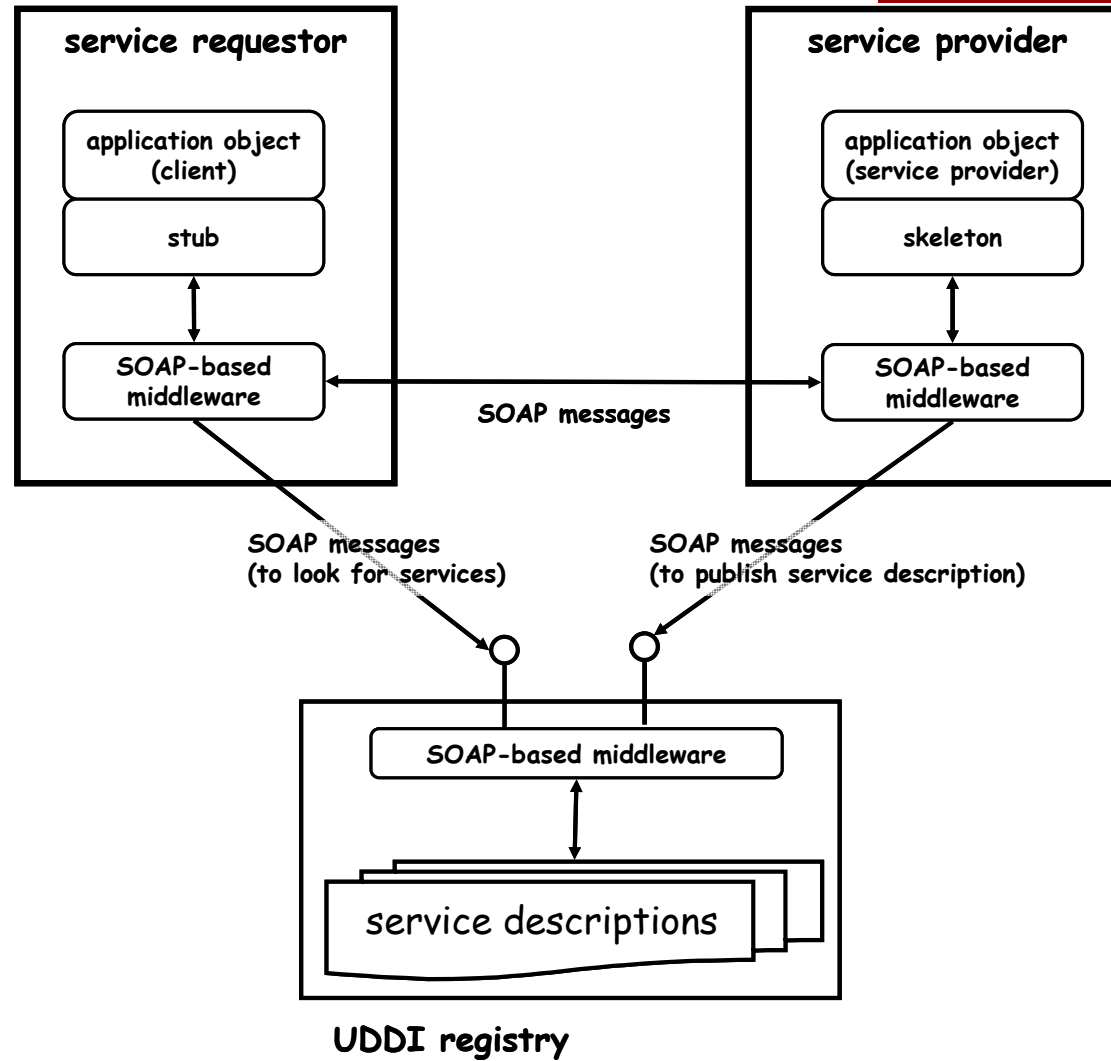
# From Interfaces to Stub/Skeleton



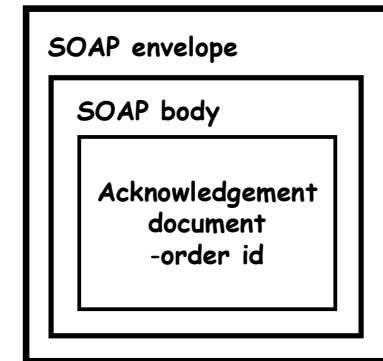
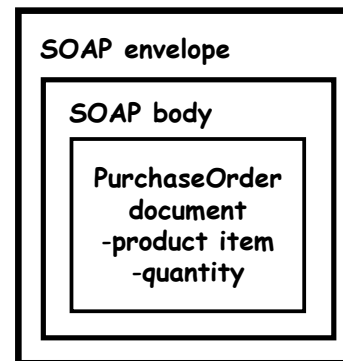
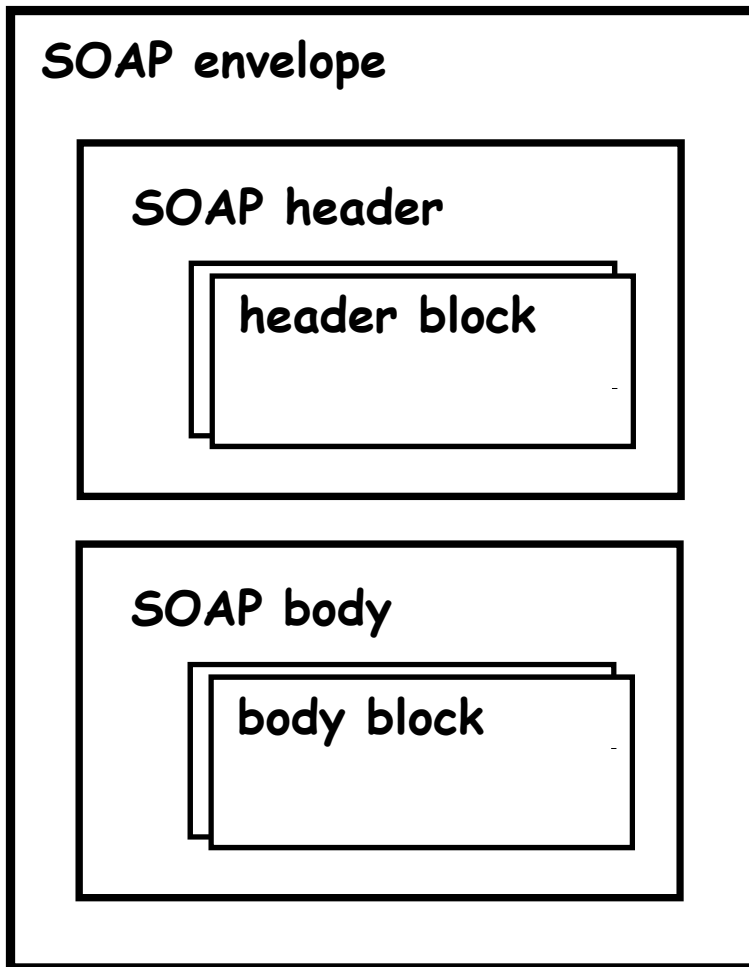
```
<operation name="orderGoods">  
  <input message = "OrderMsg"/>  
</operation>
```



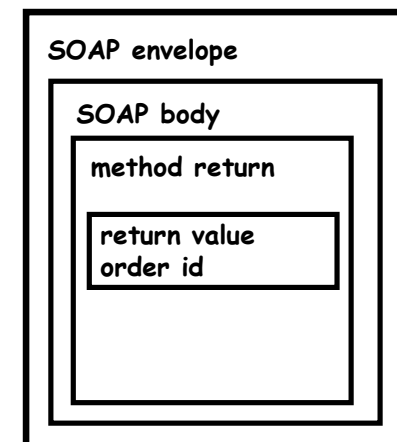
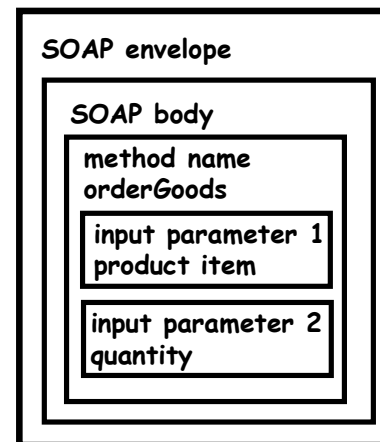
# Registry



# SOAP (1)



(a) Document-style interaction



(b) RPC-style interaction

# SOAP (2)



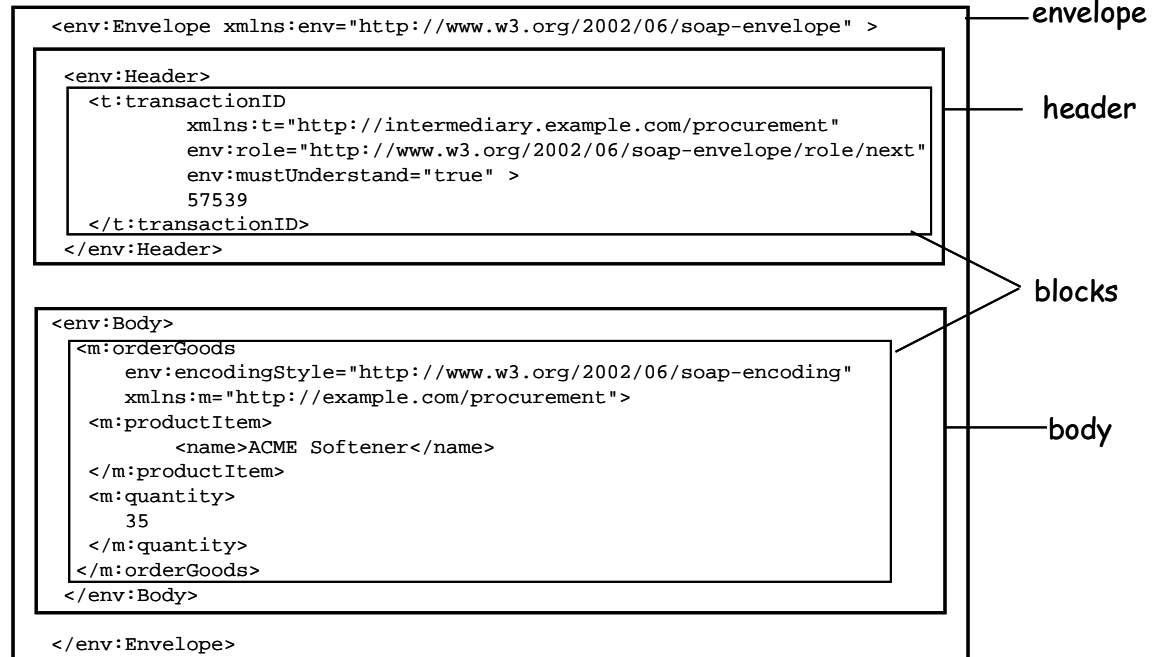
```
<ProductItem>  
  <name>...</name>  
  <type>...</type>  
  <make>...</make>  
</ProductItem>
```

```
<ProductItem  
  name="..."  
  type="..."  
  make="..."  
>
```

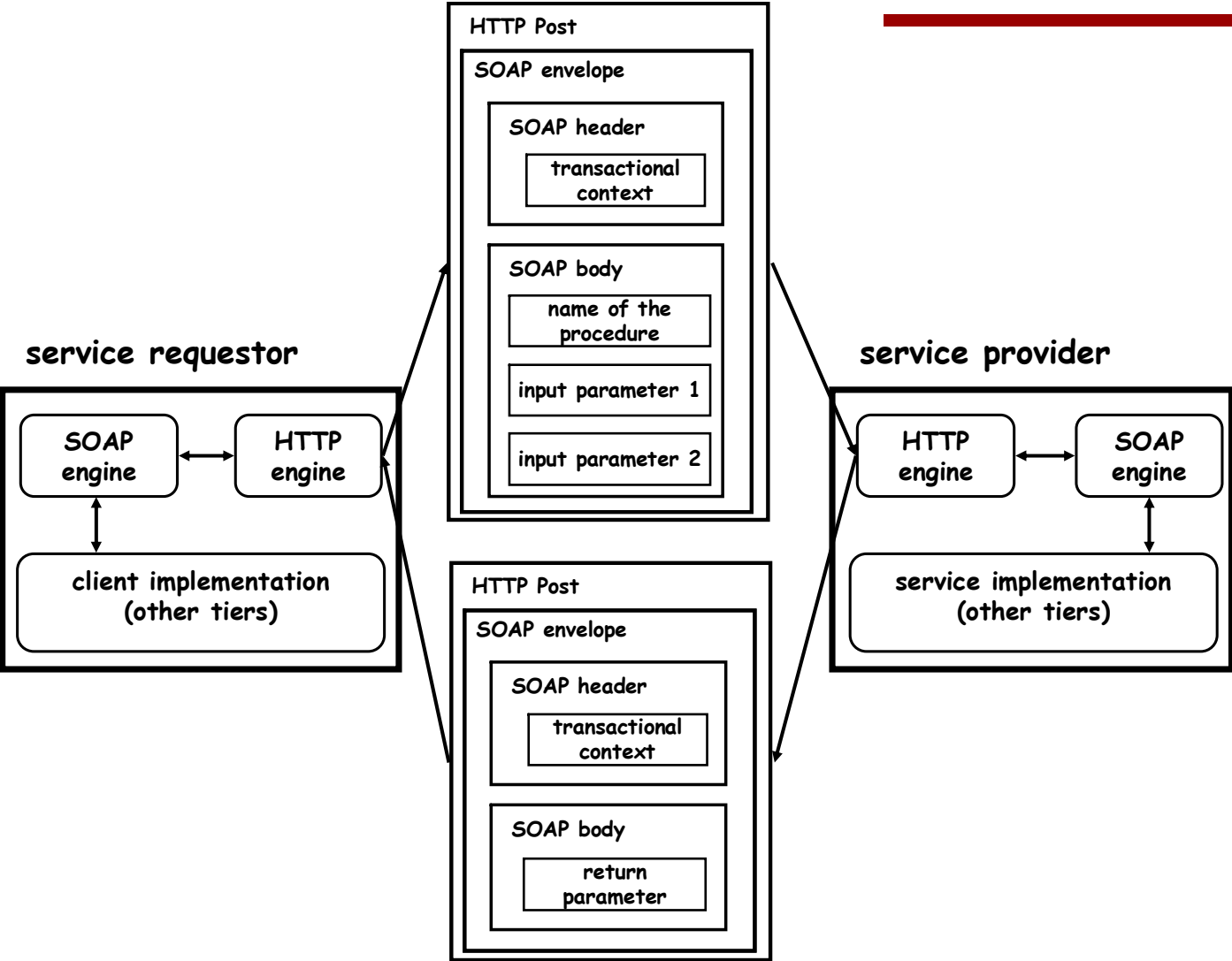
```
<ProductItem name="..."  
  <type>...</type>  
  <make>...</make>  
</ProductItem>
```

Different encoding styles

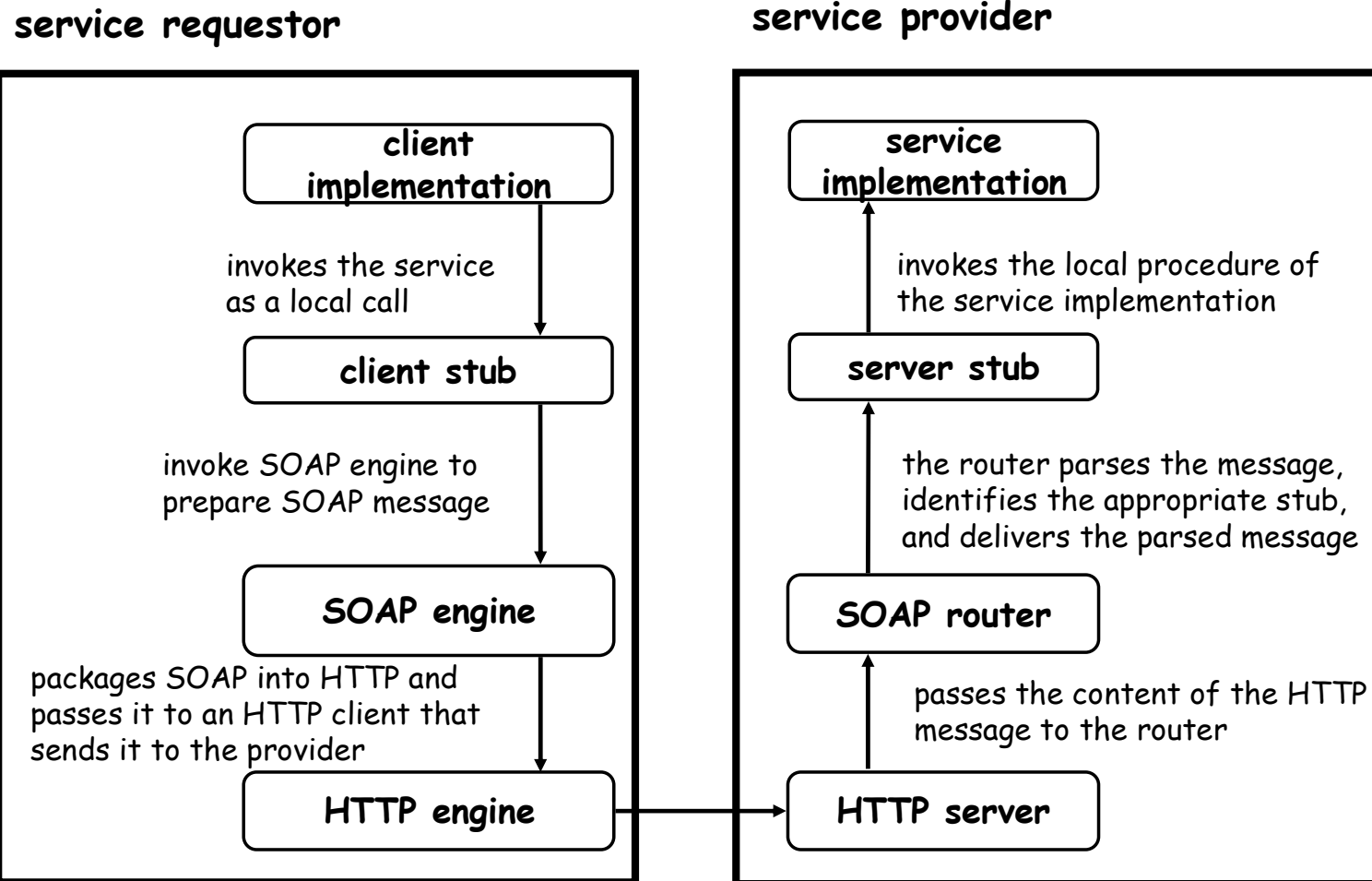
```
<?xml version='1.0' ?>
```



# RPC with SOAP

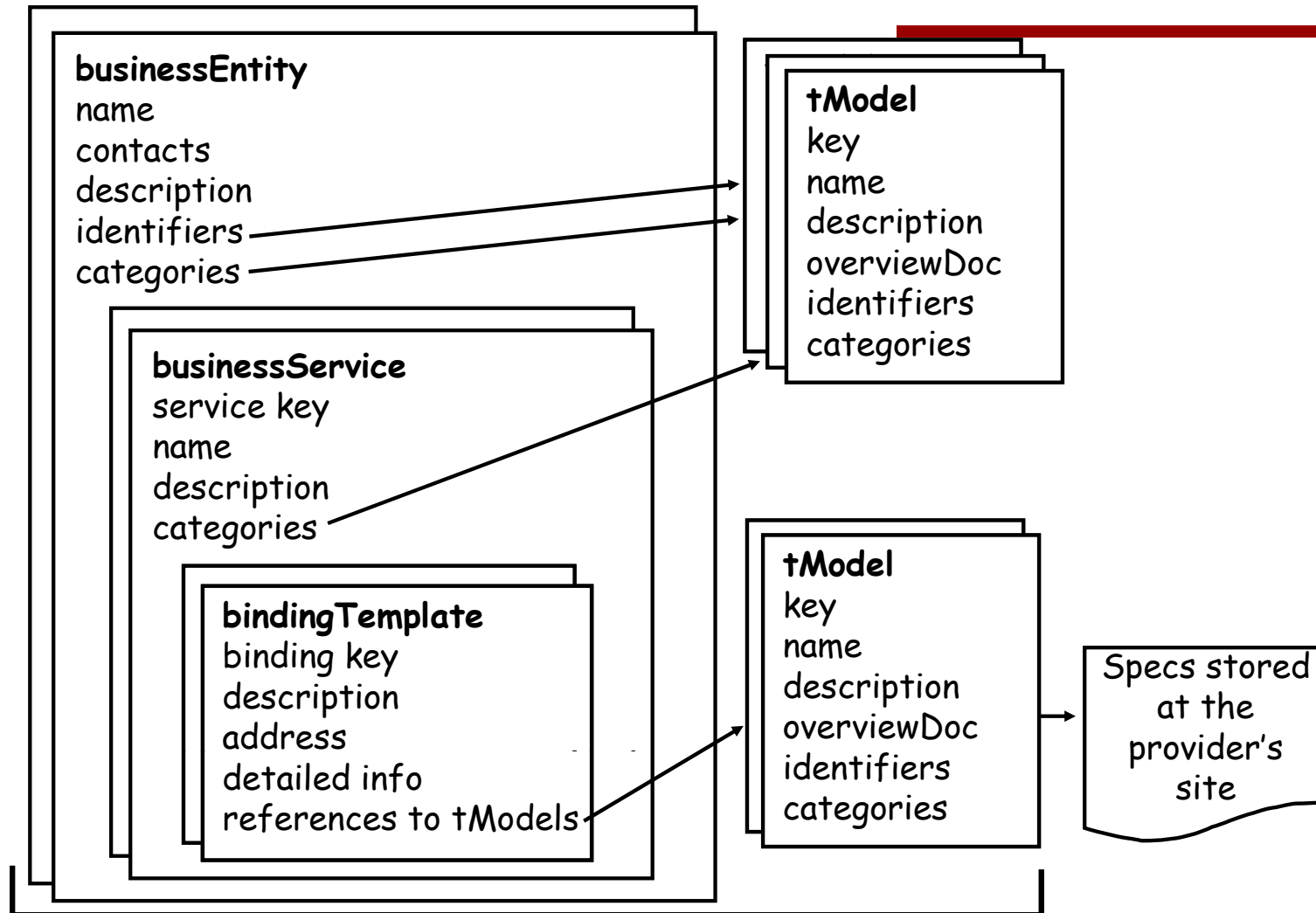


# The Simplest SOAP Middleware





# UDDI Data Structures



Stored in the UDDI registry

# A Registry Not a Repository



SAPIENZA  
UNIVERSITÀ DI ROMA

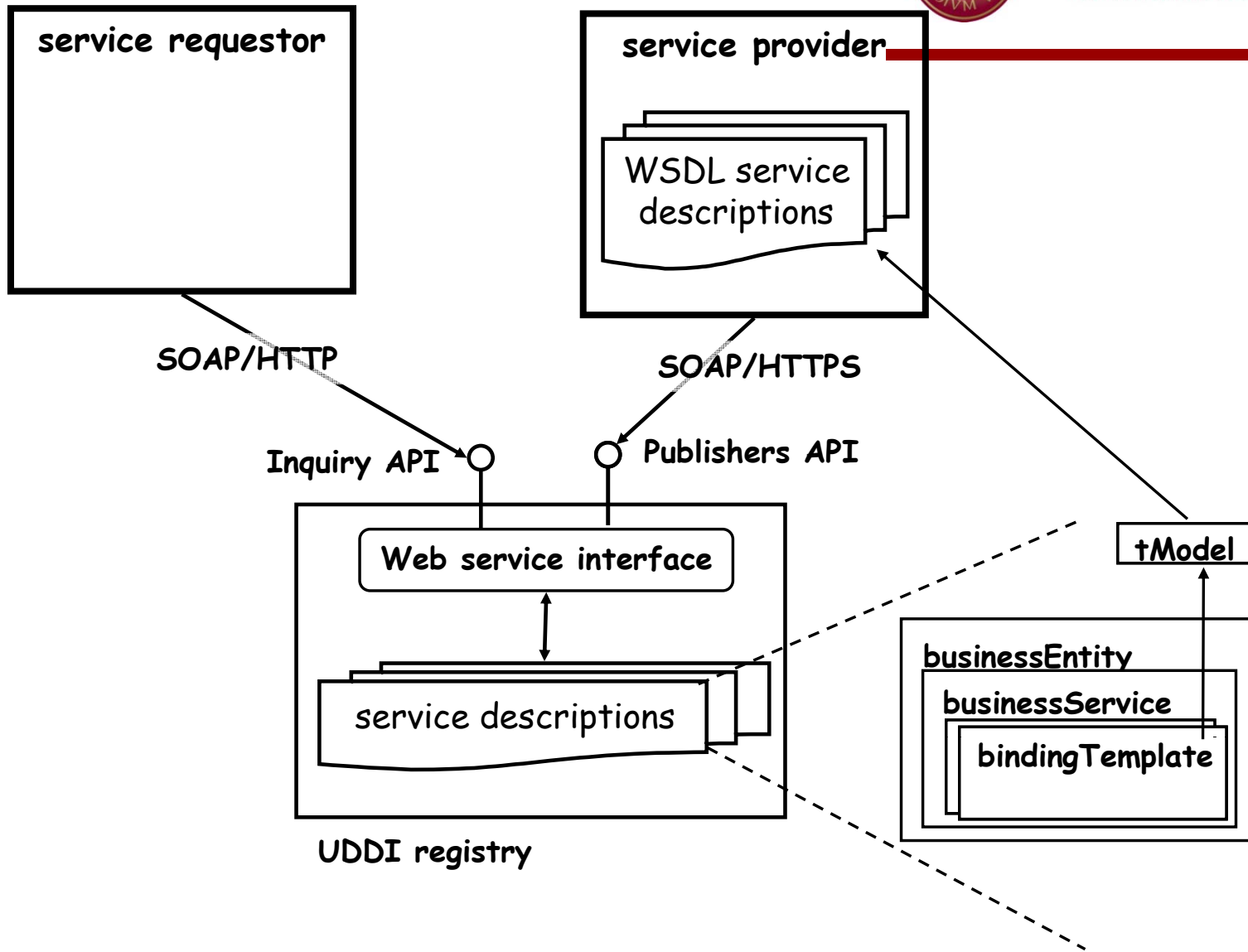
overviewDoc  
(refer to WSDL  
specs and to API  
specs)

```
<tModel tModelKey="uddi:uddi.org:v3_publication">
  <name>uddi-org:publication_v3</name>
  <description>UDDI Publication API V3.0</description>
  <overviewDoc>
    <overviewURL useType="wsdlInterface">
      http://uddi.org/wsdl/uddi_api_v3_binding.wsdl#UDDI_Publication_SoapBinding
    </overviewURL>
  </overviewDoc>
  <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#PubV3
    </overviewURL>
  </overviewDoc>
</tModel>
```

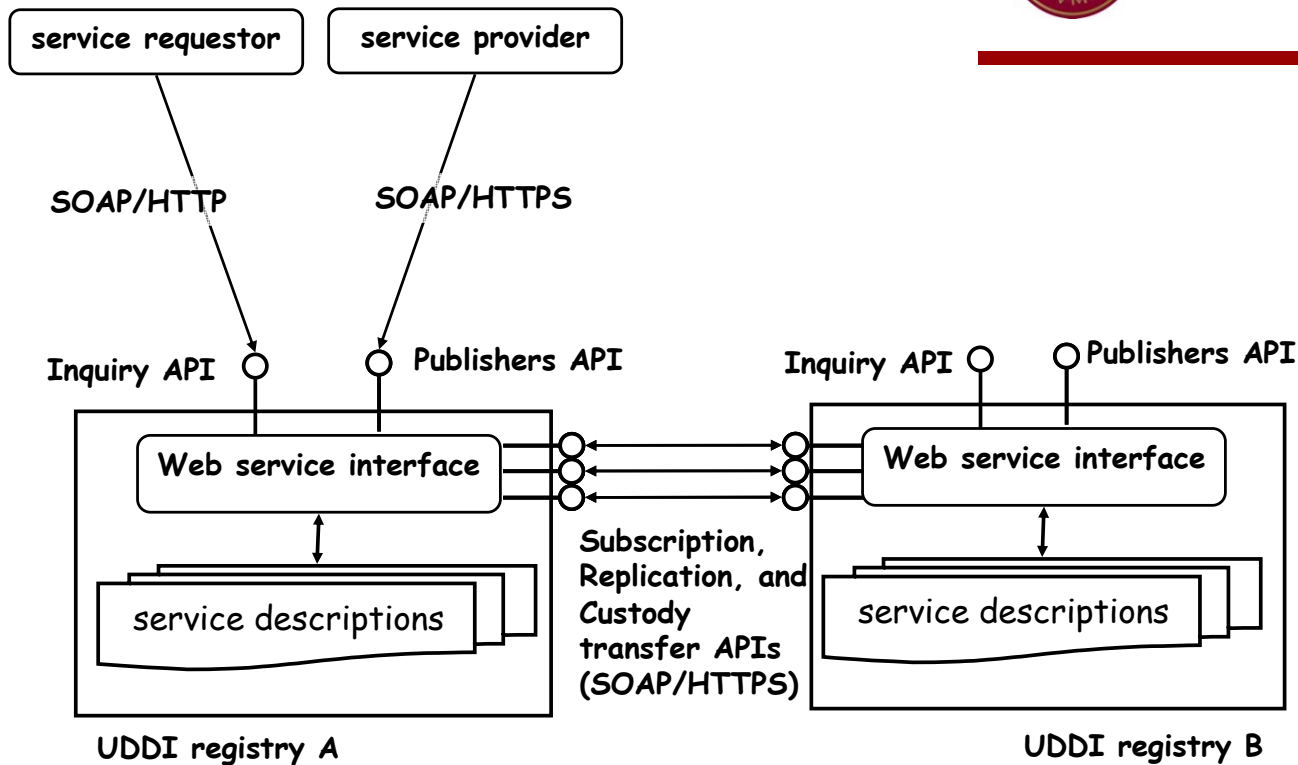
classification  
information  
(specifies that this  
tModel is about  
XML, WSDL, and  
SOAP specs)

```
<categoryBag>
  <keyedReference keyName="uddi-org:types:wsdl"
    keyValue="wsdlSpec"
    tModelKey="uddi:uddi.org:categorization:types" />
  <keyedReference keyName="uddi-org:types:soap"
    keyValue="soapSpec"
    tModelKey="uddi:uddi.org:categorization:types" />
  <keyedReference keyName="uddi-org:types:xml"
    keyValue="xmlSpec"
    tModelKey="uddi:uddi.org:categorization:types" />
  <keyedReference keyName="uddi-org:types:specification"
    keyValue="specification"
    tModelKey="uddi:uddi.org:categorization:types" />
</categoryBag>
</tModel>
```

# UDDI and WSDL



# UDDI API

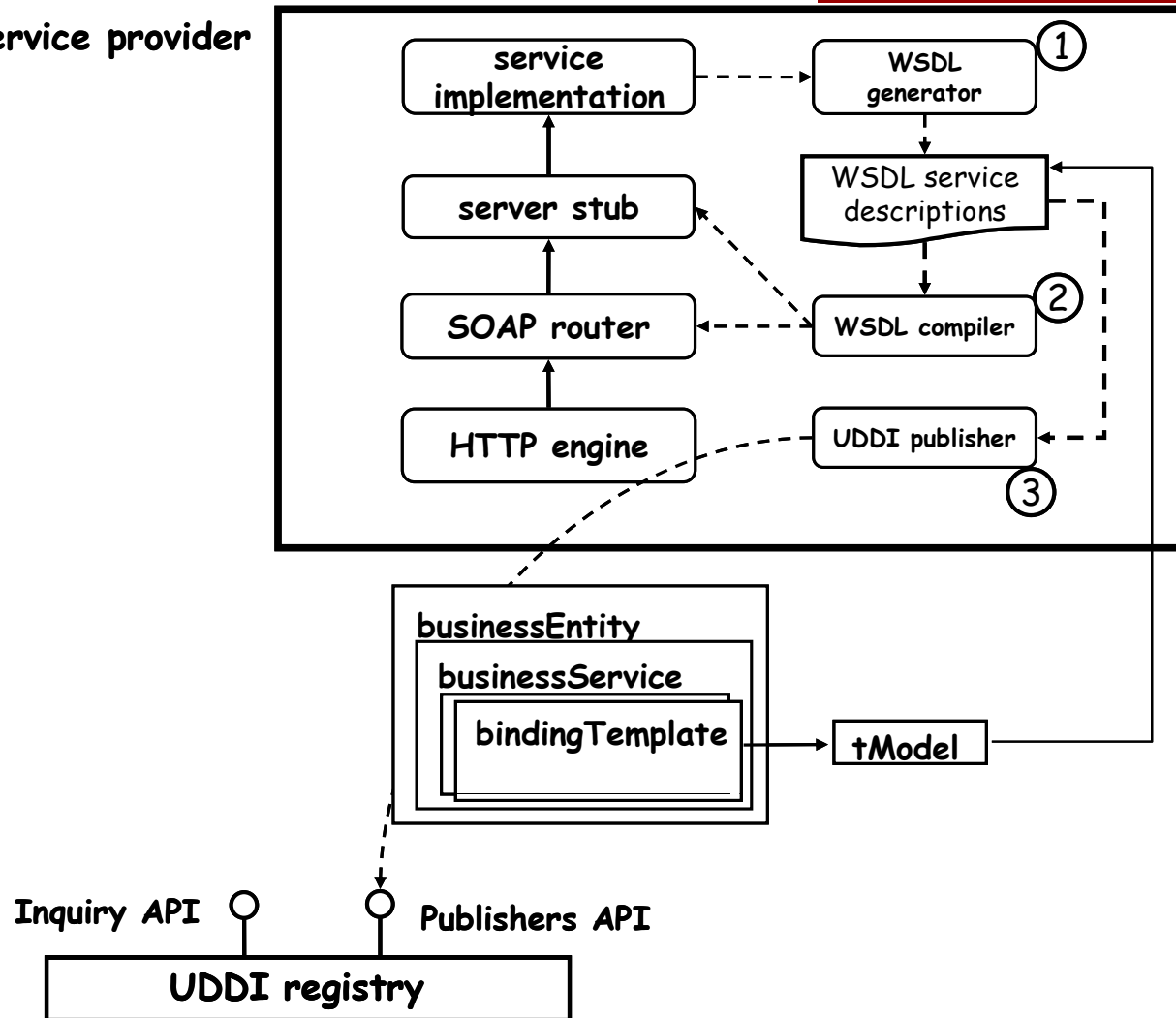


```
<?xml version="1.0"?>
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="UUID:C25893AF-1977-3528-36B5-4192C2AB9E2C"
      keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference tModelKey="UUID:A15019C5-AE14-236C-331C-650857AE0221"
      keyName="book pricing"
      keyValue="36611349"/>
  </categoryBag>
```

# Putting All Together



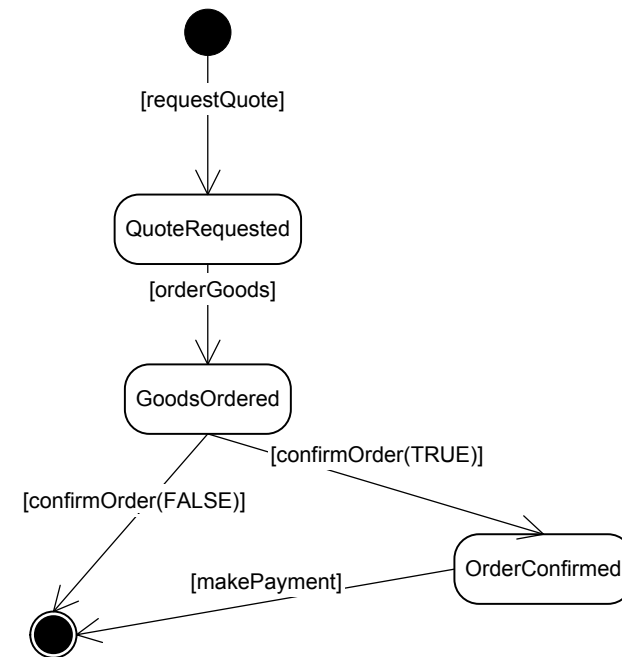
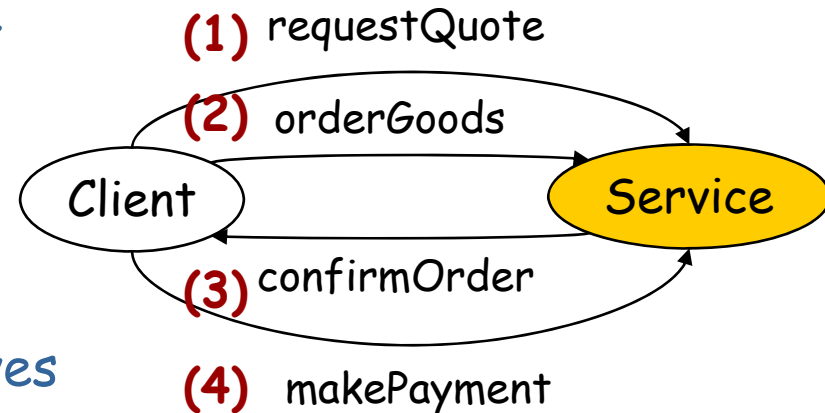
service provider



# Services



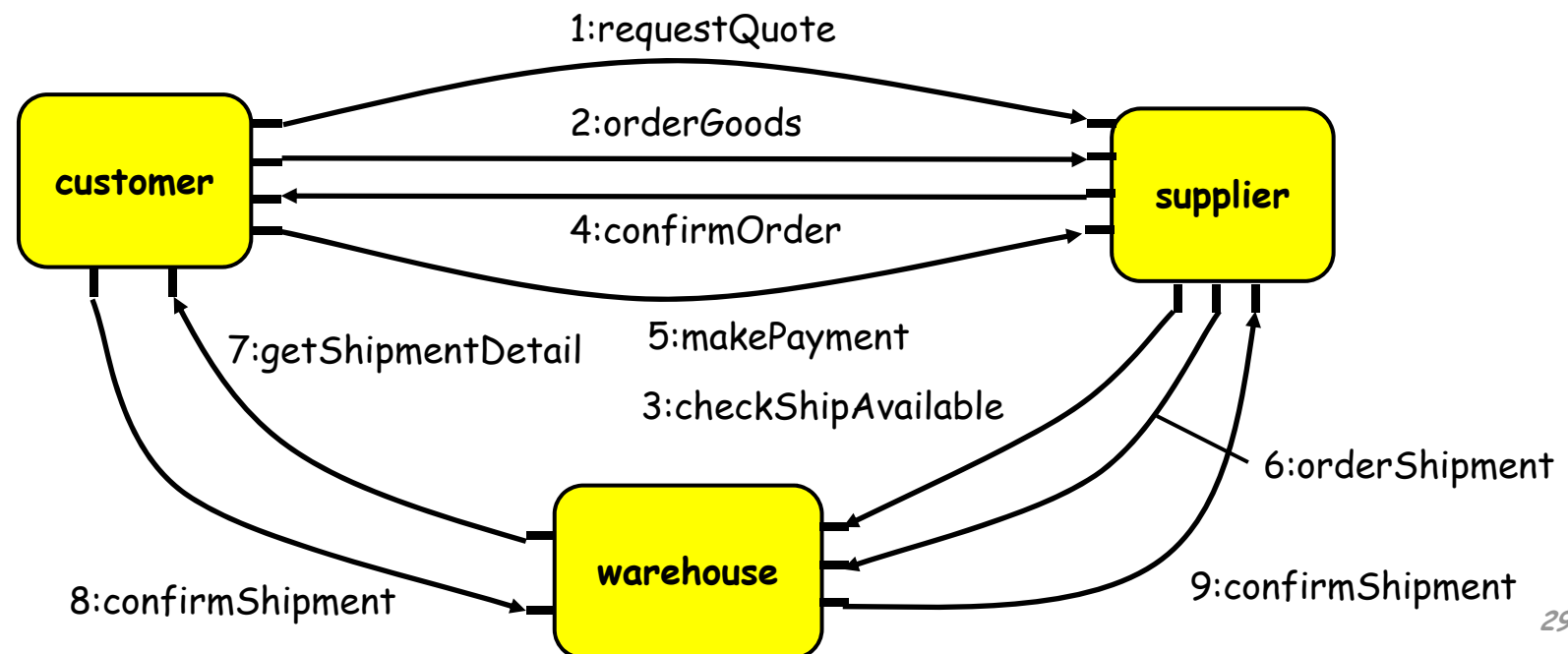
- A service is characterized by the set of (atomic) **operations** that it exports ...
- ... and possibly by constraints on the possible **conversations**
  - Using a service typically involves performing sequences of operations in a particular order (**conversations**)
  - During a conversation, the 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

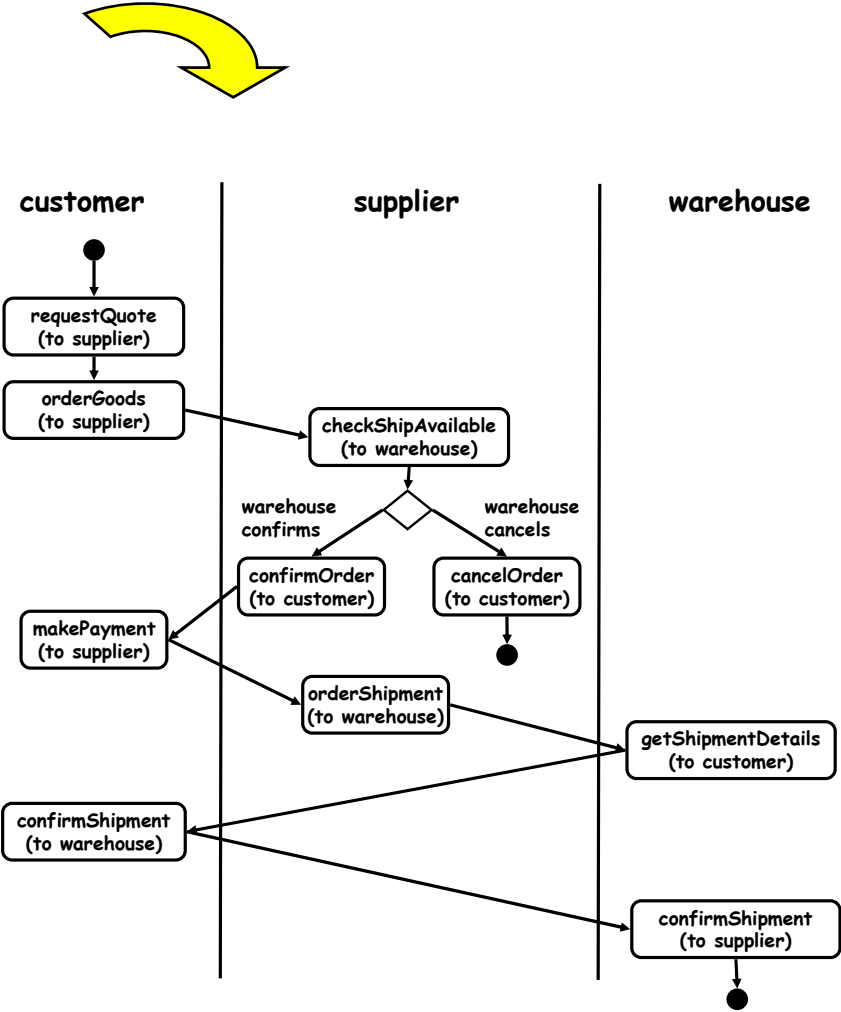
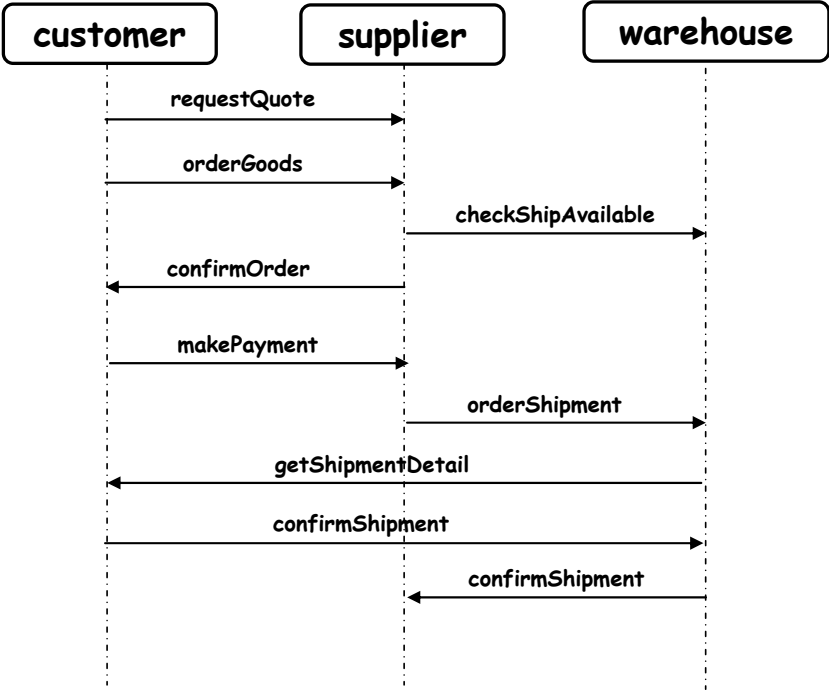


- Global specification of the conversations of N **peer** services (i.e., multi-party conversations)
  - Roles
  - Message exchanges
  - Constraints on the order in which such exchanges should occur





# Choreography: Coordination of Conversations of N Services



[from ACKM04]

# Composition



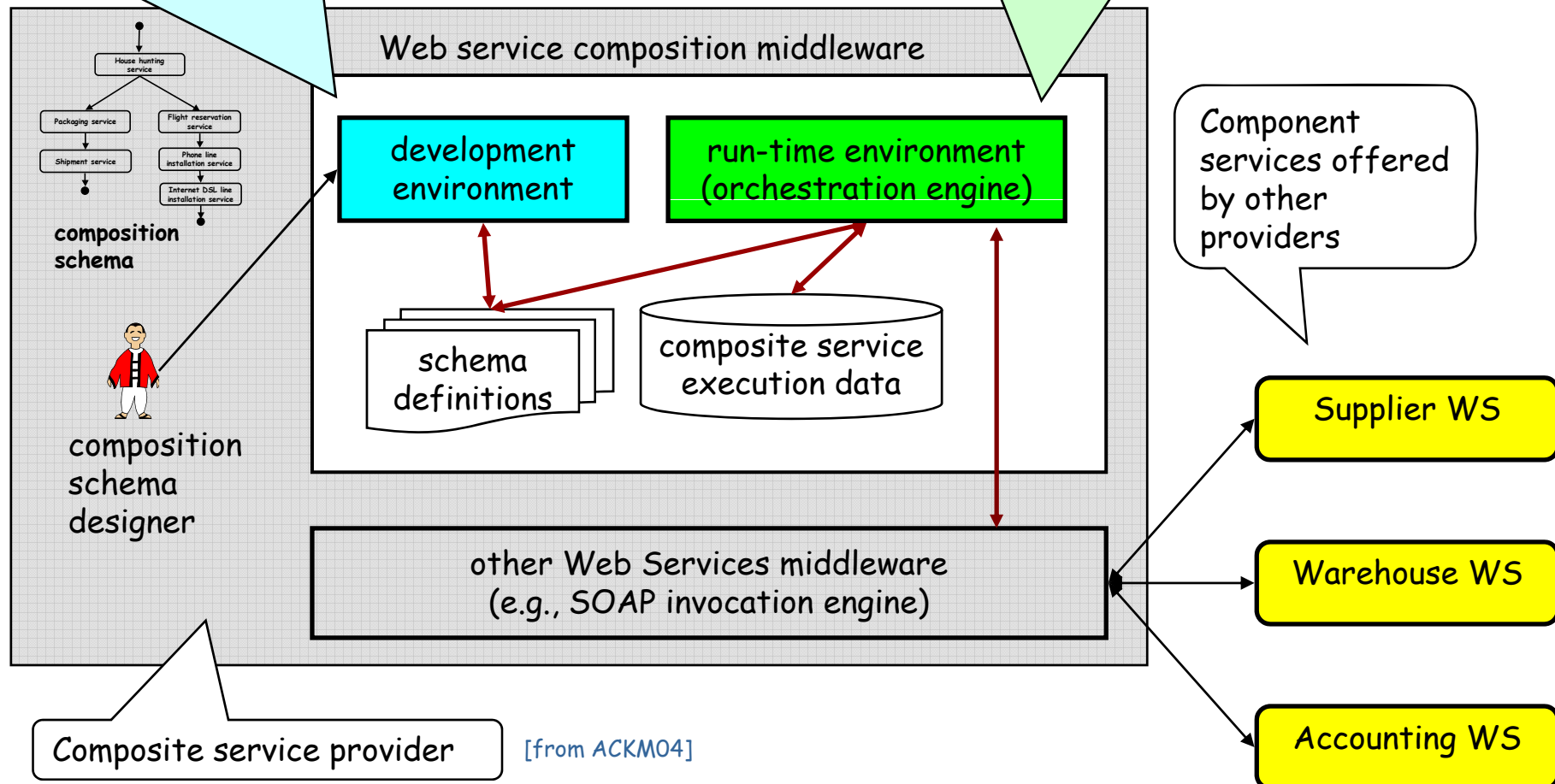
- Deals with the **implementation** of an application (in turn offered as a service) whose application logic **involves the invocation of operations** offered by other **services**
  - The new service is the *composite service*
  - The invoked services are the *component services*

# 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*



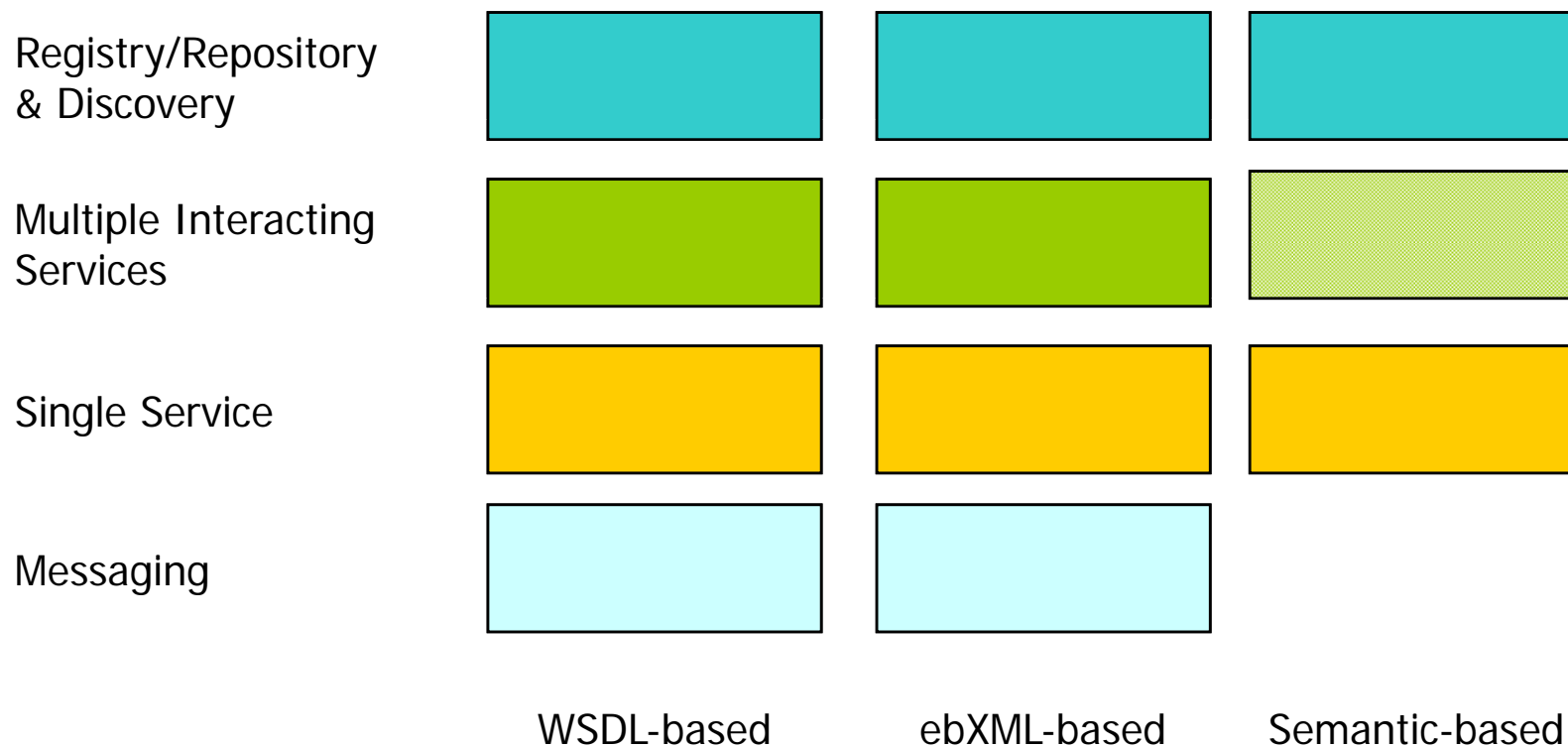
- A composition schema specifies the “process” of the composite service
  - The “workflow” of the service
- Different clients, by interacting with the composite service, satisfy their specific needs (reach their goals)
  - A specific execution of the composition schema for a given client is an orchestration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

# The "Stacks" of Service Technologies



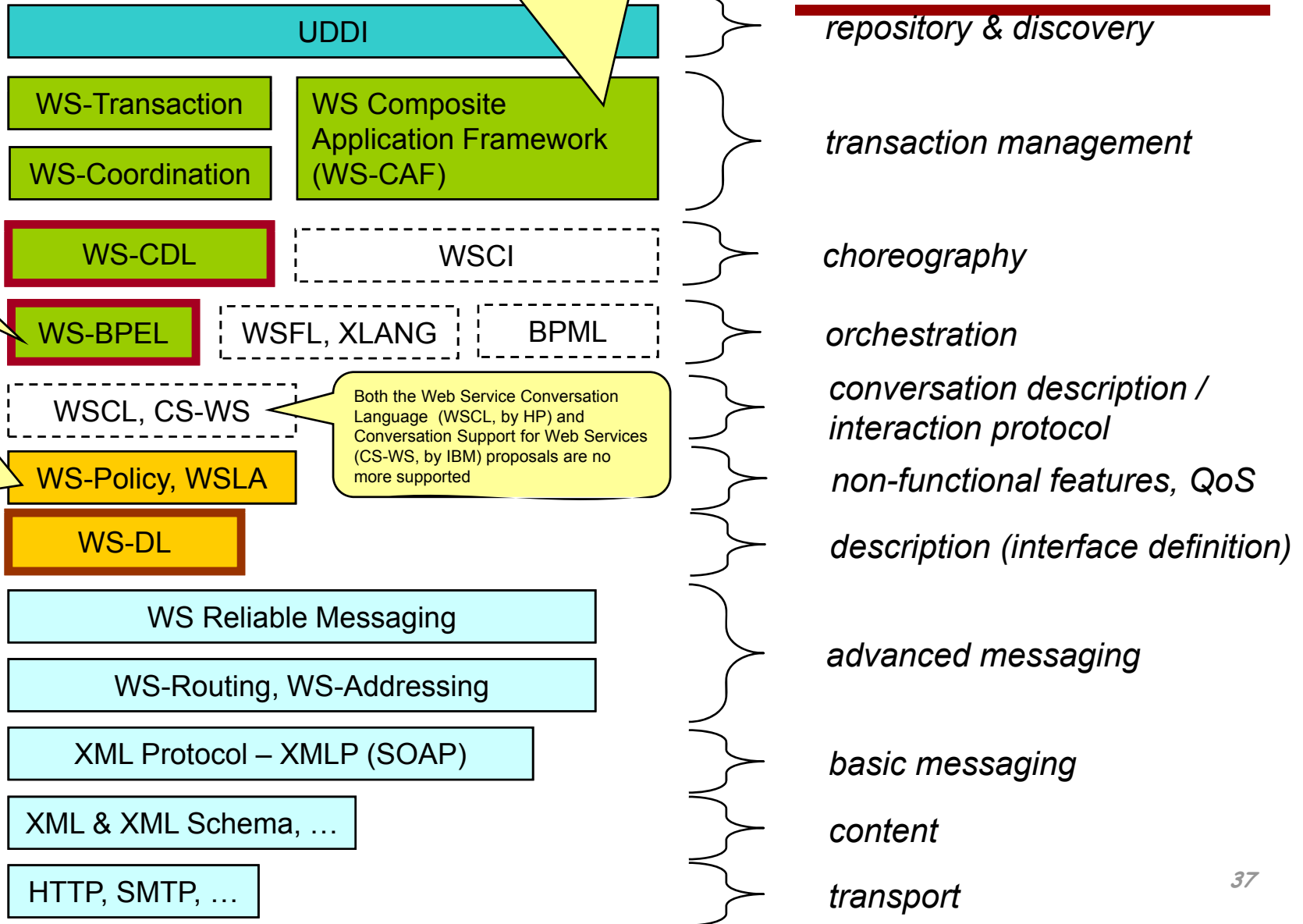
# The WSDL-based "Stack"

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



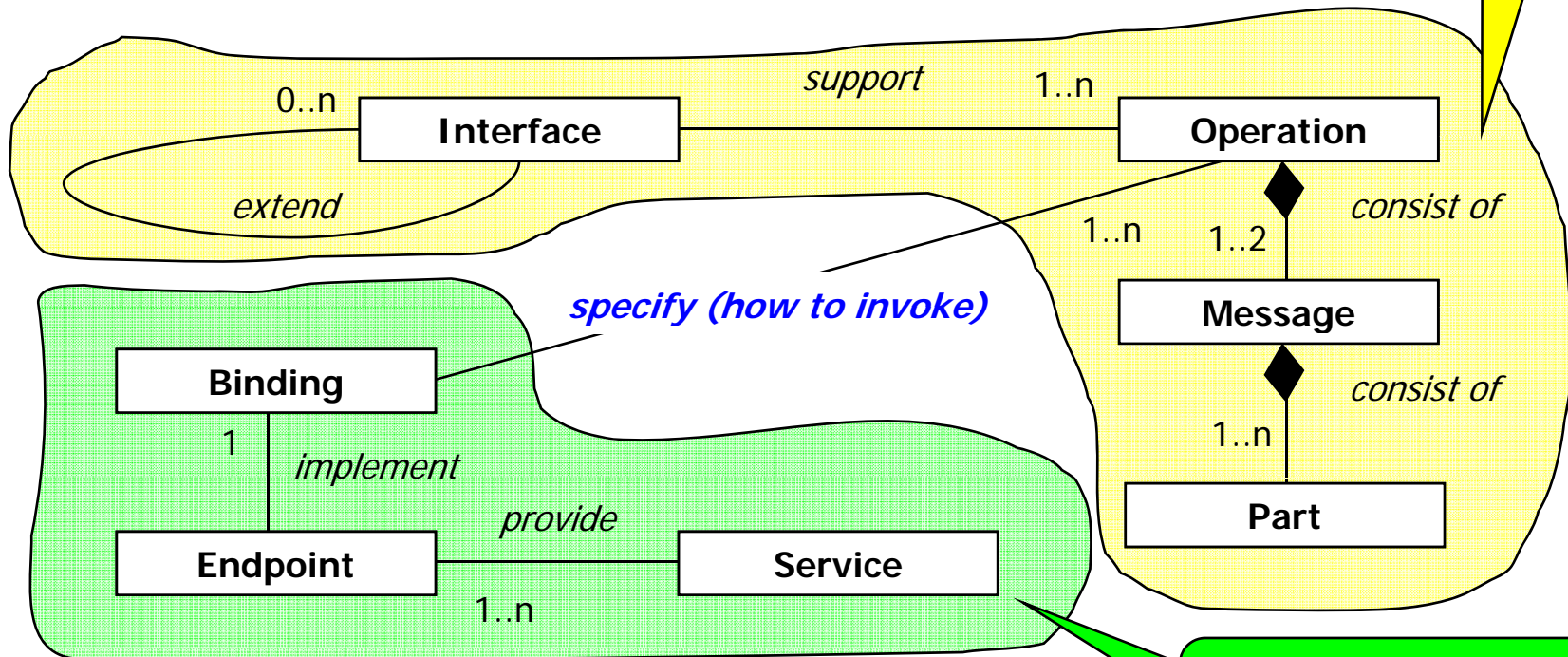


# Web Service Definition Language (WS-DL)



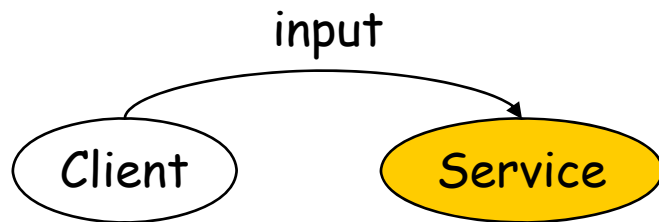
- WS-DL (v2.0) provides a framework for defining
  - Interface: operations and input/output formal parameters
  - Access specification: protocol bindings (e.g., SOAP)
  - Endpoint: the location of service

Service interface  
(abstract definition)

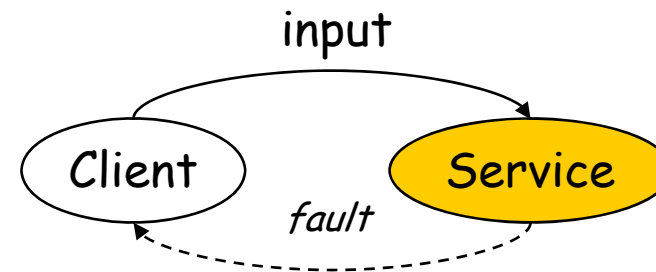


Service implementation  
(concrete definition)

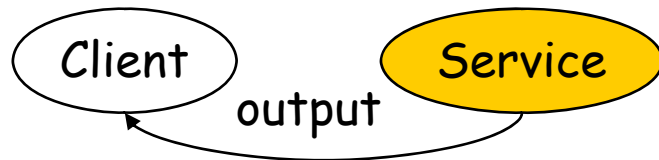
# Message Exchange Patterns (1)



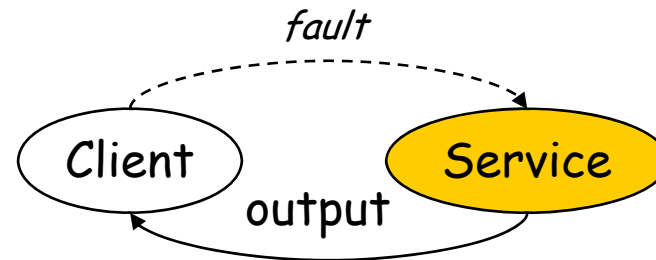
*in-only (no faults)*



*robust in-only (message triggers fault)*

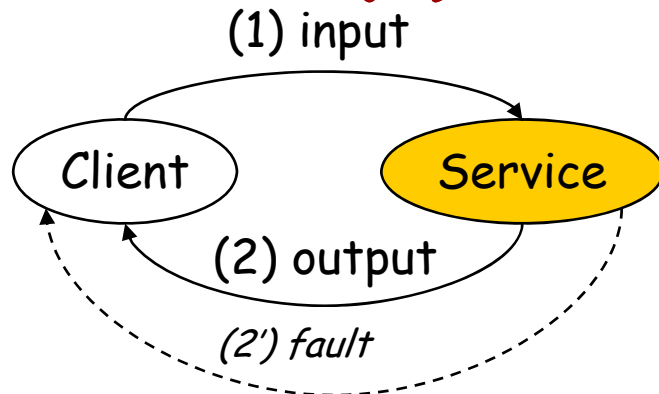


*out-only (no faults)*

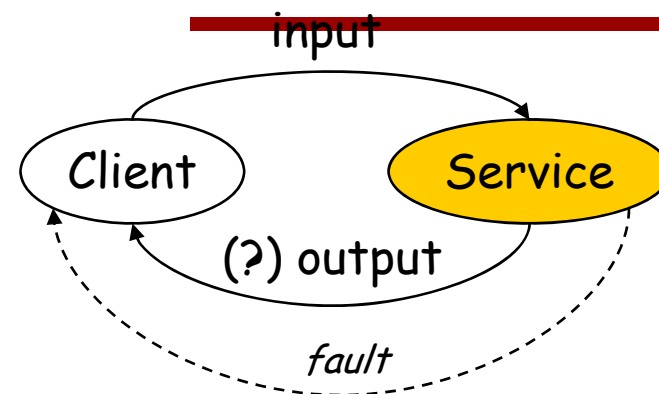


*robust out-only (message triggers fault)*

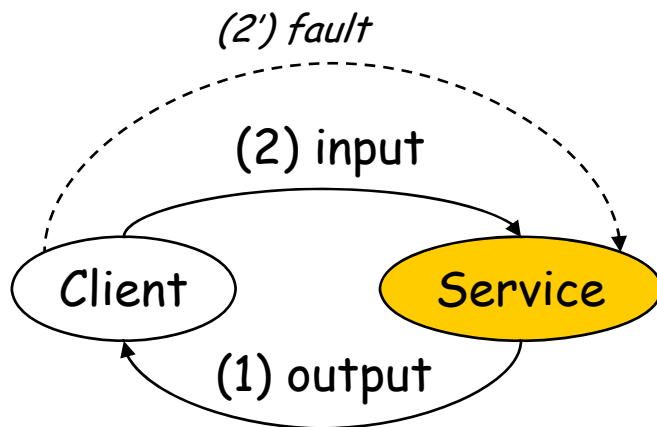
# Message Exchange Patterns (2)



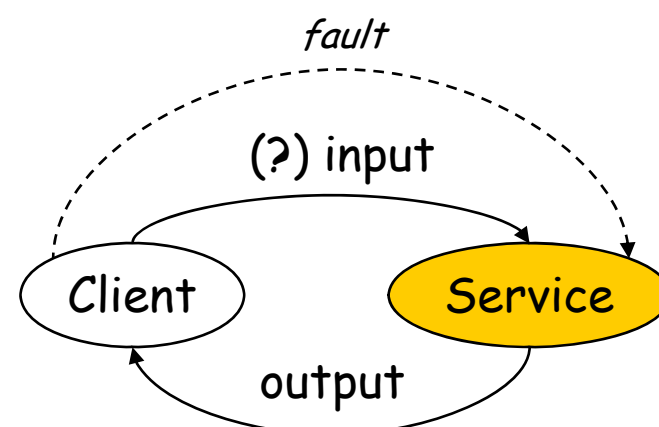
*in-out (fault replaces message)*



*in-optional-out  
(message triggers fault)*



*out-in (fault replaces message)*



*out-optional-in  
(message triggers fault)*

# An Example (1)



Definition of a message and its formal parameters

```
<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>
```

# 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

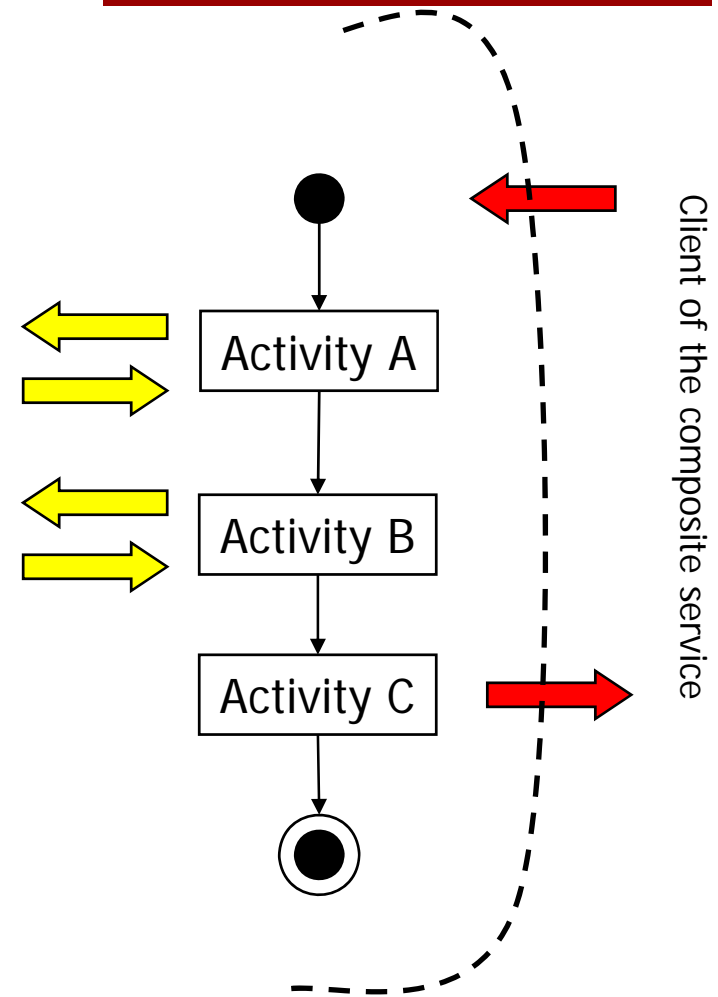
```
<interface name="MP3ServiceType">
  <operation name="search_by_title" pattern="in-out">
    <input message="SearchByTitleRequest"/>
    <output message="SearchByTitleResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
  <operation name="search_by_author" pattern="in-out">
    <input 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

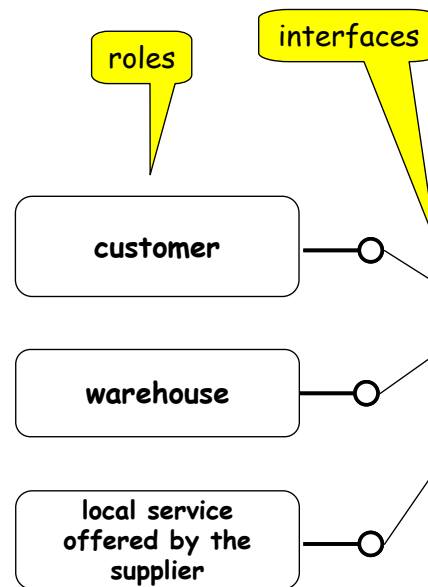


An XML document specifying

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

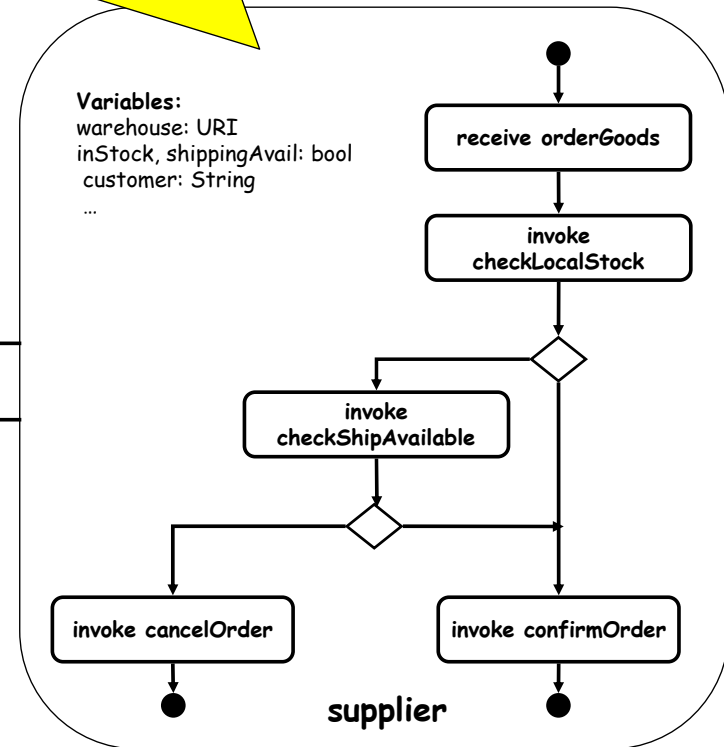
• The orchestration of the process

- 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)*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- 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)*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- 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)*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- 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)

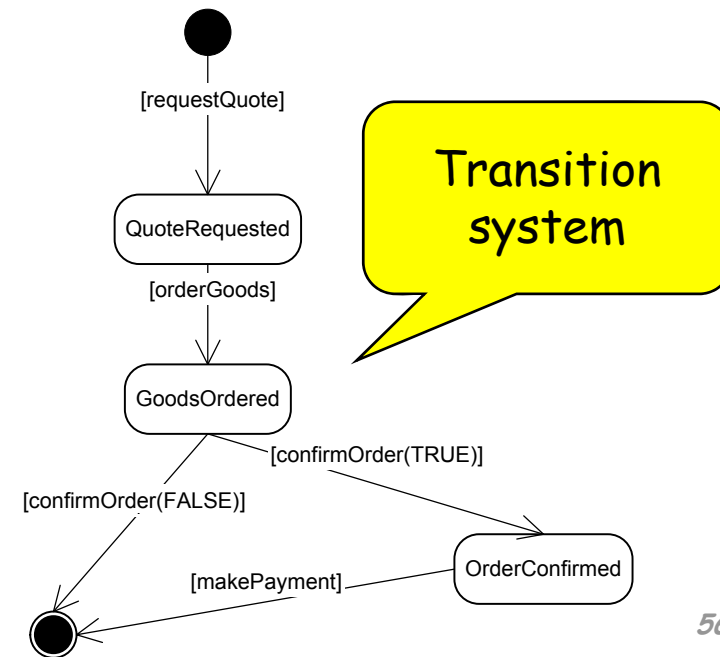
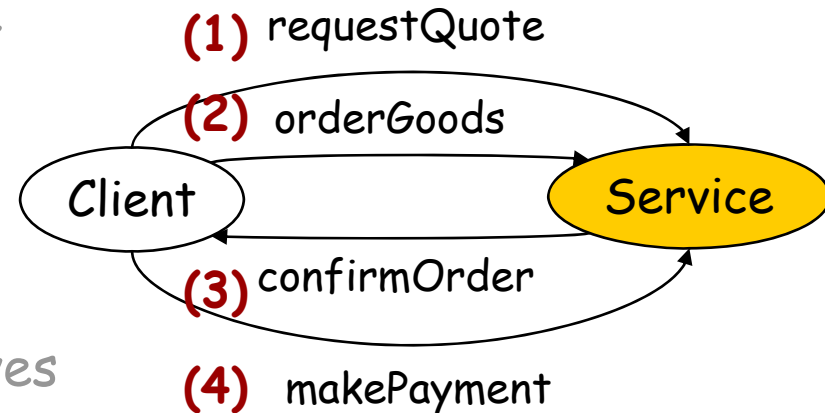


- A Choreography combines all previous elements, forming a collaboration unit of work
  - Enumerate all the binary relationships interactions act in
  - Localize the visibility of variables
    - Using variable definitions
  - Prescribe alternative patterns of behavior
    - Using work/units and reactions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

# Services



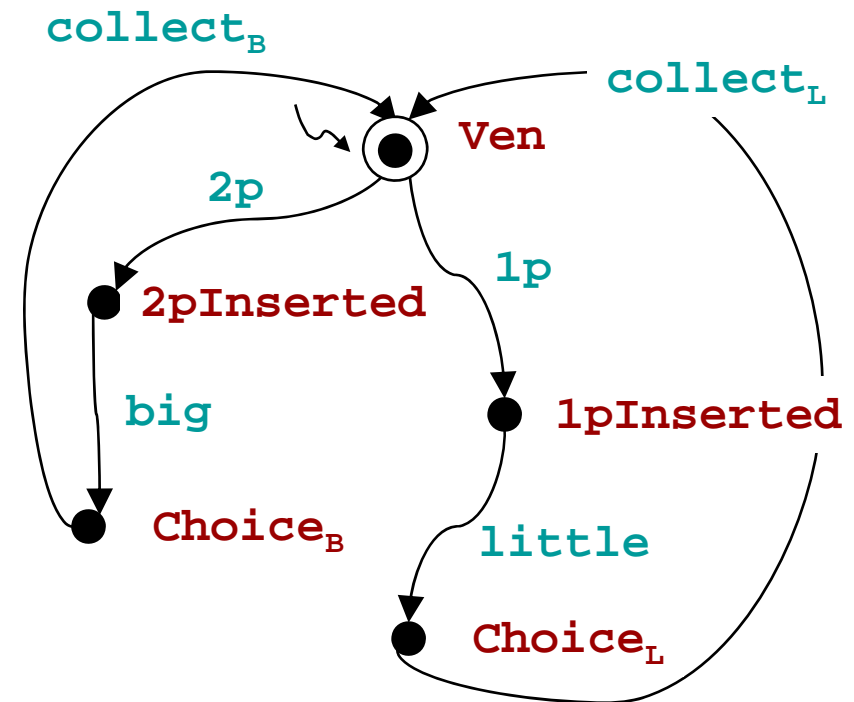
- A service is characterized by the set of (atomic) operations that it exports ...
- ... and possibly by constraints on the possible **conversations**
  - Using a service typically involves performing sequences of operations in a particular order (conversations)
  - During a conversation, the client typically chooses the next operation to invoke (on the basis of previous results, etc.) among the ones that the service allows at that point



# Transition Systems



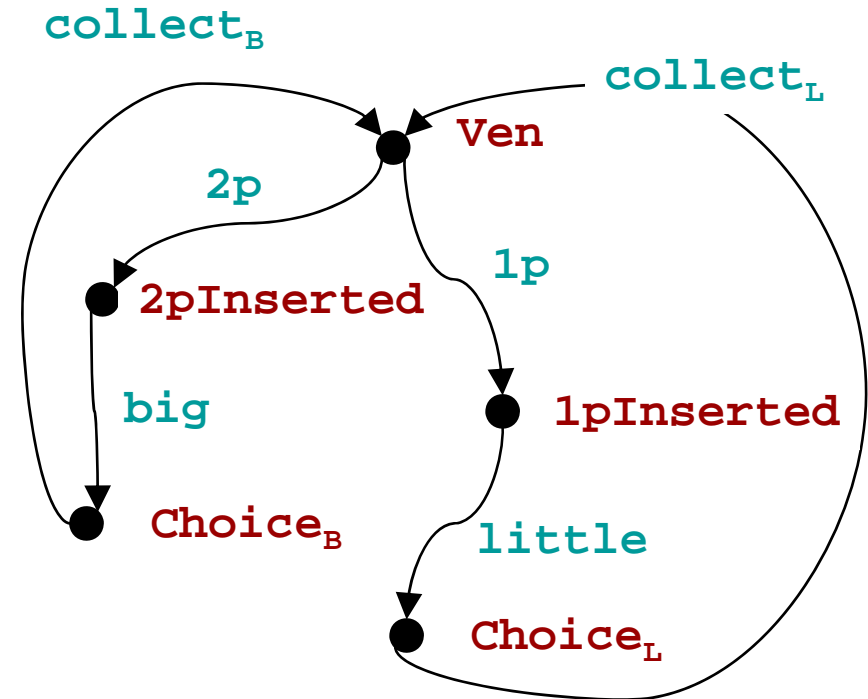
- A transition system (TS) is a tuple  $T = \langle A, S, S^0, \delta, F \rangle$  where:
  - $A$  is the set of actions
  - $S$  is the set of states
  - $S^0 \in F$  is the initial state
  - $\delta \subseteq S \times A \times S$  is the transition relation
  - $F \subseteq S$  is the set of final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



# Process Algebras and TSs



- Process theory:
  - a process is a term of an algebraic language
  - a transition  $E \xrightarrow{a} F$  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

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

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

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

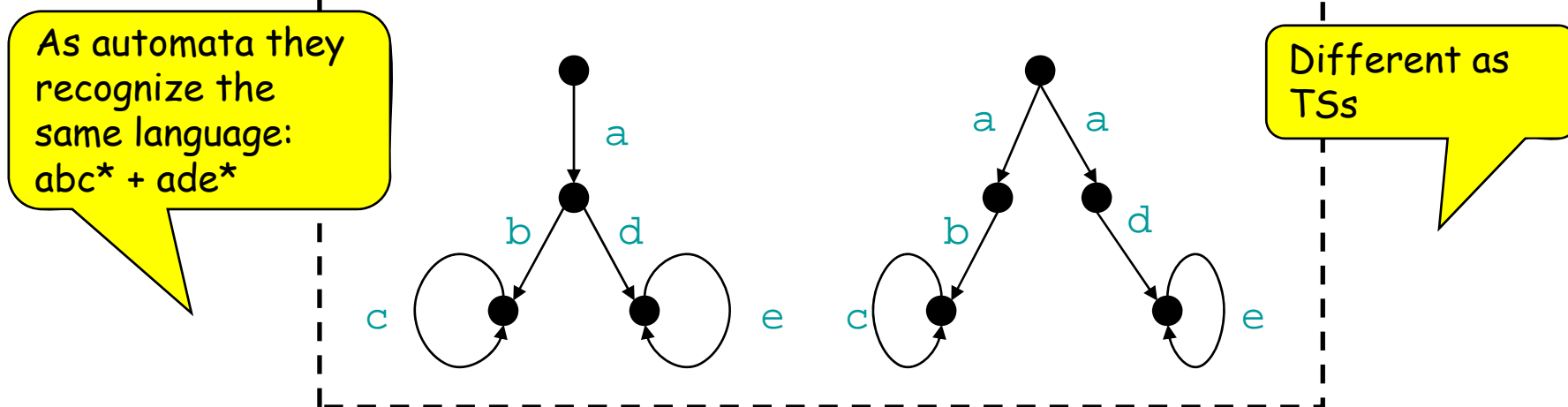
$$Choice_B = \text{collect}_B.\text{Ven}$$

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

# 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"



# *WS-DL is the Set of Actions*

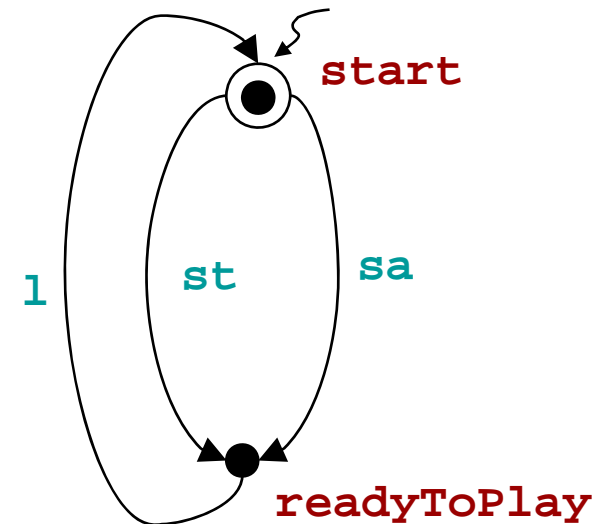


- A message exchange pattern (and the related operation) represents an **interaction** with the service client
  - an action that the service can perform by interacting with its client
- Abstracting from formal parameters, we can associate a different symbol to each operation ...
- ... thus obtaining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

# An Example



- The MP3ServiceInterface defines 3 actions:
  - search\_by\_title / st
  - search\_by\_author / sa
  - listen / l
  
- Formally  $A = \{st, sa, l\}$



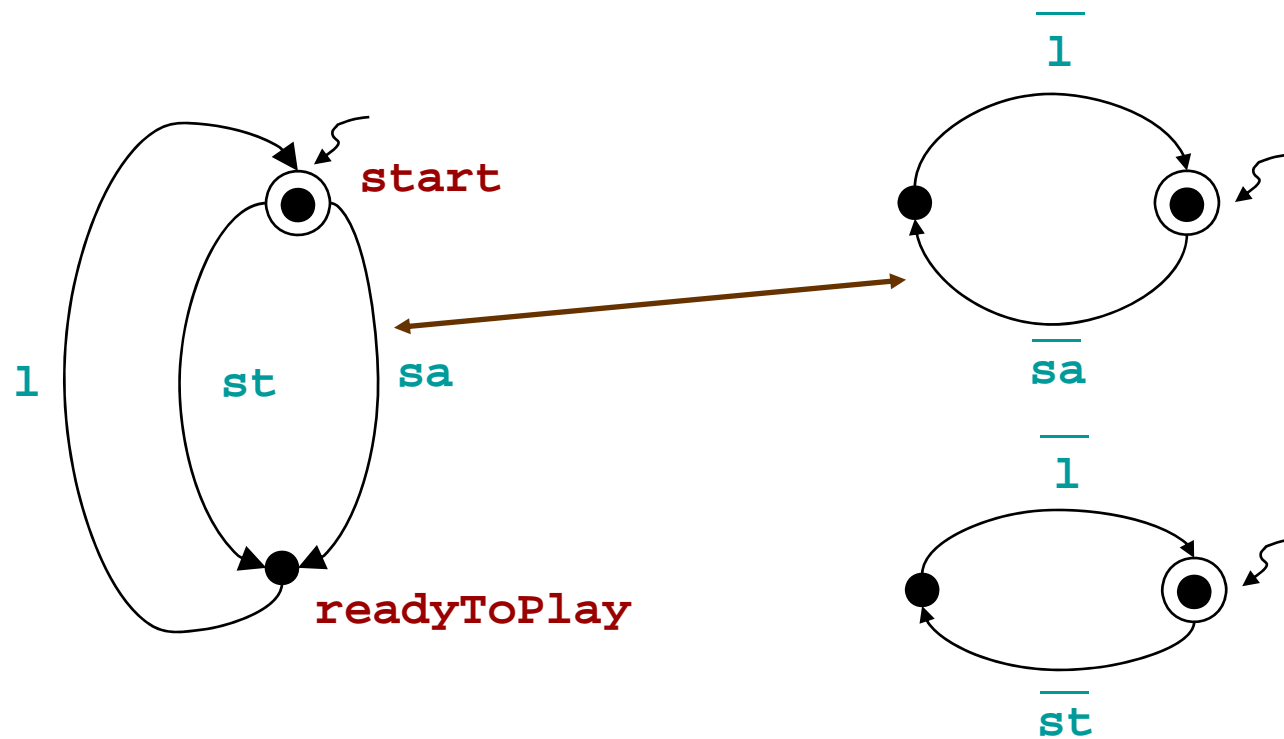


# TSs and Choreography

(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





SAPIENZA  
UNIVERSITÀ DI ROMA

---

## *Lecture 4*

1. State of the Art on Automatic Composition



client

target service invocation

functional requirements of the target service

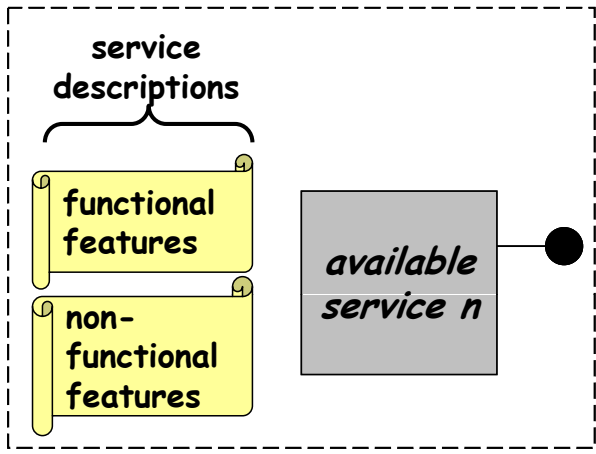
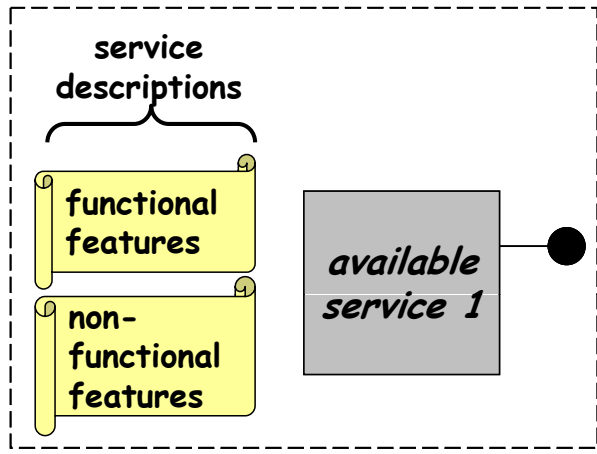
non-functional requirements of the target service

Synthesis

specification of the process of the composite service

additional requirements for orchestration

Orchestration  
Monitoring



# 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 ?

# *Service description*



- Services export a **view of their behavior**

- I/O interface

- Data Access

- focus on data
- for information gathering

- Atomic Actions

- focus on actions
- world altering services

information  
oriented services

services as  
atomic actions

services as  
processes

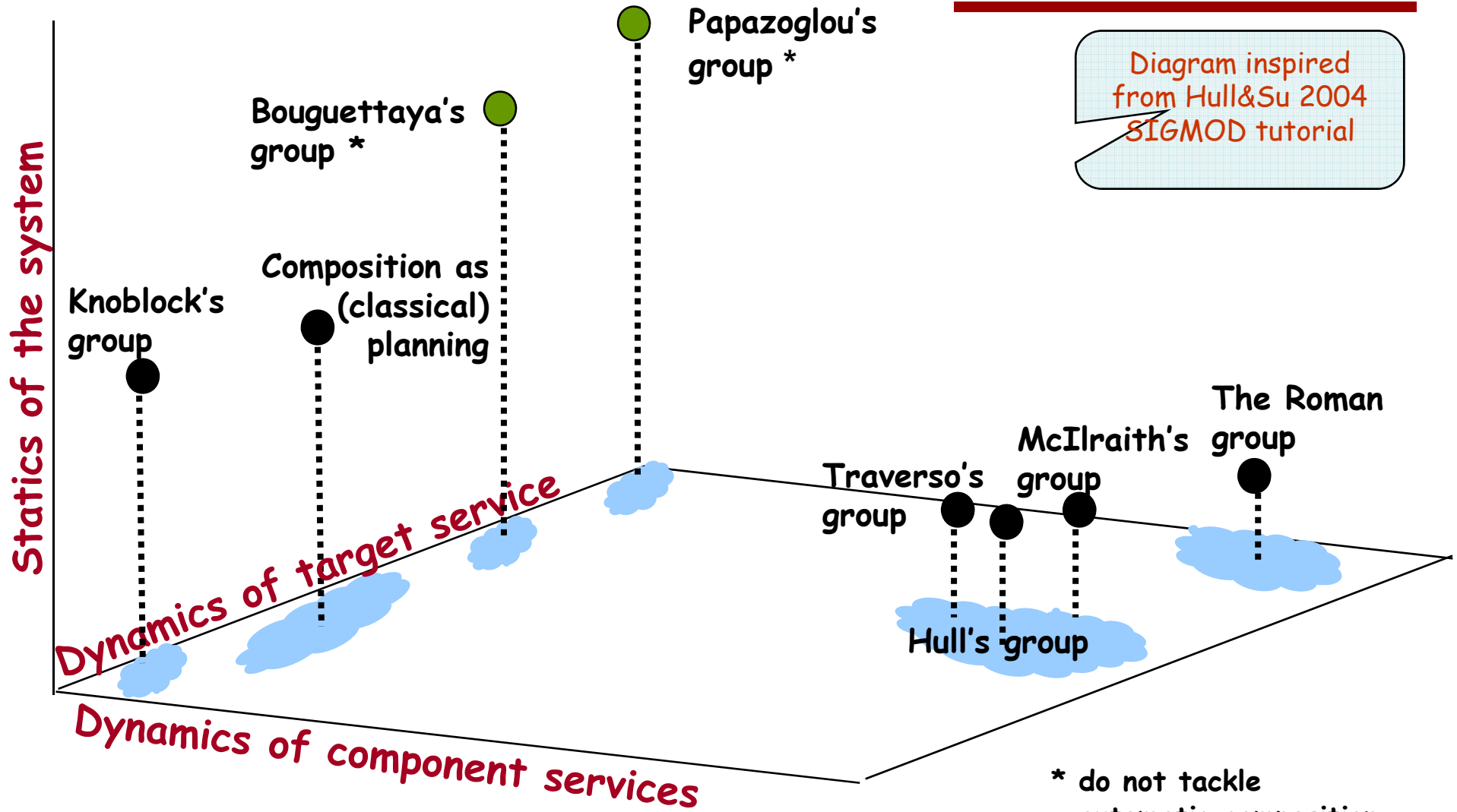
- Complex Behavioral Description

(typically represented using finite states, e.g., TSs)

# The whole picture



Diagram inspired from Hull&Su 2004 SIGMOD tutorial



\* do not tackle automatic composition

# *Key dimensions in service composition (1)*



1. **Statics of the composition system (i.e., static semantics):**
  - e.g, ontologies of services (for sharing semantics of data/information), inputs and outputs, etc.
2. **Dynamics of component services (i.e., dynamic semantics, process):**
  - e.g., behavioral features, complex forms of dataflow, transactional attitudes, adaptability to varying circumstances

# Key dimensions in service composition (2)



## 3. Dynamics of the target service (i.e., dynamic semantics, process)

The target service exposed as:

atomic  
action



- single step
- (set of) sequential steps
- (set of) conditional steps
- while/loops, running batch
- while/loops, running under an external control

process



## *Key dimensions in service composition: the 4<sup>th</sup> dimension*



### 4. Degree of (in)completeness in the specification of:

- Static Aspects (of the composition system)
- Dynamic Aspects (of component services)
- Target service specification

For simplicity not shown in the following slides

- Note: Orthogonal to previous dimensions

# *What is addressed from the technical point of view?*



- Automatic composition techniques?
  - Which formal tools?
  - Sound and complete techniques?
  - Techniques/Problem investigated from computational point of view?



## Analyzed works

- Papazoglou's group
  - Bouguettaya's group
  - Knoblock's group
  - Composition as Planning
  - Traverso's group
  - McIlraith's group
  - Hull's group
  - The Roman group
- (not automatic composition)
- (information oriented services)
- (services as atomic actions)
- (services as processes)

as called by Rick Hull  
in his SIGMOD 2004  
tutorial

# Papazoglou's group

*J. Yang and M.P. Papazoglou: Service Components for Managing the Life-cycle of Service Compositions, Information Systems 29 (2004), no. 2, 97 - 125*



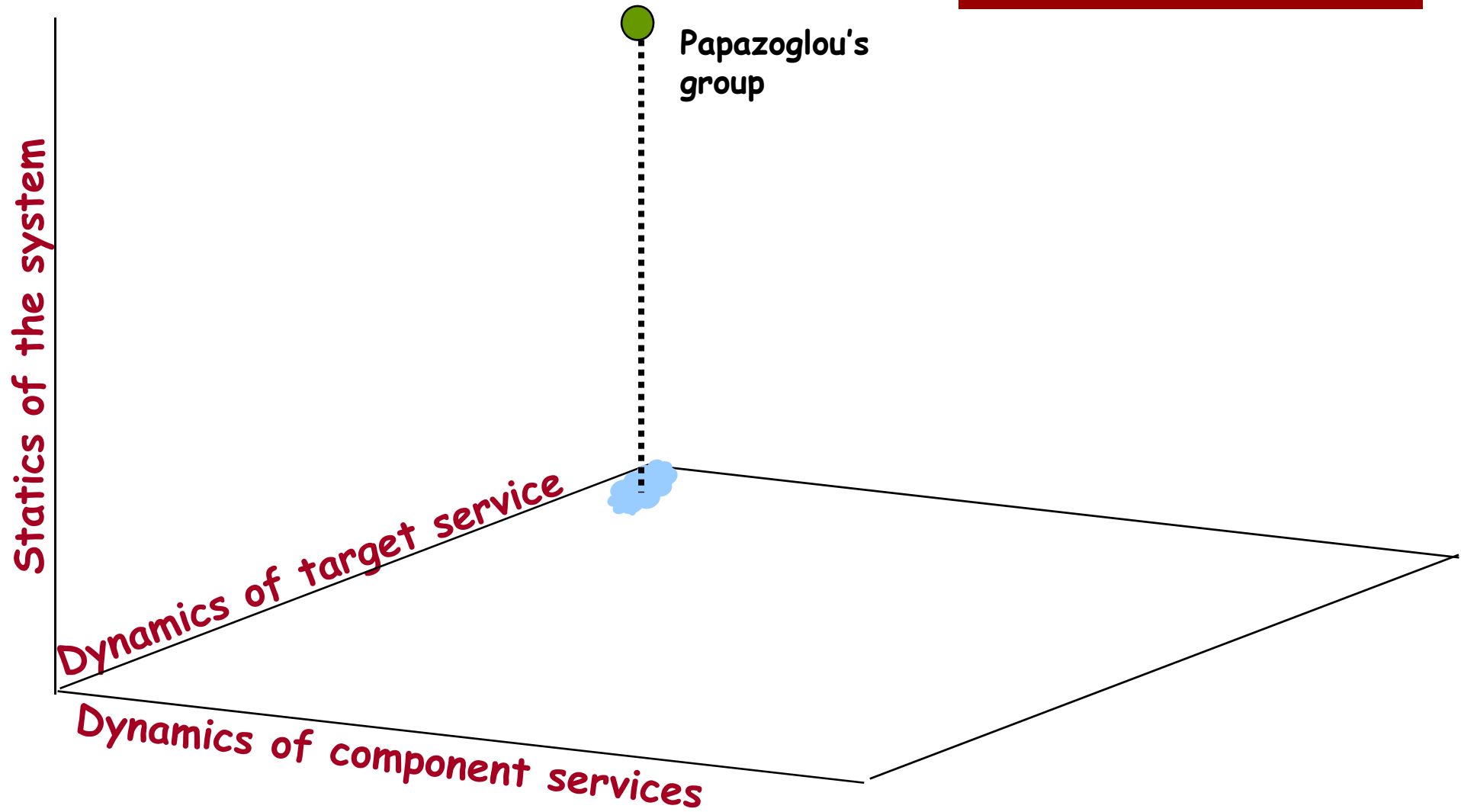
SAPIENZA  
UNIVERSITÀ DI ROMA

- 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: **complex 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

# Papazoglou's group



SAPIENZA  
UNIVERSITÀ DI ROMA



# *Bouguettaya's group*

*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*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

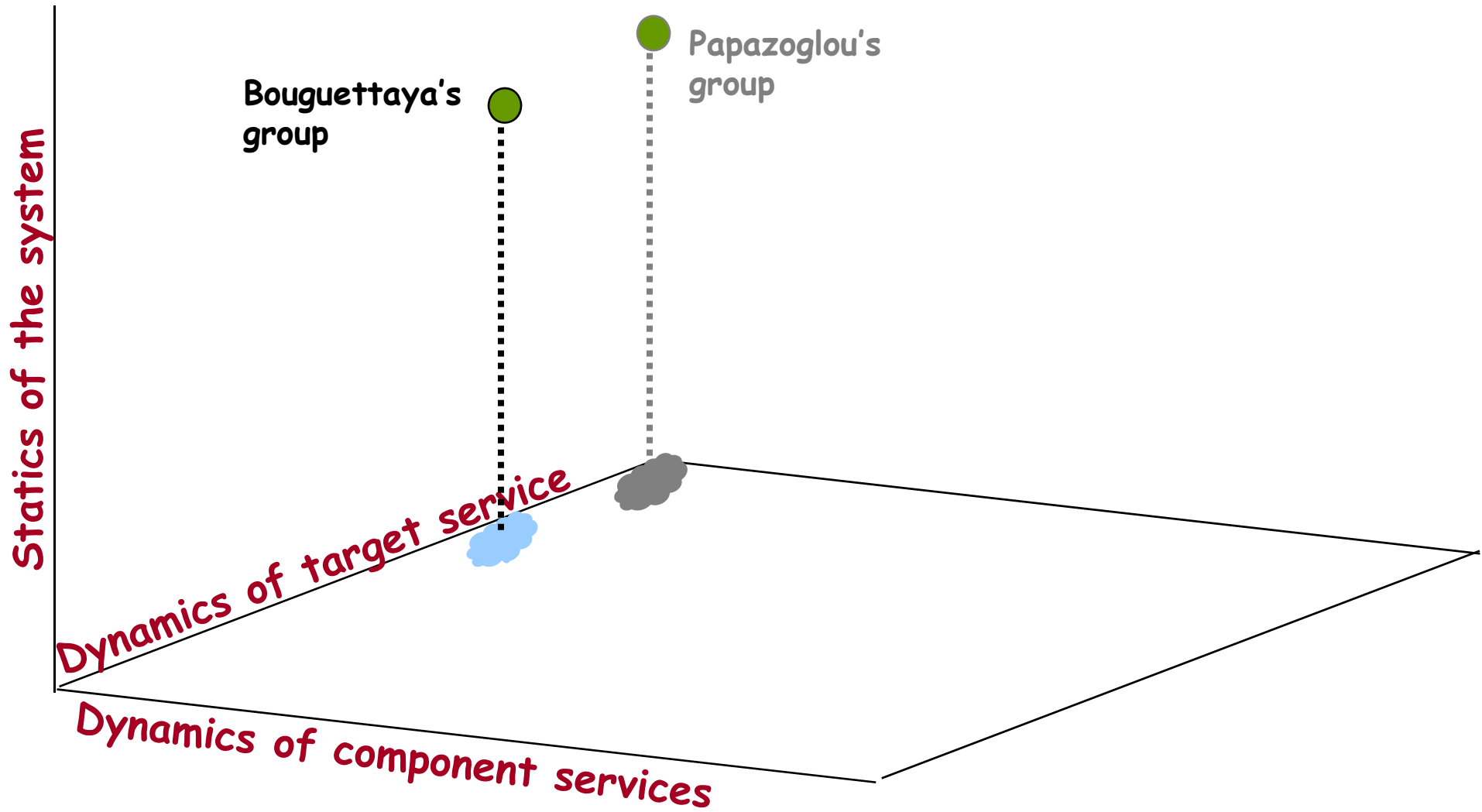
- available services: atomic actions
  - semantically described in terms of their I/O interfaces and non-functional properties such as their purpose, their category and their quality
  - Available services stored into an ontology on the basis of their non-functional properties

# Bouguettaya's group



- 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

# Bouguettaya's group





# Knoblock's group

*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*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

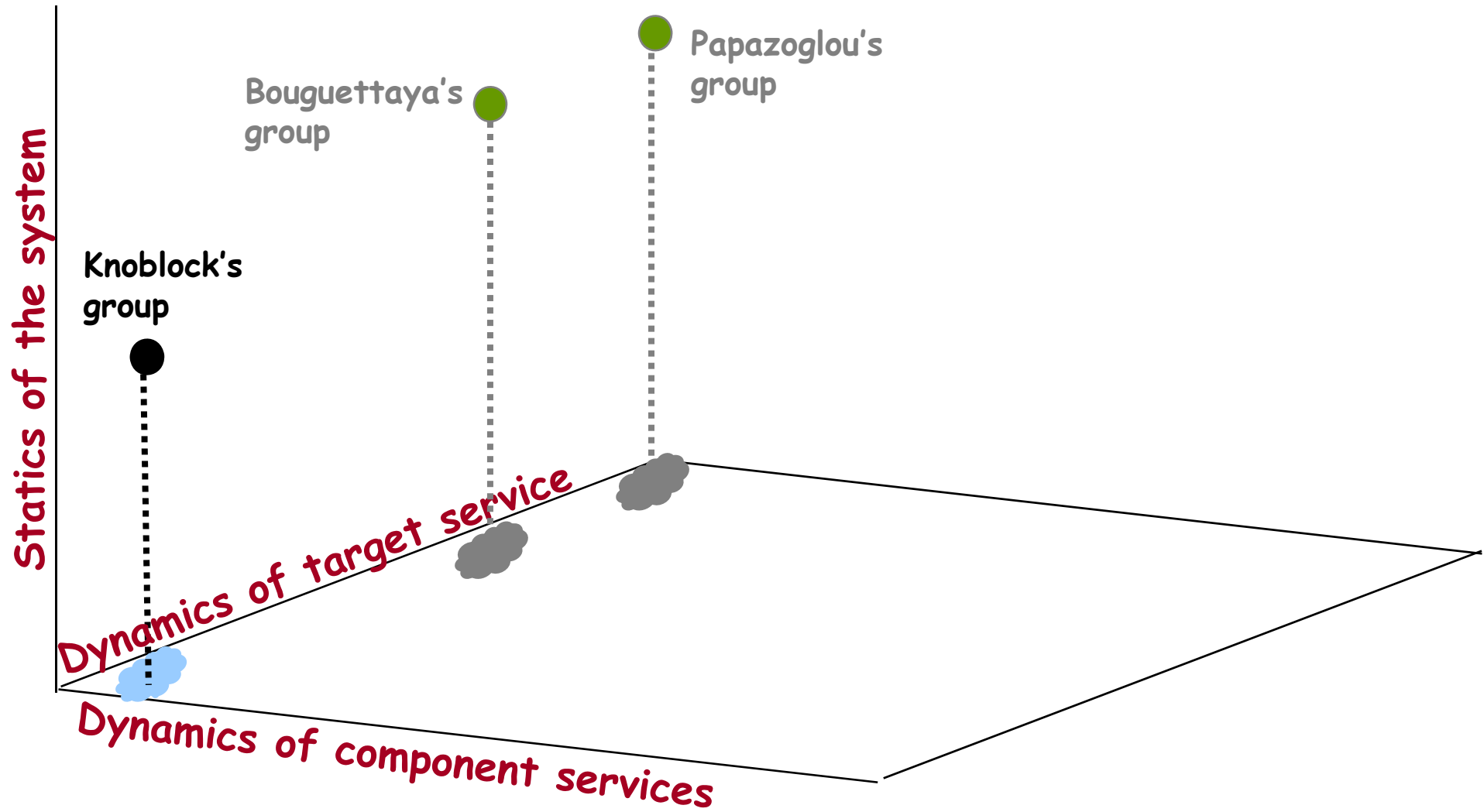
- available service: data query
  - basic idea: informative services as views over data sources
  - each service described in terms of I/O parameters (of course, the latter being provided by the data source), binding patterns and additional constraints on the source
- client request:
  - data query,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

# Knoblock's group



- **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

# Knoblock's group



# 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



## Example (1)

- Component Services
  - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$   
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
  - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$   
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
  - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Ontology:
  - $\text{TravelSettledUp} \equiv \text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
  - $\text{CommutingSettled} \equiv \text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
  - ...
- 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)



- Component Services
  - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$   
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
  - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$   
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
  - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Ontology:
  - $\text{TravelSettledUp} \equiv \text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
  - $\text{CommutingSettled} \equiv \text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
  - ...
- 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)

- Component Services
  - $S_1: \text{True} \rightarrow \{S_1:\text{bookFlight}\} \text{FlightBooked} \wedge \text{MayBookLimo}$   
 $\text{MayBookLimo} \rightarrow \{S_1:\text{bookLimo}\} \text{LimoBooked}$
  - $S_2: \text{True} \rightarrow \{S_2:\text{bookHotel}\} \text{HotelBooked}$   
 $\text{HotelBooked} \rightarrow \{S_2:\text{bookShuttle}\} \text{ShuttleBooked}$
  - $S_3: \text{True} \rightarrow \{S_3:\text{bookEvent}\} \text{EventBooked}$
- Ontology:
  - $\text{TravelSettledUp} \equiv \text{FlightBooked} \wedge \text{HotelBooked} \wedge \text{EventBooked}$
  - $\text{CommutingSettled} \equiv \text{ShuttleBooked} \vee \text{LimoBooked} \vee \text{TaxiAvailabilityChecked}$
  - ...
- 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)



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



## Another Example (2)



### Client Service Request:

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

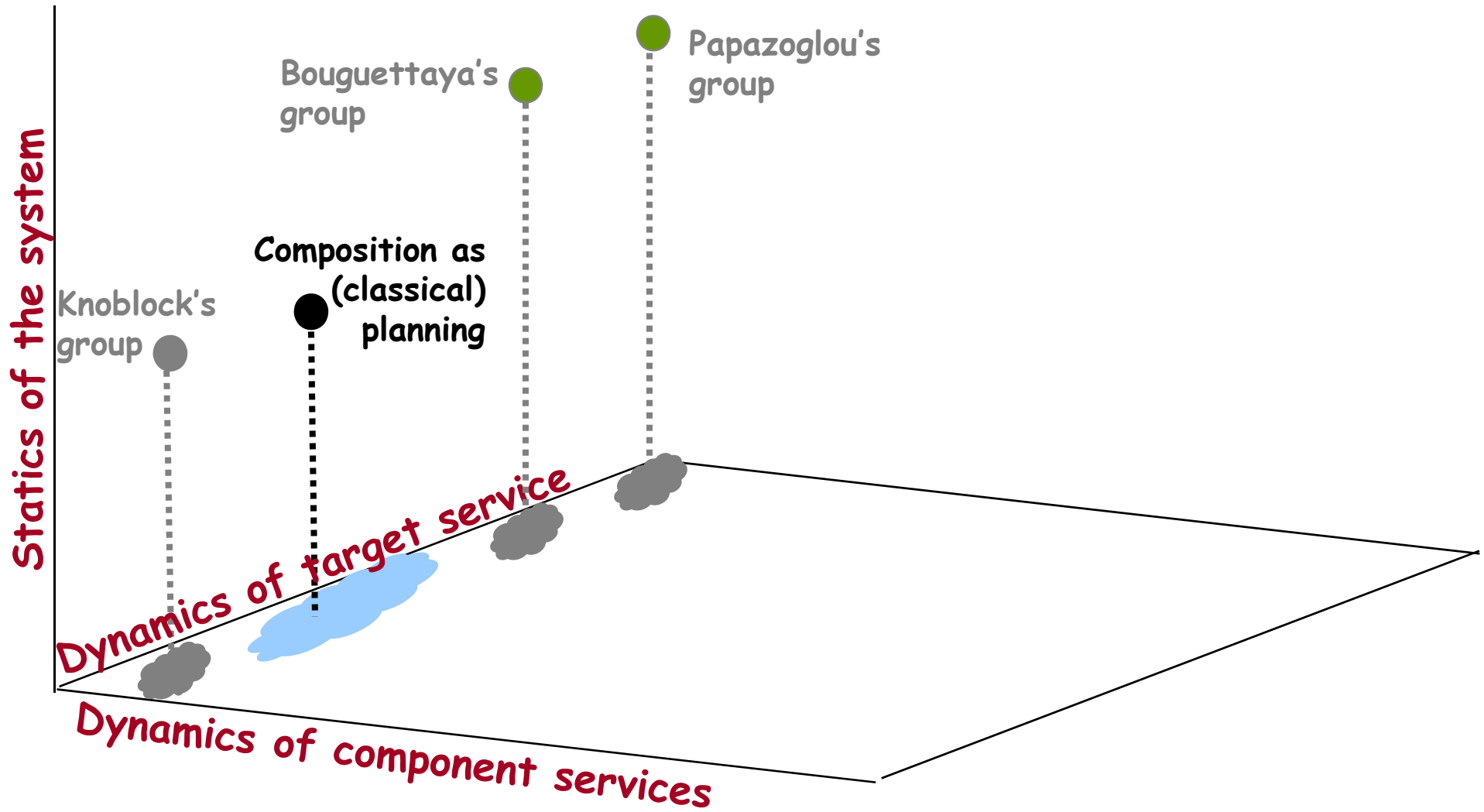
### *What about Registered?*

*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
```

# Composition as Planning



# *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

# References on Planning



SAPIENZA  
UNIVERSITÀ DI ROMA

- **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 et al 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

# Traverso's group



SAPIENZA  
UNIVERSITÀ DI ROMA

---

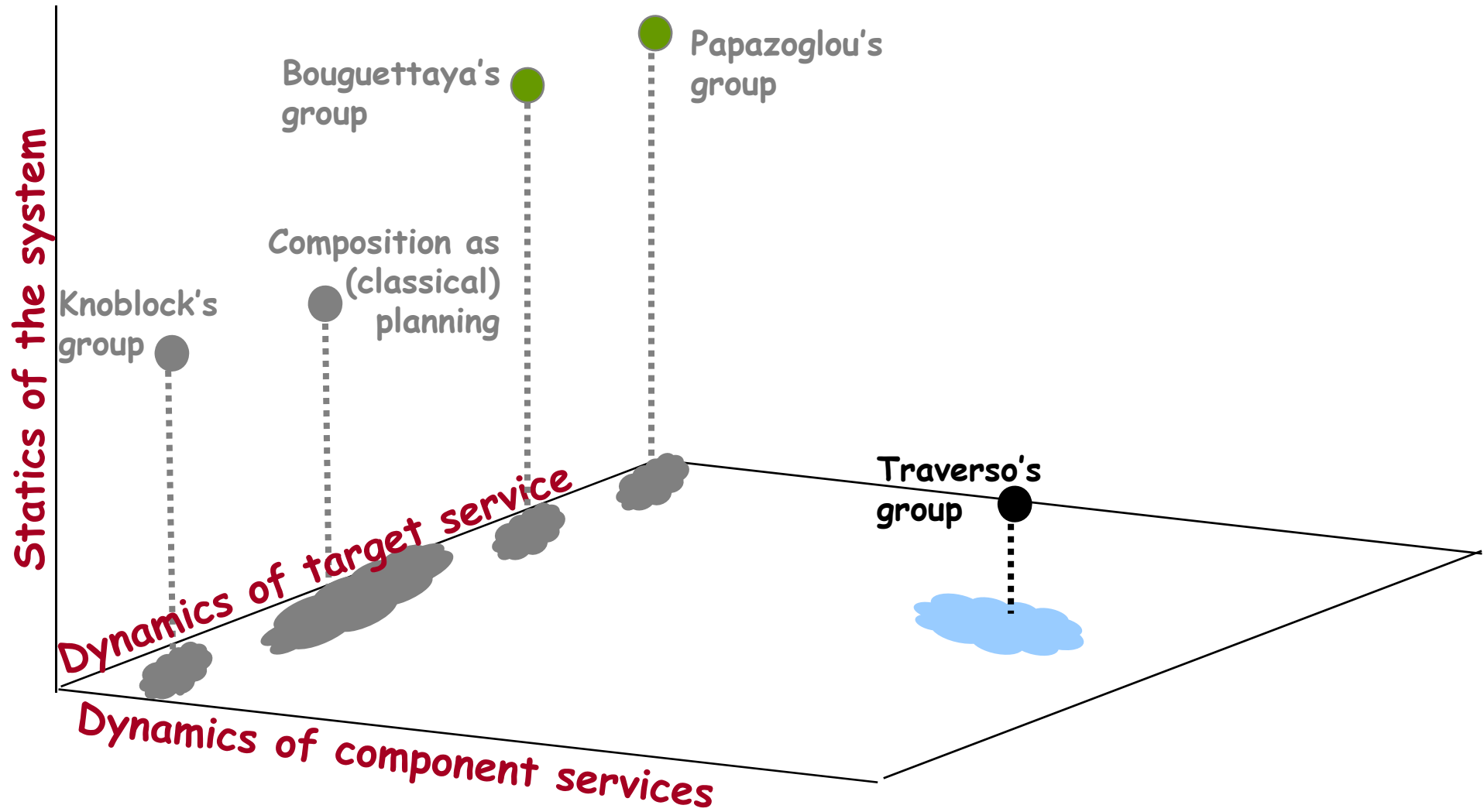
- available services:
  - non-deterministic transition systems characterized by a set of initial states and by a transition relation that defines how the execution of each action leads from one state to a set of states
  - among such services, one represents the client
- client request (called global goal):
  - it specifies a main execution to follow, plus some side paths that are typically used to resolve exceptional circumstances e.g., **Do  $\Phi$  else Try  $\Psi$**

# Traverso's group



- **service composition problem:** (extended) planning problem
  - Input: (i) a set of services, including the one representing the client (behavior), and (ii) the global goal,
  - Output: a plan that specifies how to coordinate the execution of various services in order to realize the global goal.
- **NOTE:**
  - the composition is not tailored towards satisfying completely the client requested behavior, but concerns with the global behavior of the system in which some client desired executions may happen not to be fulfilled

# Traverso's group



# References on Traverso's group



SAPIENZA  
UNIVERSITÀ DI ROMA

---

## 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 AIMSA04] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso: Planning and Monitoring Web Service Composition. AIMSA 2004: 106-115



# *McIlraith's group*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

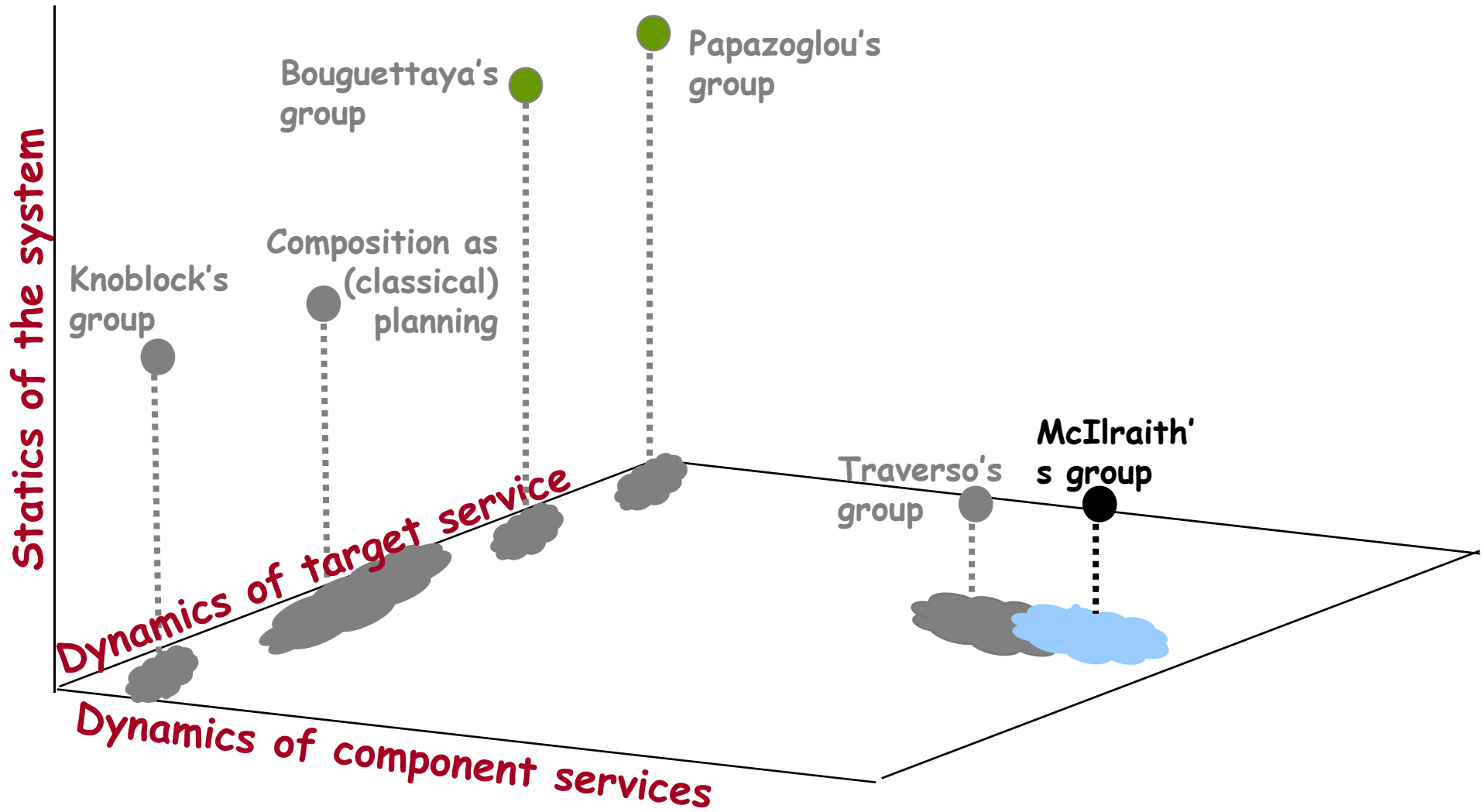
- both available and composite service:  
behavioral description seen as procedures  
invokable by clients
  - Golog procedure, atomically executed, i.e., seen by its client as an atomic Situation Calculus action, presenting an I/O interface
  - each service stored in an OWL-S ontology

# McIlraith's group



- client request:
  - skeleton of a Golog procedure expressing also client 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 client preferences and constraints into account
- NOTE: the client has not influence over the control flow of the composite service

# McIlraith's group



# References on McIlraith's group



SAPIENZA  
UNIVERSITÀ DI ROMA

## 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
- [Burststein etal ISWC02] M. H. Burststein, 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's group



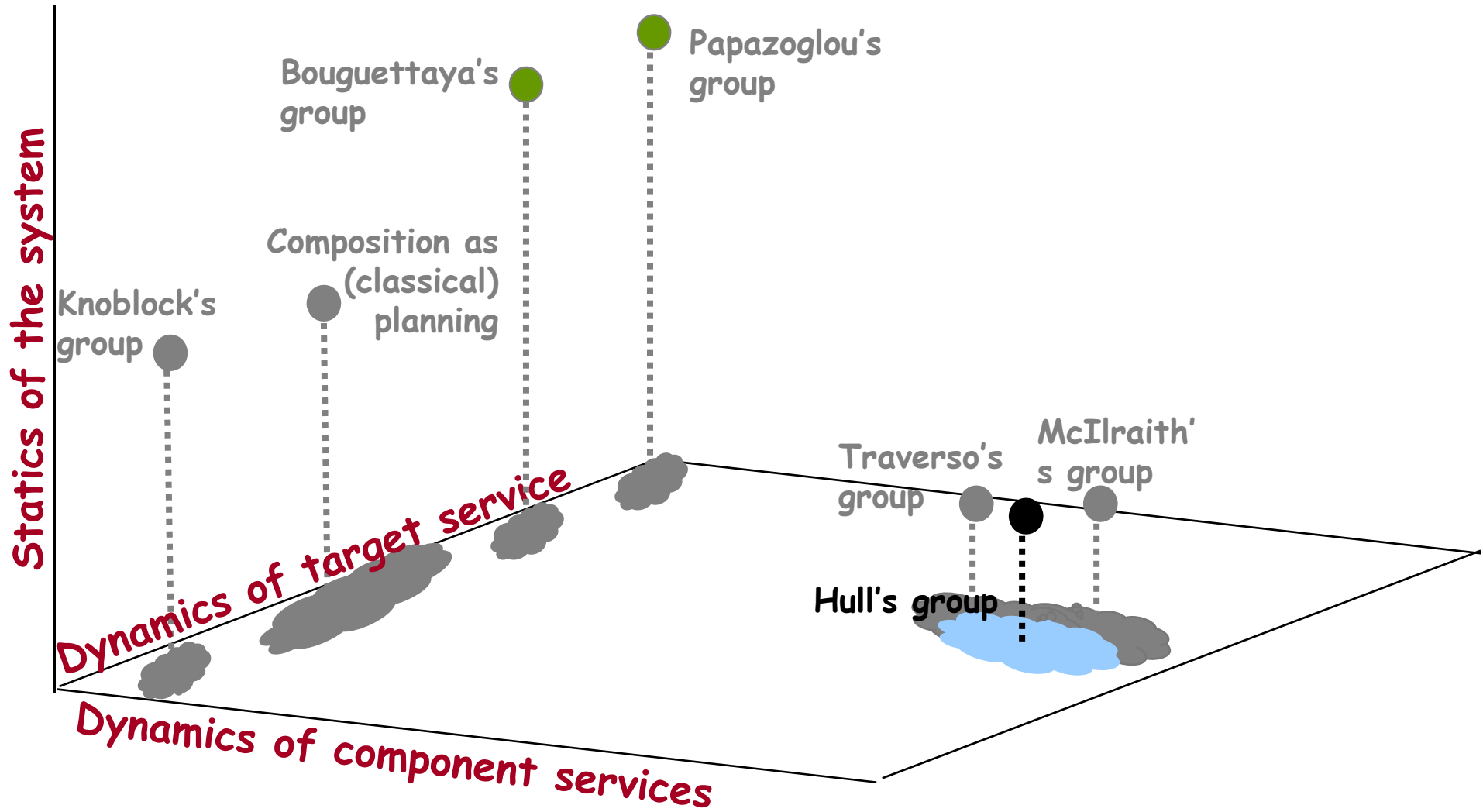
- both available and composite service (peer):  
behavioral description
  - Mealy machine, that exchanges messages with other peers according to a predefined communication topology (channels among peers)
  - peers equipped with (bounded) queue to store messages received but not yet processed
  - Conversation: 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's group



- **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

# Hull's group



# *References on Hull's group*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- [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



# *The Roman group*



- 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

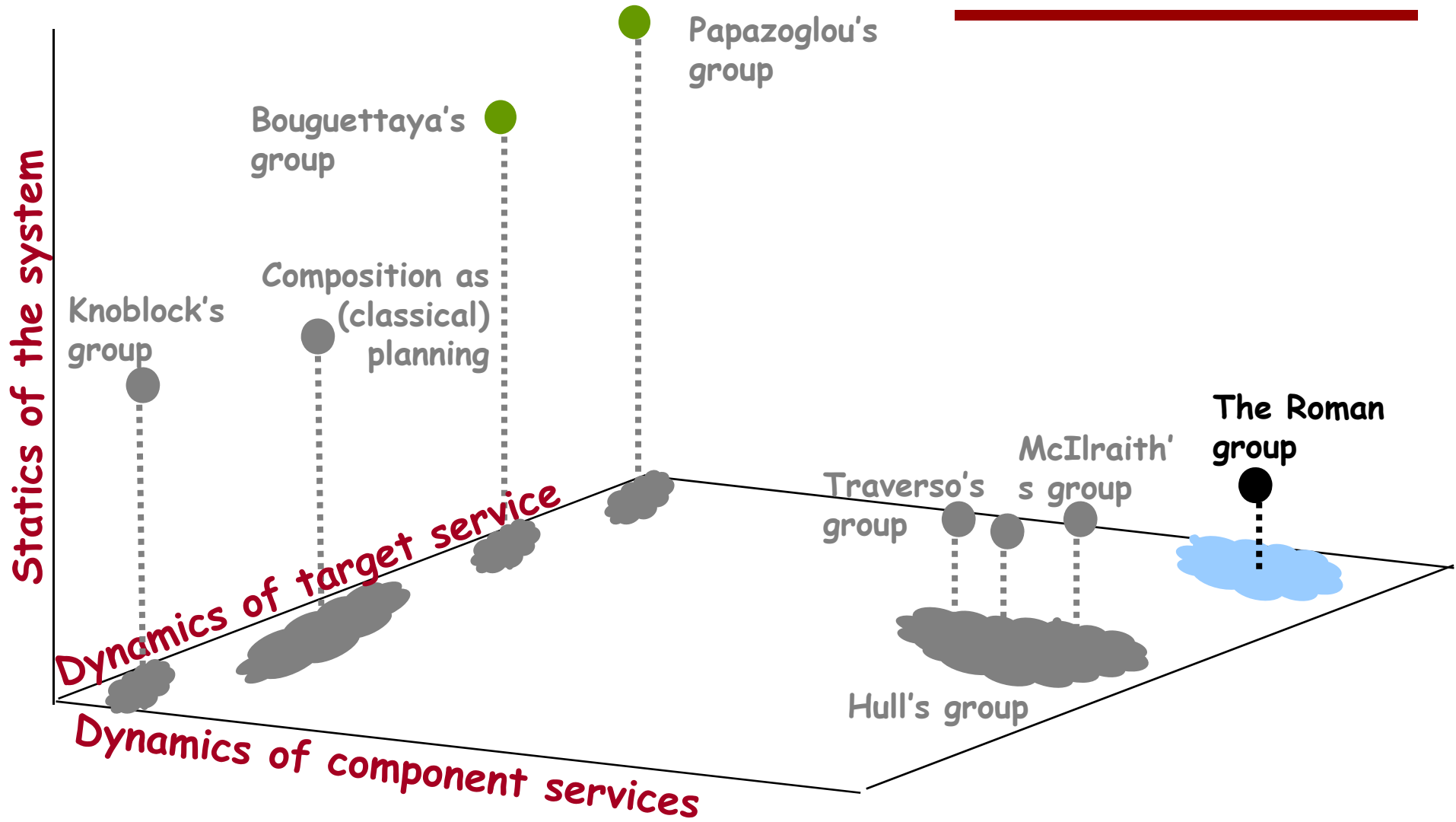


- client request (target service):

---

  - 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

# The Roman group



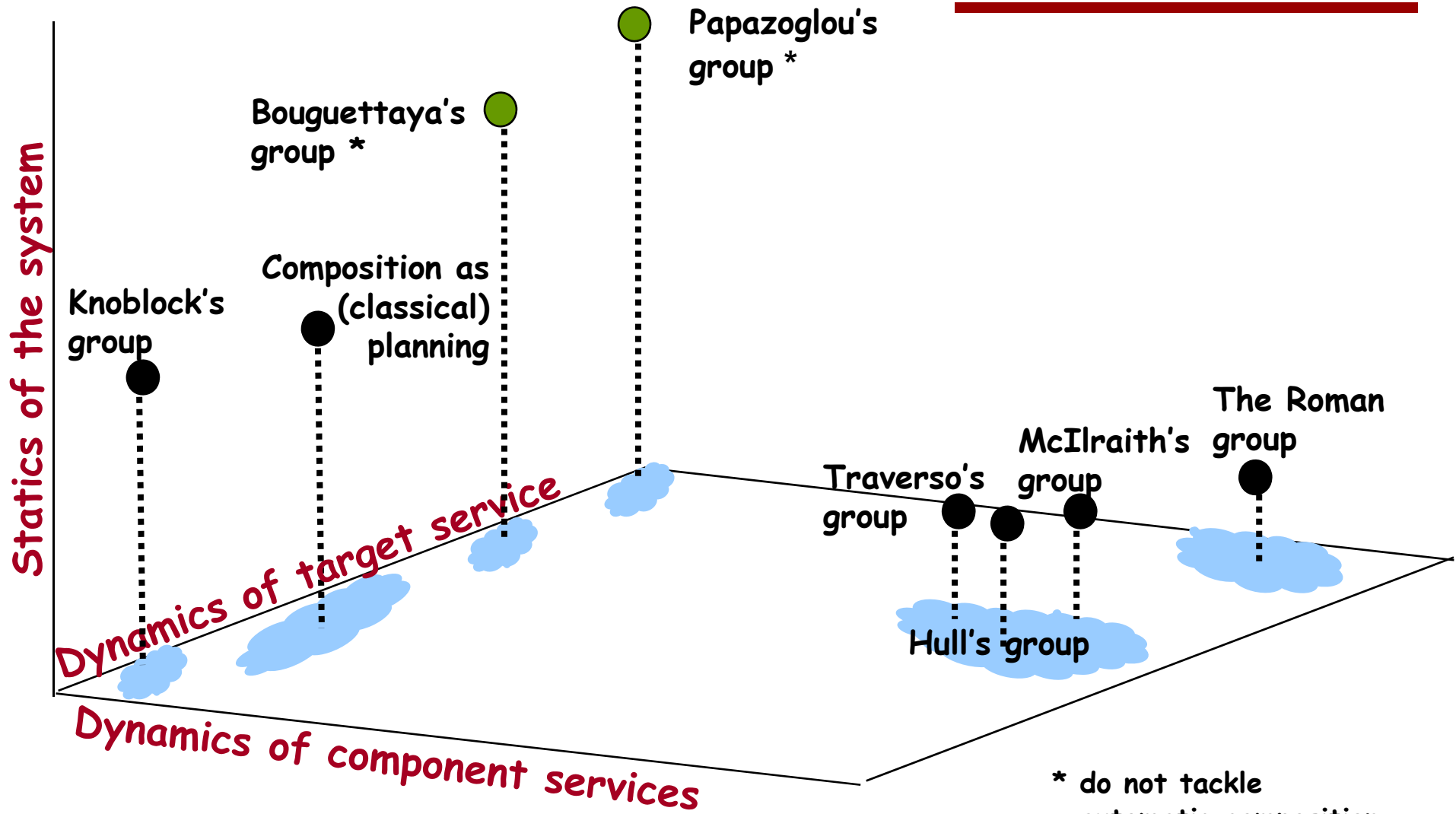
# References on the Roman group



SAPIENZA  
UNIVERSITÀ DI ROMA

- [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
- [Berardi Ph.D] 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.
- [IJCIS 2004] D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella, D. Calvanese: Automatic Service Composition based on Behavioral Description. To appear in IJCIS 2005
- [Gerede et al ICSOC04] C. E. Gerede, R. Hull, O. H. Ibarra, J. Su: Automated Composition of E-services: Lookaheads. ICSOC 2004

# The whole picture



\* do not tackle automatic composition

## *Other Relevant Works*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- Approaches proposing interesting conceptual models for services, not targeted towards composition:
  - Vianu 's group
  - Benatallah & Casati's group

# Vianu's group

*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*



SAPIENZA  
UNIVERSITÀ DI ROMA

- 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 & Casati's

## group

*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*



SAPIENZA  
UNIVERSITÀ DI ROMA

- 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



# *(Only) 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 etal 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 etal VLDB-TESS02]:*
  - 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



SAPIENZA  
UNIVERSITÀ DI ROMA

---

*Automatic Composition:  
A Basic Research Perspective*

# *Basic Research*



- Envision of a sort of  
    "Service Semantic Integration System"
- Semantic integration via **composition synthesis**
- Several directions (as we have seen):
  - **Information Oriented Services**
  - **Services as Atomic Actions**
  - **Services as Processes**

# Semantic Service Integration



Client

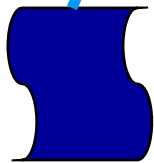
Service request



Mapping1

Mapping2

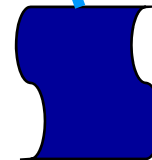
MappingN



Service1



Service2



ServiceN

## Community Basics

**Client** makes a **service request** in term of the **community ontology**

**Available services** express their behavior in terms of the **community ontology**

The **community** realizes the **client service request** making use of the **available services**



# *Service Integration Systems*

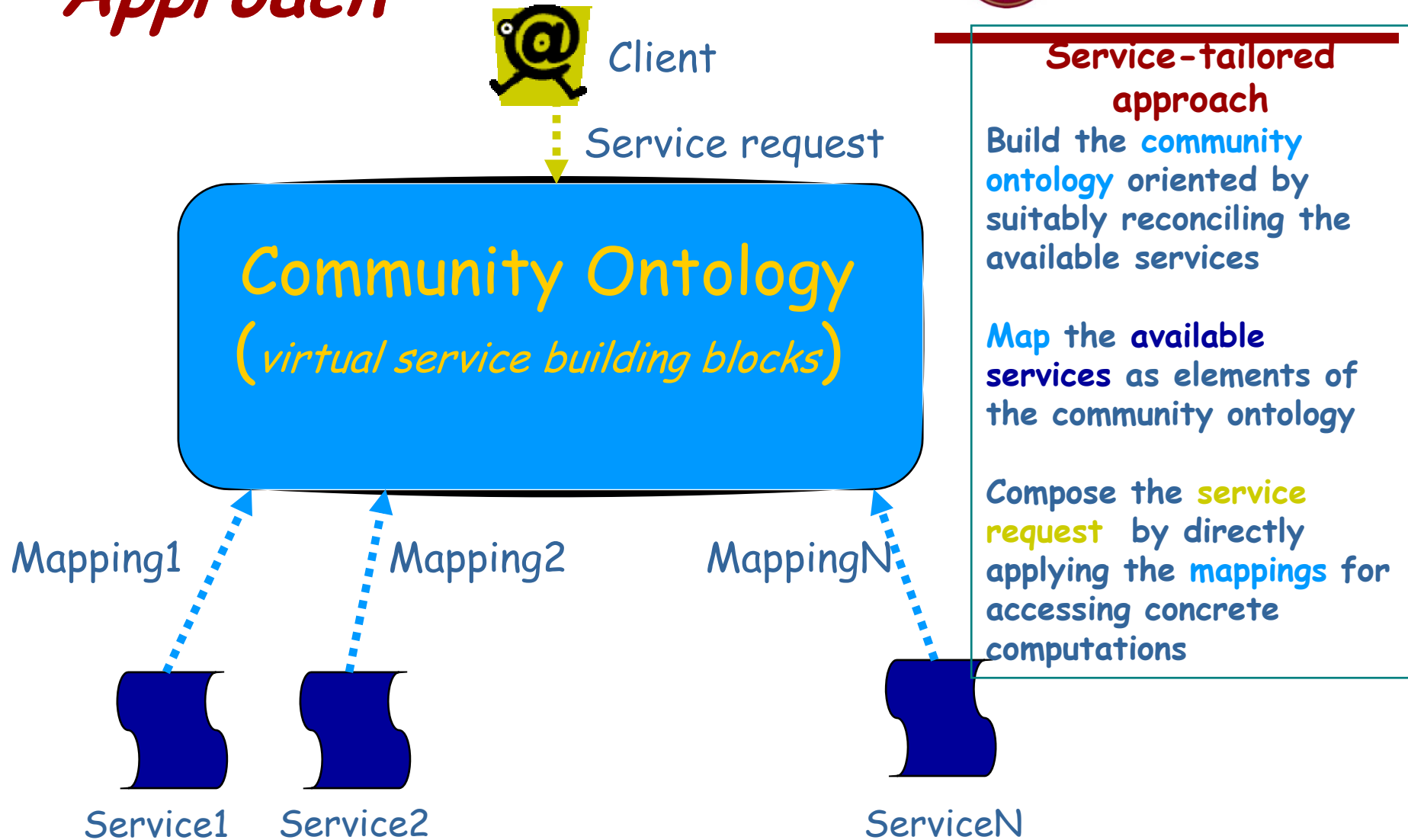


SAPIENZA  
UNIVERSITÀ DI ROMA

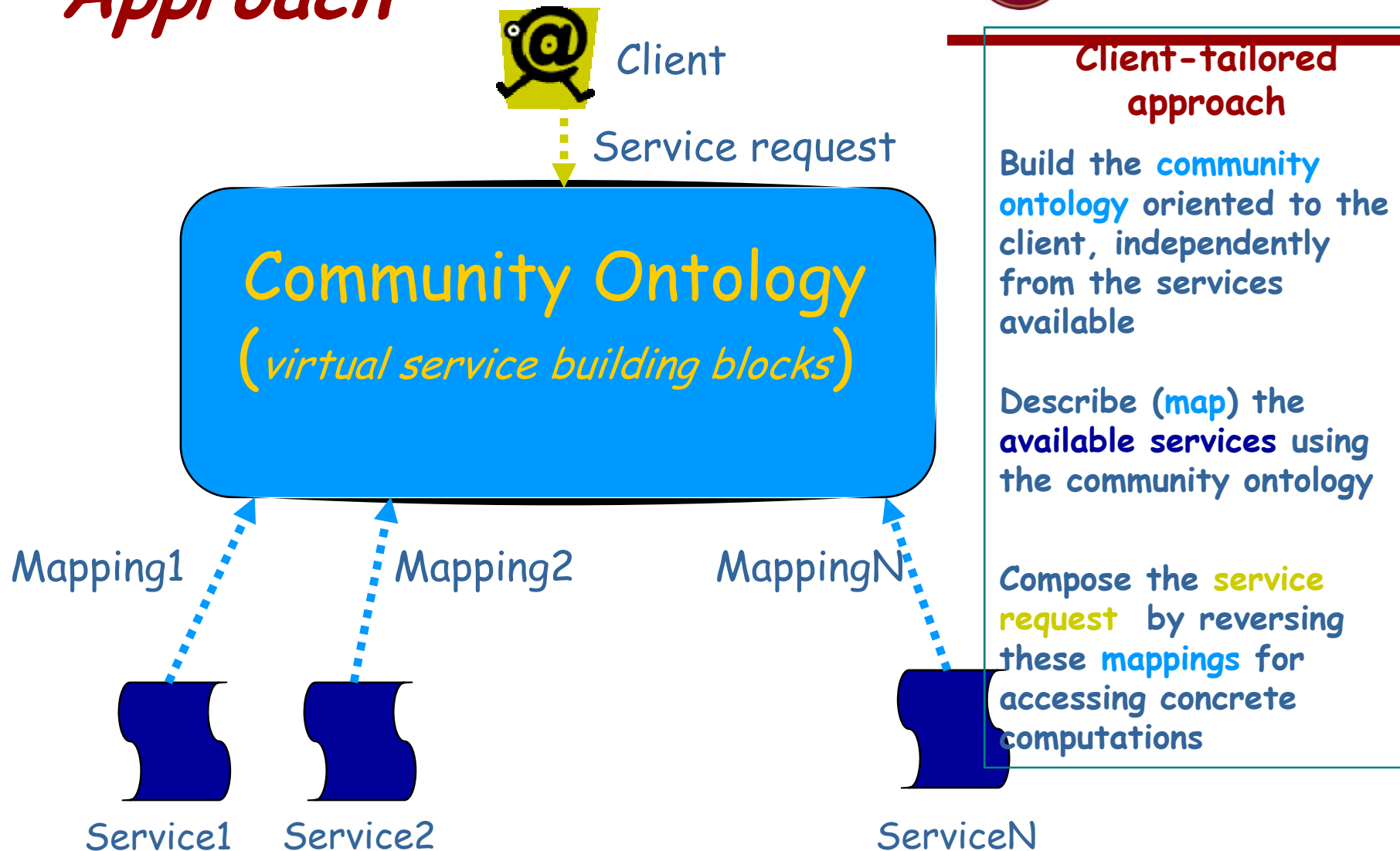
---

- In building such system we can take two general approach:
  - Service-tailored
  - Client-tailored

# Service-Tailored Approach



# Client-Tailored Approach

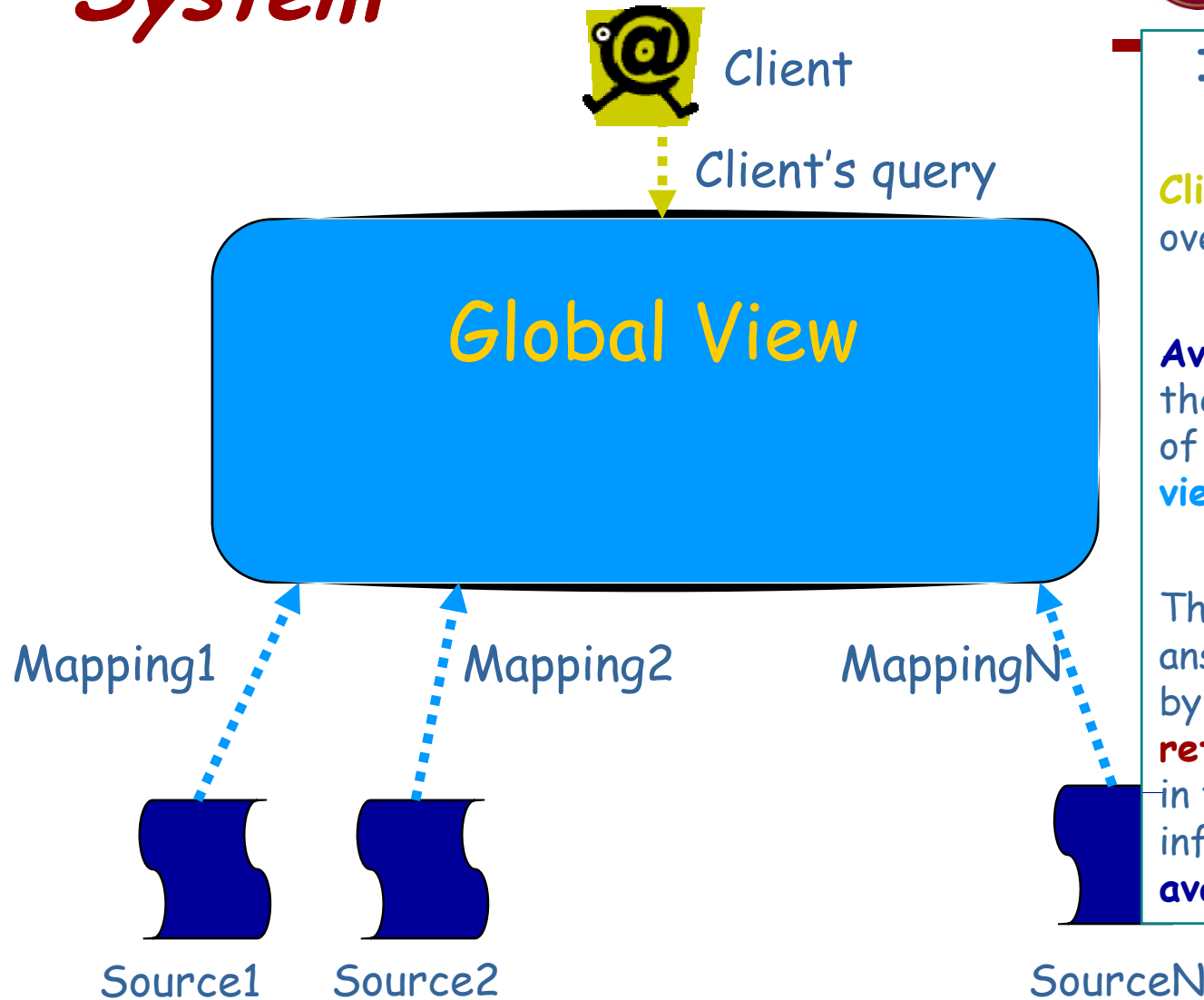


# *Data Integration*



- The Service-tailored vs Client-tailored distinction mimics the **GAV** (Global As View) vs **LAV** (Local As View) approach in data integration ...

# Data Integration System



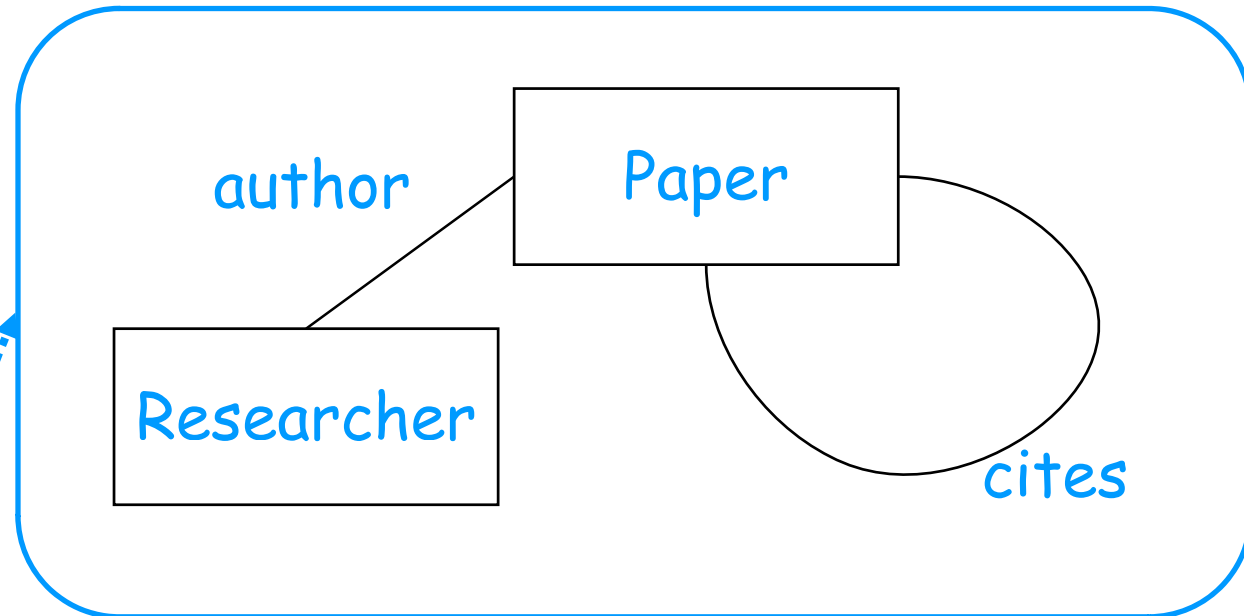
## Integration System Basics

**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 **reformulating/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 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

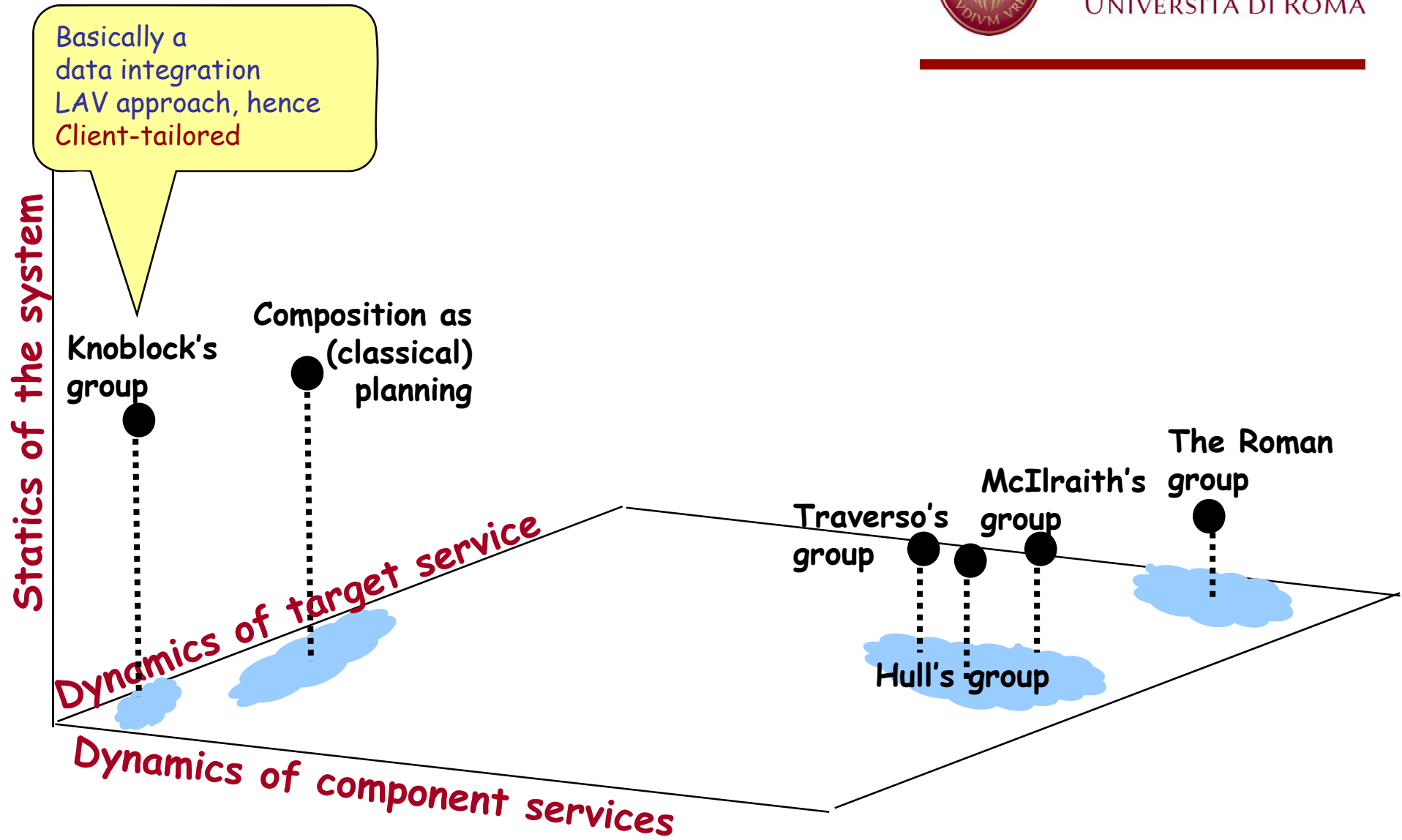


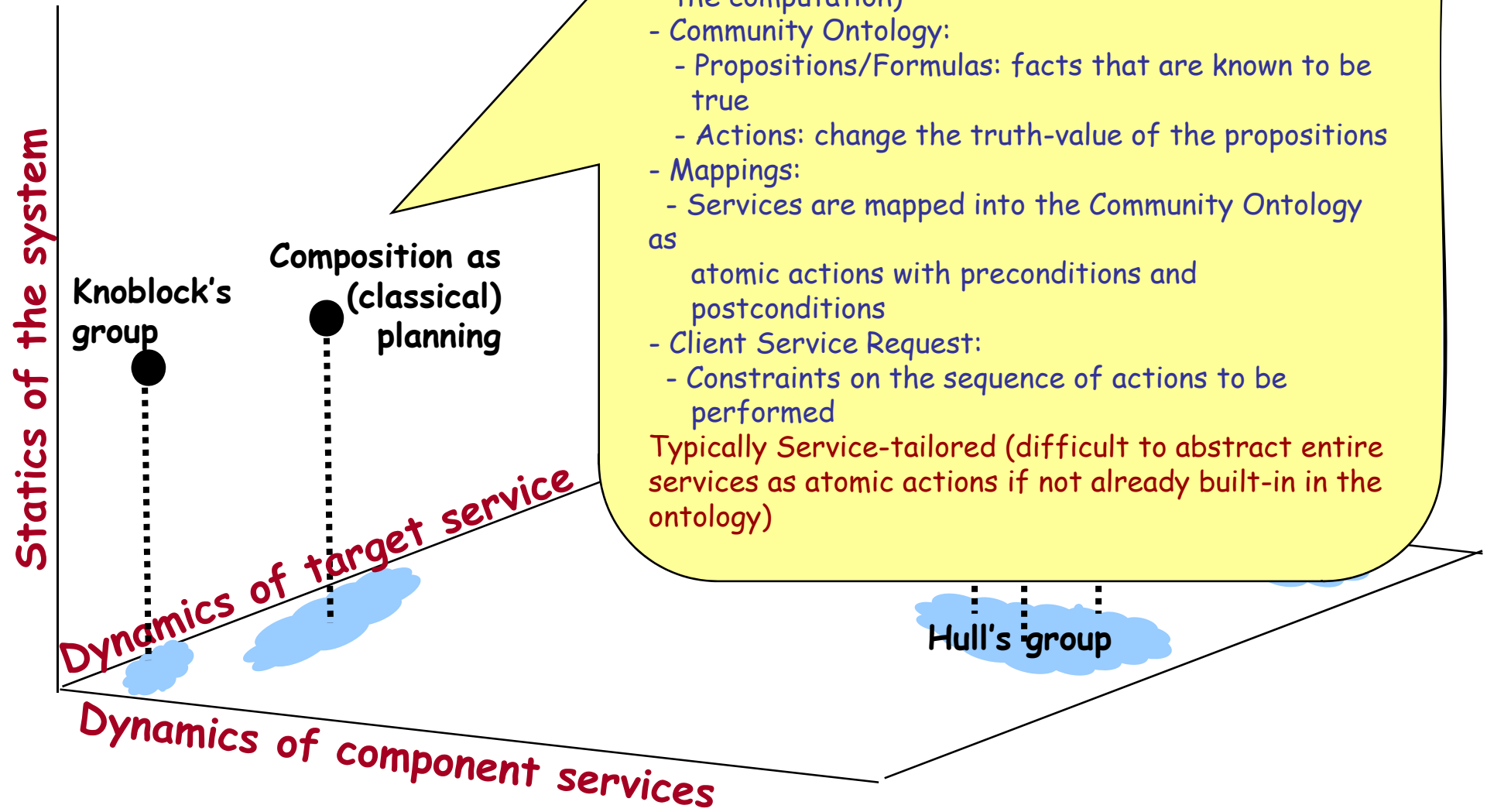
- LAV:
  - Recent research on data integration favors the LAV approach over the GAV approach
  - **Better support of dynamic changes in the system:** sources (services) can be added and deleted without restructuring the global view (community ontology), and hence without impacting the clients
  - **Query 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)

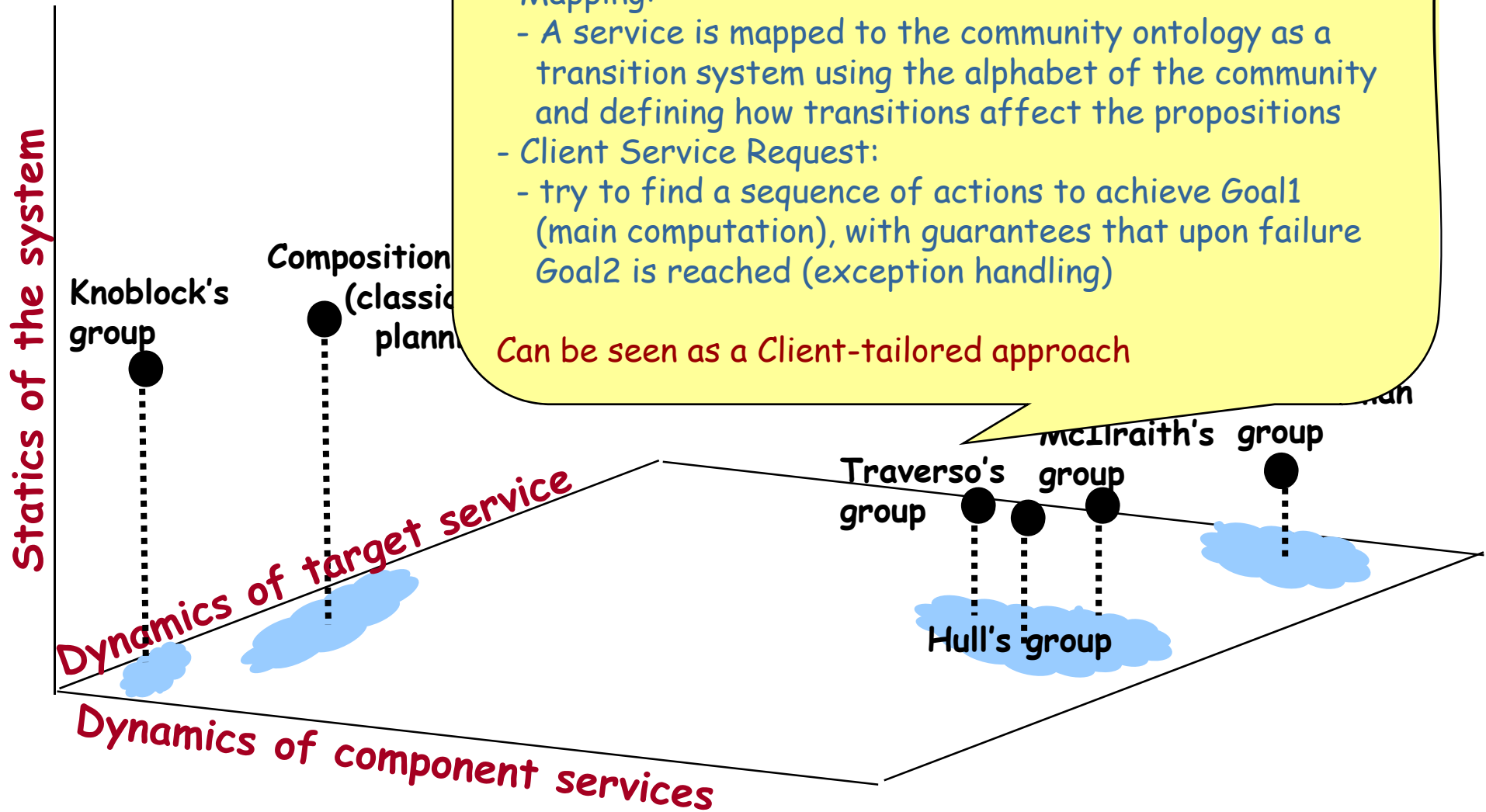




- Services seen as atomic - only I/O behavior modeled (no entry points other than the start and the end of the computation)
  - Community Ontology:
    - Propositions/Formulas: facts that are known to be true
    - Actions: change the truth-value of the propositions
  - Mappings:
    - Services are mapped into the Community Ontology as atomic actions with preconditions and postconditions
  - Client Service Request:
    - Constraints on the sequence of actions to be performed
- Typically Service-tailored (difficult to abstract entire services as atomic actions if not already built-in in the ontology)

- Services seen as (finite) transition systems
- Common ontology:
  - atomic actions and propositions, as in Planning
- Mapping:
  - A 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 Goal1 (main computation), with guarantees that upon failure Goal2 is reached (exception handling)

Can be seen as a Client-tailored approach



- Services seen as (possibly infinite) transition systems
- Common ontology is a 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 undertandment that it specify acceptable sequenced of actions for the client (as in planning) and not a transition system that the client want to realize (see later)

Essentially a Service-tailored approach

Statics of the system

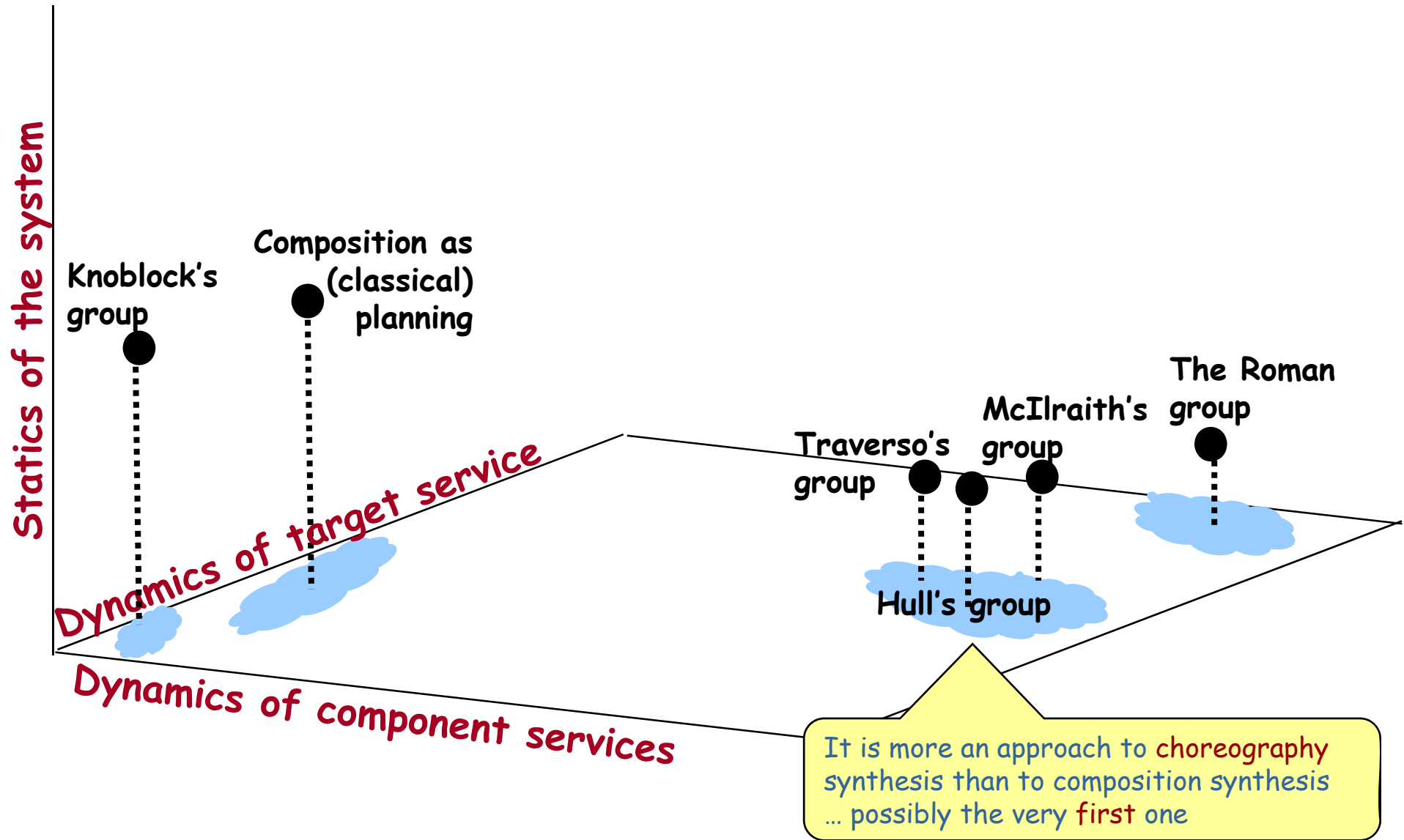
Dynamics of target service

Dynamics of component services

Knoblock's group

Traverso's group    McIlraith's group    The Roman group

Hull's group

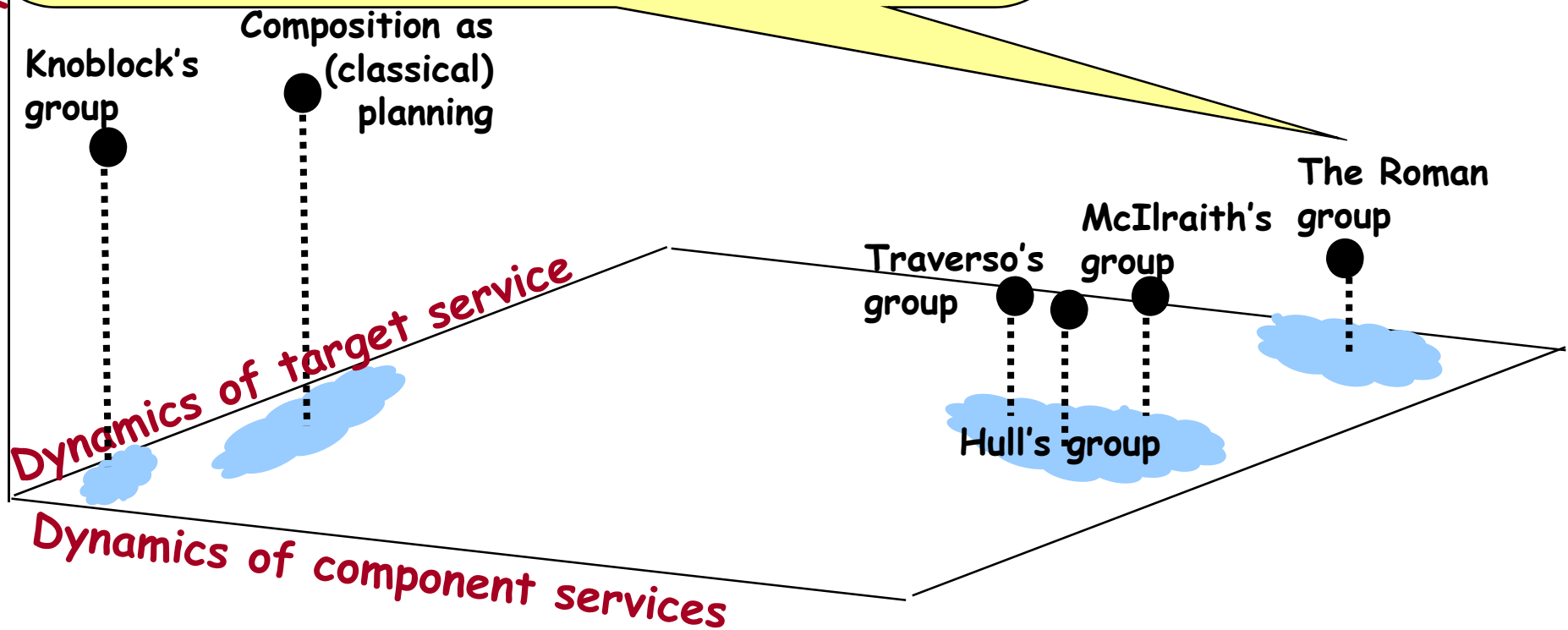




- Services seen as (finite) transition systems
- Common ontology: Alphabet of atomic actions
- Mapping:
  - A service is mapped to the community ontology as a transition system using the alphabet of the community
- Client Service Request:
  - Desired service behavior, i.e., a (finite) TS using the common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

Is a Client-tailored approach

Statics of the system





# 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



## *Lecture 5*

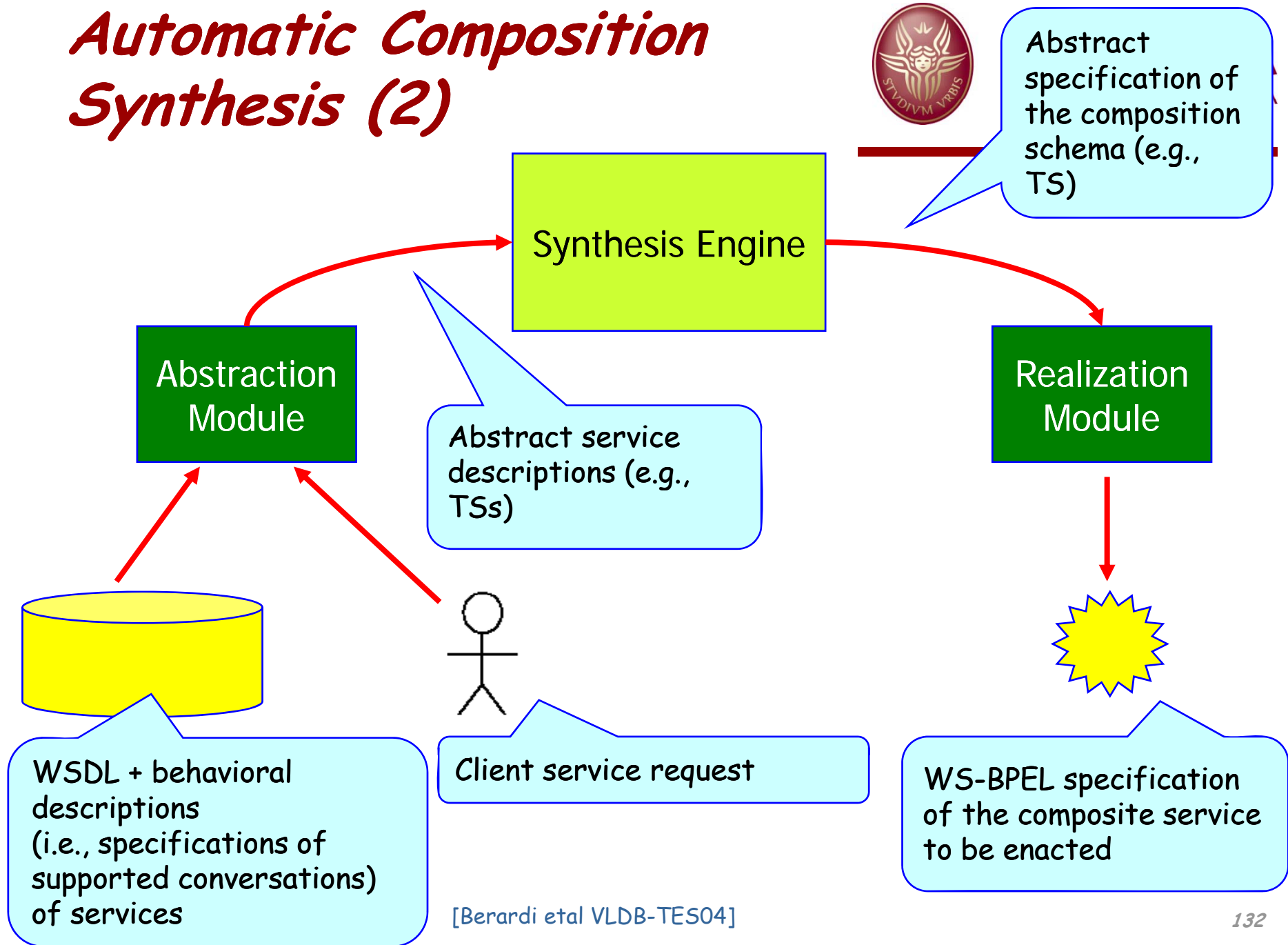
1. Technical Details on WSCE
2. Security
3. Composition in Distributed Mobile Scenarios

# *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)



# Representing Service Behaviors in XML



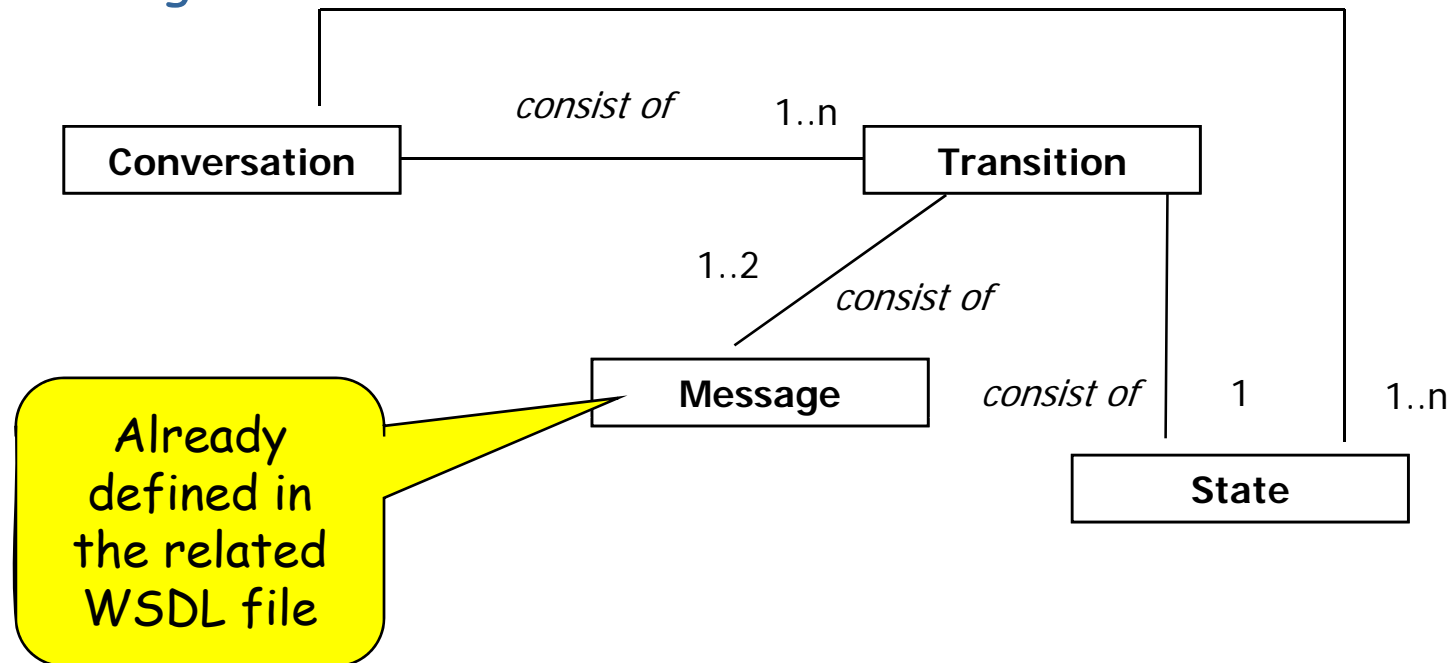
- Different approaches for representing TSs
  - Web Service Transition Language (WSTL)
    - Accademic proposal
  - Web Service Choreography Description Language (WS-CDL)
    - Standard
    - Not really designed for this
  - Web Service Business Process Execution Language (WS-BPEL) **abstract**
  - OWL-S
    - see, e.g., [Pistore&Traverso ISWC04]
  - WSMO
  - ... ..

# Web Service Transition Language (WSTL)

[Berardi et al. @ Transactions of the SDPS: Journal of Integrated Design and Process Science 8 (2004), no. 2]



- WSTL is a XML-based description language able to represent the observable (i.e., from the point of view of the service users) behavior of service
  - describe the correct sequence of the exchanged messages



# *An Example*



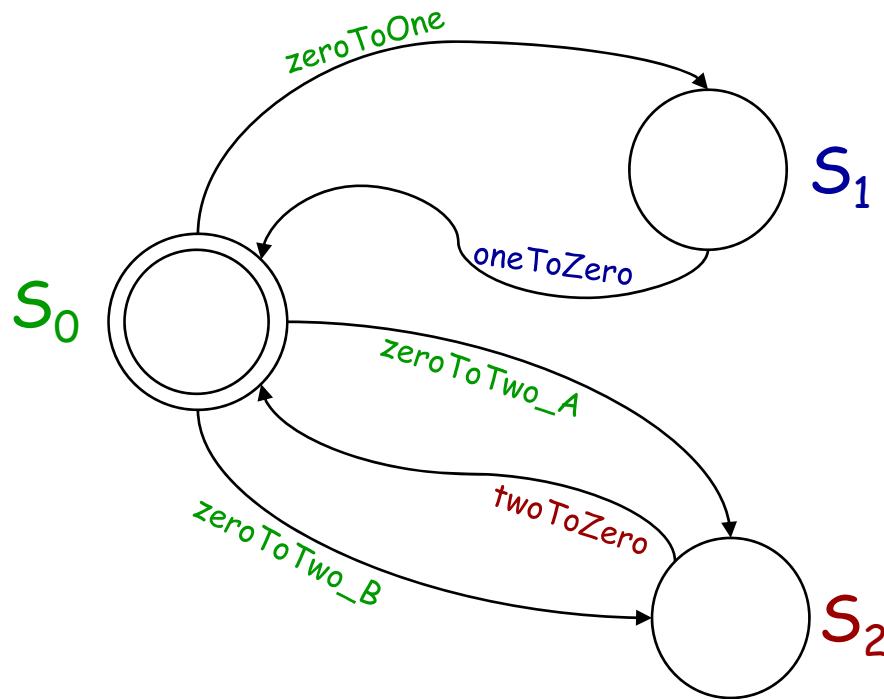
```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation ... ..
  <Transition source="start" target="1">
    <InputMessage>SearchByTitleRequest</InputMessage>
    <OutputMessage>SearchByTitleResponse</OutputMessage>
  </Transition>
  <Transition source="start" target="1">
    <InputMessage>SearchByAuthorRequest</InputMessage>
    <OutputMessage>SearchByAuthorResponse</OutputMessage>
  </Transition>
  <Transition source="1" target="start">
    <InputMessage>ListenRequest</InputMessage>
    <OutputMessage>ListenResponse</OutputMessage>
  </Transition>
</Conversation>
```

# TS in WS-CDL<sub>(1)</sub>



What we have to represent?

- States : name and typology (*initail, final or transient*)
- Transitions : name of the operation and the states that will be reached



## States :

- $S_0$  = initail/final
- $S_1, S_2$  = transient

## Transitions :

{*zeroToOne, zeroToTwo\_A, zeroToTwo\_B, oneTwoZero, twoToZero*}



# TS in WS-CDL<sub>(2)</sub>



In WS-CDL some useful *elements* can be used for the issue :

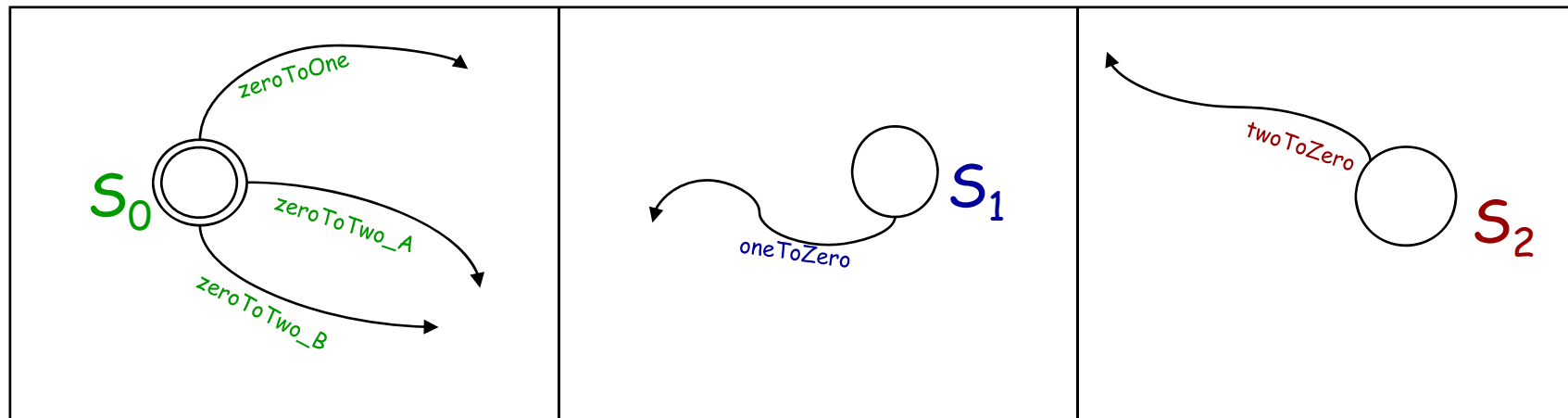
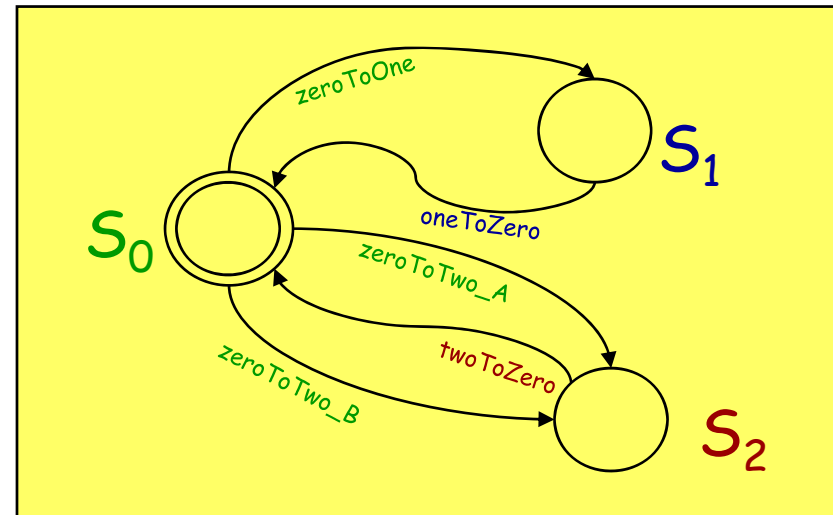
- **<choerography>** represents a set of atomic actions that a web service perfoms inside a choreography. In our representation a state can be mapped in a choreography element declaring in its attributes the name of the state
- **<workunit>** is used in a choreography when a declared event is coming up. In mapping operation, a workunit can be used to declare the goal state reached by the state declared in *<choreography>*
- **<interaction>** is used in WS-CDL to describe the interactions among the Web Services. This element can map all the possible transitions that the state declared in *<choreography>* can perform to reach the goal state
- **<exchange>** is used to declare the message exchanged during the web service invocation
- **<description>** is used to declare the typology of the states : *initial, final, transient*

# TS in WS-CDL<sub>(3)</sub>



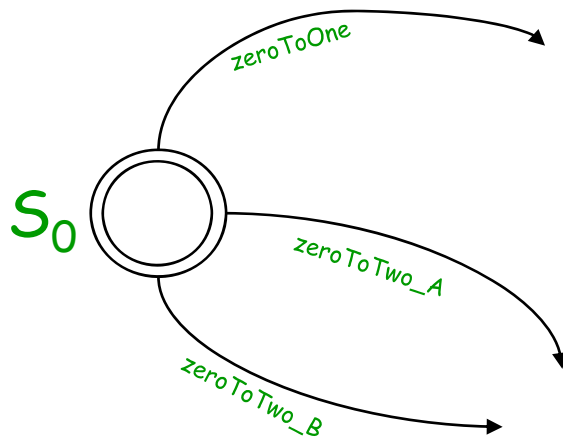
```
For each state  $S_i$  {  
  declare <choreography name =  $S_i$  >  
  declare <description name = "documentation" ..... //typology of state  
  
  For each state reached by  $S_i$  {  
    declare <workunit name =  $S_j$  >  
    declare <description type="documentation" ..... //typology of state  
  
    For each transition from  $S_i$  to  $S_{i+1}$  {  
      declare <interaction name="nameOfAction" operation="operationName">  
      declare <exchange action="request" name="operationNameRequest">  
      declare <exchange action="response" name="operationNameResponse">  
    }  
  }  
}
```

# TS in WS-CDL<sub>(4)</sub>



# TS in WS-CDL<sub>(5)</sub>

... start translating state  $S_0$  ...

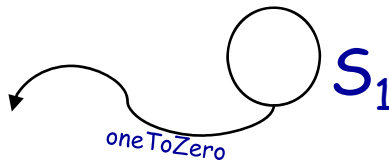


```
<choreography name="S0" root="false">
  <description type="documentation">initial-final</description>
  <workunit name="S1">
    <description type="documentation">transient</description>
    <interaction name="S0_to_S1" operation="zeroToOne">
      <exchange action="request" name="zeroToOneRequest">
        <exchange action="response" name="zeroToOneResponse">
      </interaction>
    </workunit>

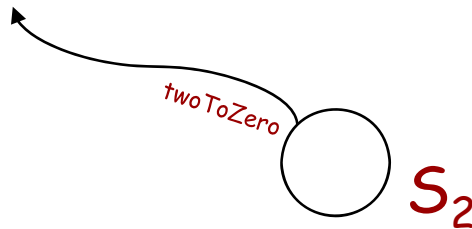
  <workunit name="S2">
    <description type="documentation">transient</description>
    <choice>
      <interaction name="S0_to_S2_A" operation="zeroToTwo_A">
        <exchange action="request" name="zeroToTwo_ARequest">
          <exchange action="response" name="zeroToTwo_AResponse">
        </interaction>
      <interaction name="S0_to_S2_B" operation="zeroToTwo_B">
        <exchange action="request" name="zeroToTwo_BRequest">
          <exchange action="response" name="zeroToTwo_BResponse">
        </interaction>
      </choice>
    </workunit>
  </choreography>
```

# TS in WS-CDL<sub>(6)</sub>

... and the state  $S_1$  and  $S_2$ .



```
<choreography name="S1">
  <description type="documentation">transient</description>
  <workunit name="S0">
    <description type="documentation">initial-final</description>
    <interaction name="S1_to_S0" operation="oneToZero">
      <exchange action="request" name="oneToZeroRequest">
      <exchange action="response" name="oneToZeroResponse">
    </interaction>
  </workunit>
</choreography>
```



```
<choreography name="S2">
  <description type="documentation">transient</description>
  <workunit name="S0">
    <description type="documentation">initial-final</description>
    <interaction name="S2_to_S0" operation="twoToZero">
      <exchange action="request" name="twoToZeroRequest">
      <exchange action="response" name="twoToZeroResponse">
    </interaction>
  </workunit>
</choreography>
```

# TS in WS-CDL<sub>(7)</sub>



## Complete WS-CDL file created by mapping operation<sub>(1)</sub>

```
<?xml version="1.0" encoding="UTF-8"?>
<package
  name="SampleFSM"
  author="DIS Dipartimento di Informatica e Sistemistica"
  version="1.0"
  targetNamespace="uri"
  xmlns="http://www.w3.org/2004/12/ws-chor/cdl" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
<choreography name="S0" root="false">
  <description type="documentation">initial-final</description>
  <workunit name="S1">
    <description type="documentation">transient</description>
    <interaction name="S0_to_S1" operation="zeroToOne">
      <exchange action="request" name="zeroToOneRequest">
      <exchange action="response" name="zeroToOneResponse">
    </interaction>
  </workunit>
  <workunit name="S2">
    <description type="documentation">transient</description>
    <choice>
      <interaction name="S0_to_S2_A" operation="zeroToTwo_A">
        <exchange action="request" name="zeroToTwo_ARequest">
        <exchange action="response" name="zeroToTwo_AResponse">
      </interaction>
      <interaction name="S0_to_S2_B" operation="zeroToTwo_B">
        <exchange action="request" name="zeroToTwo_BRequest">
        <exchange action="response" name="zeroToTwo_BResponse">
      </interaction>
    </choice>
  </workunit>
</choreography>
```

# FSM Mapping in WS-CDL<sub>(8)</sub>



Complete WS-CDL file created by mapping operation<sub>(2)</sub>

```
<choreography name="S1">
  <description type="documentation">initial-final</description>
  <workunit name="S0">
    <description type="documentation">transient</description>
    <interaction name="S1_to_S0" operation="oneToZero">
      <exchange action="request" name="oneToZeroRequest">
        <exchange action="response" name="oneToZeroResponse">
          </interaction>
        </exchange>
      </interaction>
    </workunit>
  </choreography>

<choreography name="S2">
  <description type="documentation">initial-final</description>
  <workunit name="S0">
    <description type="documentation">transient</description>
    <interaction name="S2_to_S0" operation="twoToZero">
      <exchange action="request" name="twoToZeroRequest">
        <exchange action="response" name="twoToZeroResponse">
          </interaction>
        </exchange>
      </interaction>
    </workunit>
  </choreography>
</package>
```

# TS in WSMO

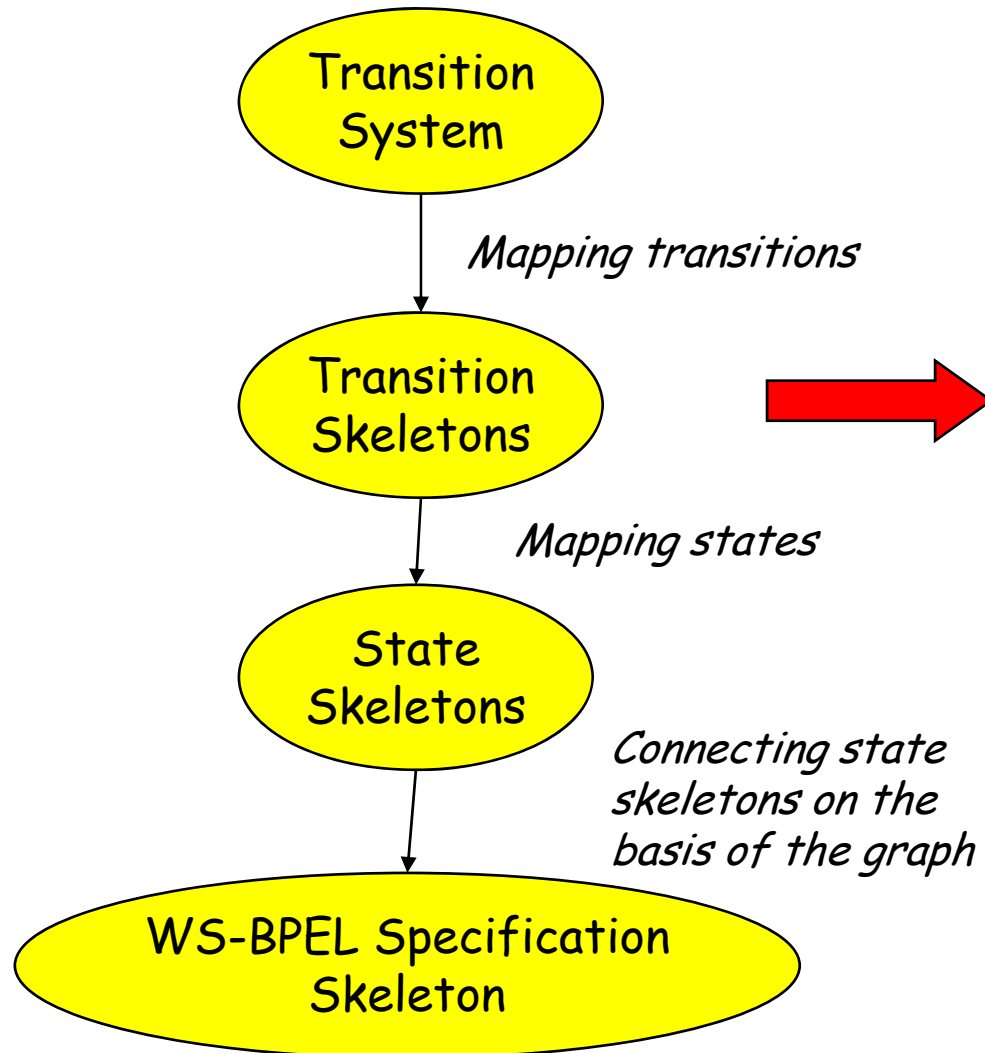


- Represented using a “reduced” **Abstract State Machine** consisting of a sequence of **if-then** rules without **forall** and **choose** rules (that conversely deal with data)
- Transition Rule for **SearchByTitle**

```
if(?SearchByTitleRequest[
  ... ..
  ]memberOf uor#SearchByTitleRequest)and
  (exists ... ..)
  // this condition used for coding states of the TS
  )then add(_# memberOf SearchByTitleResponse)
endif
```



# 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 all the nodes of the graph, starting from the initial state and applying the previous rules.

N.B.: (1) and (2) works for in-out interactions (the ones 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.

# 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:

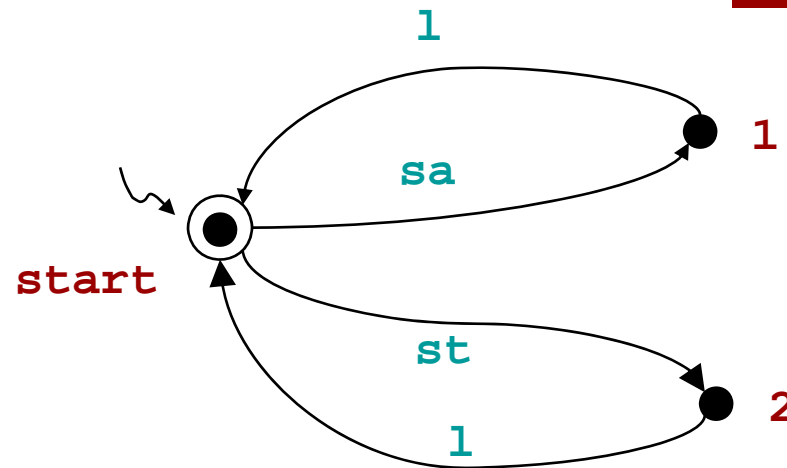
```
<pick name = "Si">  
  <!-- transition #1 -->  
  <onMessage ... >  
    <!-- transition skeleton -->  
  </onMessage>  
  ... ..  
  <!-- transition #N -->  
  <onMessage ... >  
    <!-- transition skeleton -->  
  </onMessage>  
</pick>
```

# 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
  - Transitions going back into 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)

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

```
<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>
```

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>
```

# *Web Service Composition Engine (WS-CE)*



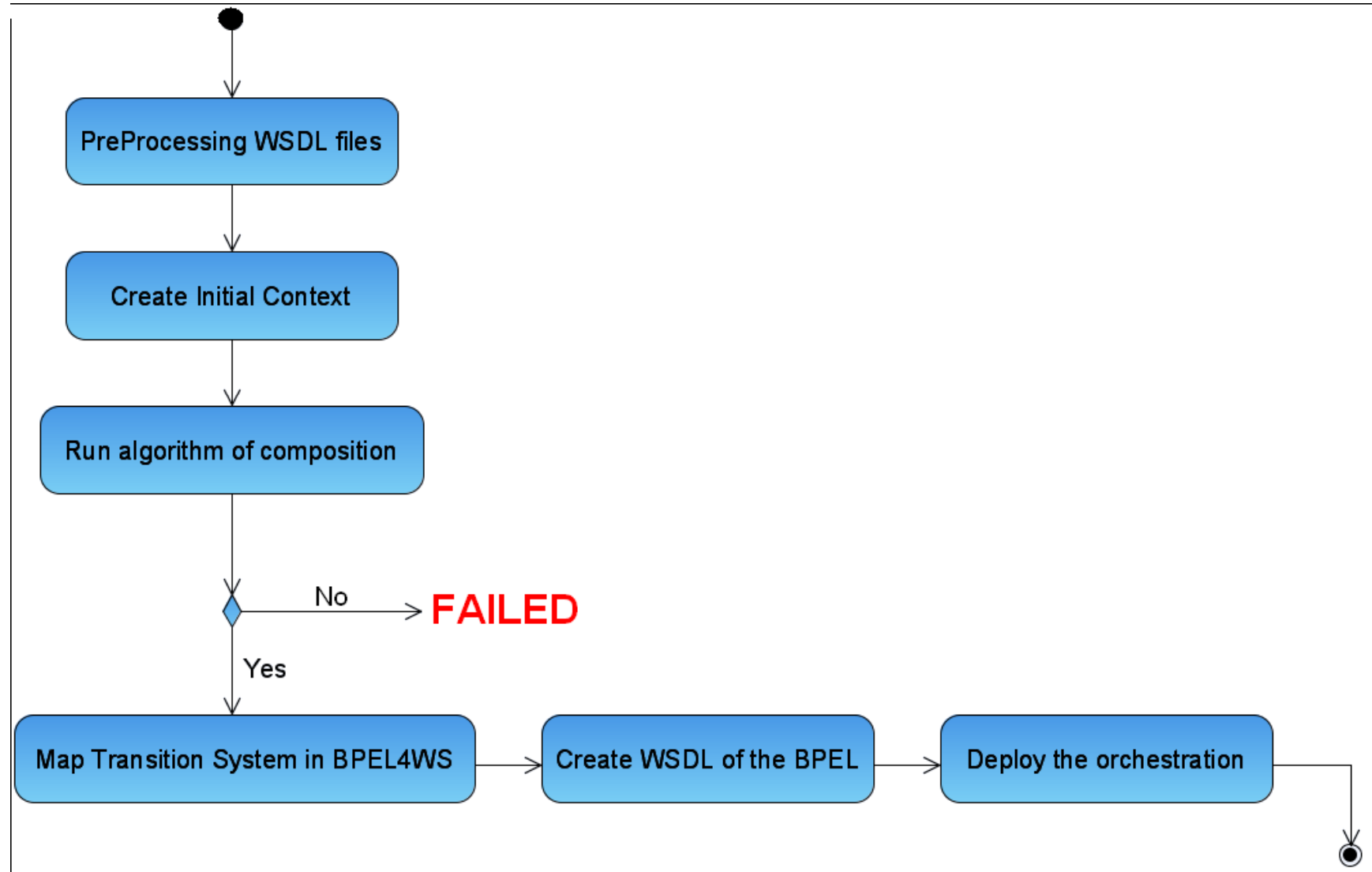
SAPIENZA  
UNIVERSITÀ DI ROMA

---

Each Web Service consists of the *WSDL* document and an *TS* that represent the behavior. Currently all TSs are in *WS-CDL*

The output of the composition process is a *URL* (endpoint) to the WS-BPEL instance of the synthesized process

# *(High level) Program*



# Preprocessing WSDL files<sub>(1)</sub>



Each WSDL file has to be processed in order to obtain descriptors compliant with BPEL specifications

At the the beginning of the file, in <description> element must be declared the *namespace* relative to the partner link type definition  
Take as example SearchMP3.wsdl of the Target Web Service

```
<wsdl:definitions
  targetNamespace="http://localhost:8080/axis/services/SearchMP3"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/services/SearchMP3"
  xmlns:intf="http://localhost:8080/axis/services/SearchMP3"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
.
.
```

# Preprocessing WSDL files<sub>(2)</sub>



In the bottom of the WSDL file must be declared the *partner link type (PLT)*.

A *PLT* describes the kind of the exchanged messages that two WSDL services intend to carry out. A partner link type characterizes this exchange by defining the roles played by each service and by specifying the port type provided by the service to receive messages appropriate to the exchange

```

:
:
    <plnk:partnerLinkType name="SearchMP3PLT" >
        <plnk:role name="SearchMP3Service" >
            <plnk:portType name="impl:SearchMP3" />
        </plnk:role>
    </plnk:partnerLinkType>

</wsdl:definitions>
```

## *Create initial context<sub>(1)</sub>*



1. "kb.txt": contains the knowledge bases.
2. "init.txt": contains the initial state.
3. "actions.xml": contains the correspondence between the action name and the relating input/output message.
4. "moved.xml": contains the correspondence between the proposition moved and the relating *e*-Service.
5. "FsmToMinimize.xml": it is the not minimized FSM relating to the target *e*-Service.
6. "FsmMinimized.xml": it is the minimized FSM of the composed target *e*-Service.

The files 5) and 6) will be created only in the case that it is possible to realize the requested composition.

# Composition Algorithm



```
INPUT:  $S_0$  /* TS of client specification */
        $S_1..S_n$  /* TSs of Services in the Community  $C$  */
OUTPUT: if a composition of  $S_0$  wrt  $S_1..S_n$  exists
        then return  $T_S$  of composition schema
        else return nil

begin
 $\Phi$  = TS_2_ALC( $S_0, S_1, \dots, S_n$ ) /* encode client spec. and services of  $C$ 
                                     into a PDL formula  $\Phi$  */
 $I_f$  = ALC_Tableau( $\Phi$ ) /* compute a finite model  $I_f$  for  $\Phi$  */
if ( $I_f == \text{nil}$ ) /* if  $I_f$  does not exist, i.e., no composition exists */
  then return nil
  else /* else  $I_f$  exist */
     $S_c$  = Extract_TS( $I_f$ ) /* extract a TS from  $I_f$  */
     $T_S$  = Minimize( $S_c$ ) /* minimize it */
    return  $T_S$  /* return it */
end
```

# Mapping TS in BPEL



As previously described we can map the TS in BPEL.

However we have to refine the BPEL document in order to consider the *correlationSet* issue

```
<correlationSets>  
  <correlationSet name="PID"  
                  properties="ns3:ProcessIdentifier"/>  
</correlationSets>
```



# Creating WSDL for BPEL<sub>(1)</sub>



Every BPEL process is exposed like a Web Service and therefore it has a WSDL descriptor file. In WS-CE this descriptor is the WSDL of the Target Service given in input.

This file have to be processed introducing a `<correlationSet>` in order to couple each instance of the process to the appropriate client

A *correlation set* is a set of properties shared by messages. The purpose of the correlation set is to act as a conversation identifier: it keeps together all messages intended for the same conversation.

In order to address this issue it's necessary declare a PID for each client and couple it with all input messages coming from the same client.

# Creating WSDL for BPEL<sub>(2)</sub>



BPEL introduces the concept of *properties* inside the WSDL document in order to obtain a coupling between incoming messages and an instance of process. In order to address this issue we need to define a "key" inside the input message

The *SearchMP3.wsdl*, after being processed, extends the input messages introducing a new *part* element : *ProcessID*. As an example examine the *SearchByAuthorRequest* message.

```
<wsdl:message name="SearchByAuthorRequest">  
  <wsdl:part name="authorName" type="xsd:string" />  
  <wsdl:part name="ProcessID" type="xsd:int" />  
</wsdl:message>
```

# Creating WSDL for BPEL<sub>(3)</sub>



The *property alias* in our example is called `ProcessIdentifier`. Below is shown the *property* in `SearchMP3.wsdl`

```
<bpws:property name="ProcessIdentifier" type="xsd:int"/>
<bpws:propertyAlias messageType="impl:SearchByTitleRequest "
    part="ProcessID" propertyName="impl:ProcessIdentifier"/>
<bpws:propertyAlias messageType="impl:ListenRequest "
    part="ProcessID" propertyName="impl:ProcessIdentifier"/>
<bpws:propertyAlias messageType="impl:ErrorMessageRequest "
    part="ProcessID" propertyName="impl:ProcessIdentifier"/>
<bpws:propertyAlias messageType="impl:SearchByAuthorRequest "
    part="ProcessID" propertyName="impl:ProcessIdentifier"/>
```

Follow the declaration of the `<correlationSet>` in BPEL :

```
<correlationSets>
    <correlationSet name="PID" properties="ns3:ProcessIdentifier"/>
</correlationSets>
```

## *Deploy of the composed WS*



The deploy of the BPEL process is composed by the following operations :

1. Find the **PID** to associate at the current instance of the process.
2. Create a **proxyService** in order to hide the correlation management to the client.
3. Create the **package** runnable under the ActiveBpel engine.

## *Deploy of the composed WS* *Find the PID.*



WS-CE has a repository containing the interface of the deployed processes. Each of these are identified by a number (the PID). So every new instance of a process has to be associated with a PID.

The interface of the process is able to handle the correlation set. The client, on the other hand, doesn't know the current PID and it would be very hard re-write the software in order to associate the correct PID with the instance of the process.

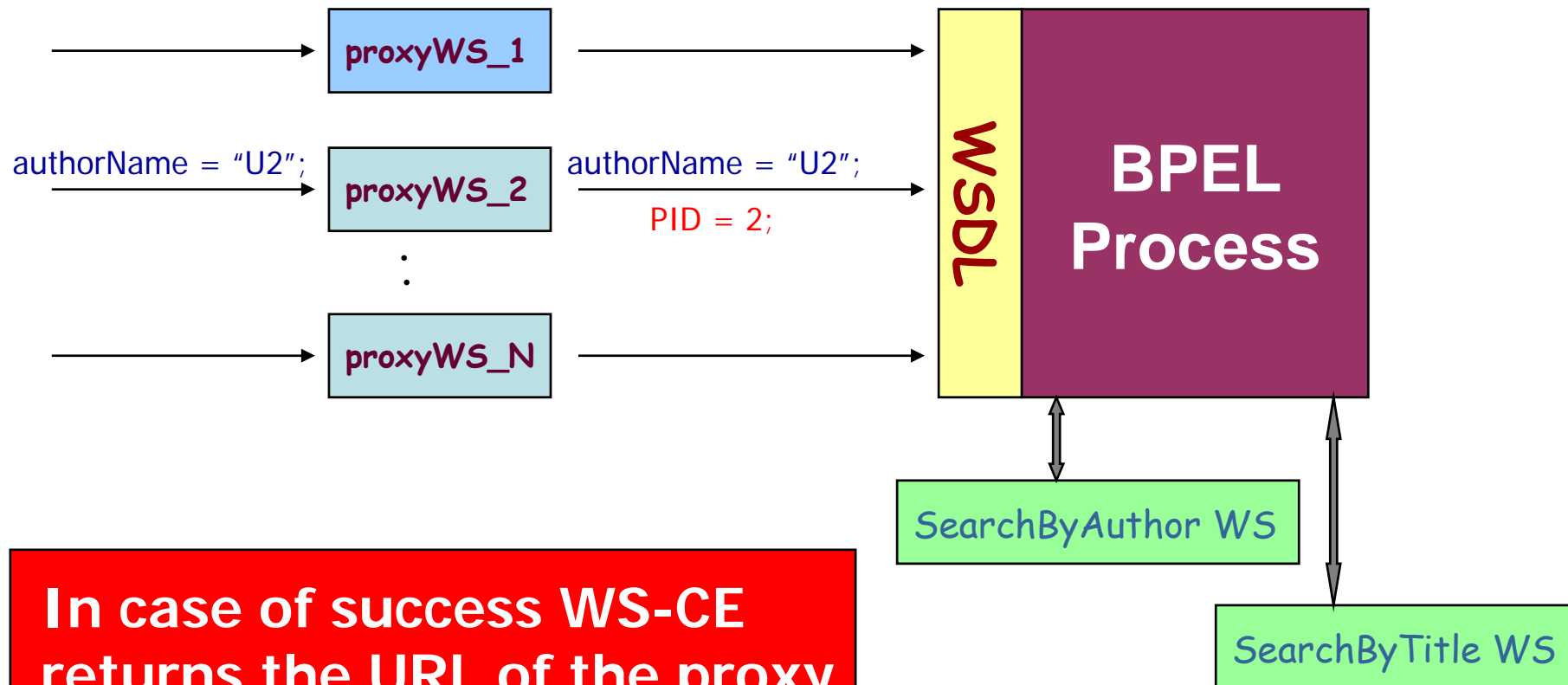
In order to avoid this issue, WS-CE creates a *proxyService*. This is a Web Service which receives the request from the client, puts in line the PID, and dispatches the message to the correct instance of process

# Deploy of the composed WS

## Example of proxyService.



Consider the case where a client looks for the list of "U2" songs.



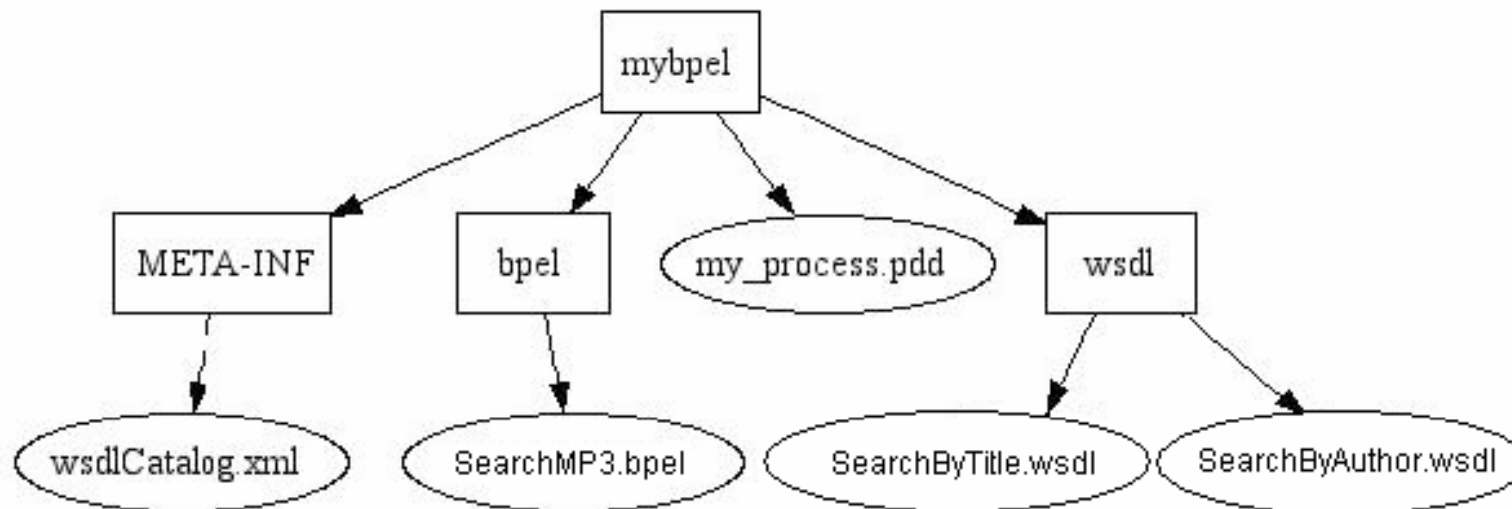
**In case of success WS-CE returns the URL of the proxy service to the client**

# Deploy of the composed WS

## Create the package



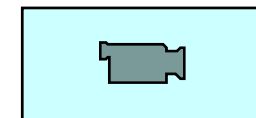
Deploying a BPEL process involves creating a deployment archive file (a JAR with an extension of ".bpr") and copying that file in the servlet container. To create this archive, we need to organize the files into a particular directory structure, create one or two configuration files, and then create an archive from that directory.



# *A Demo Application*



- Automatic Composition can be used for substituting unavailable services with new ones synthesized on-the-fly
- A demo application has been developed in the context of the MAIS project (<http://www.mais-project.it>), in which WS-CE has been incorporated into a complex service platform
- When a client (application) of a Web service receives an error, a request for composition is sent to WS-CE, that synthesizes a new services, deploys it and returns the new service endpoint

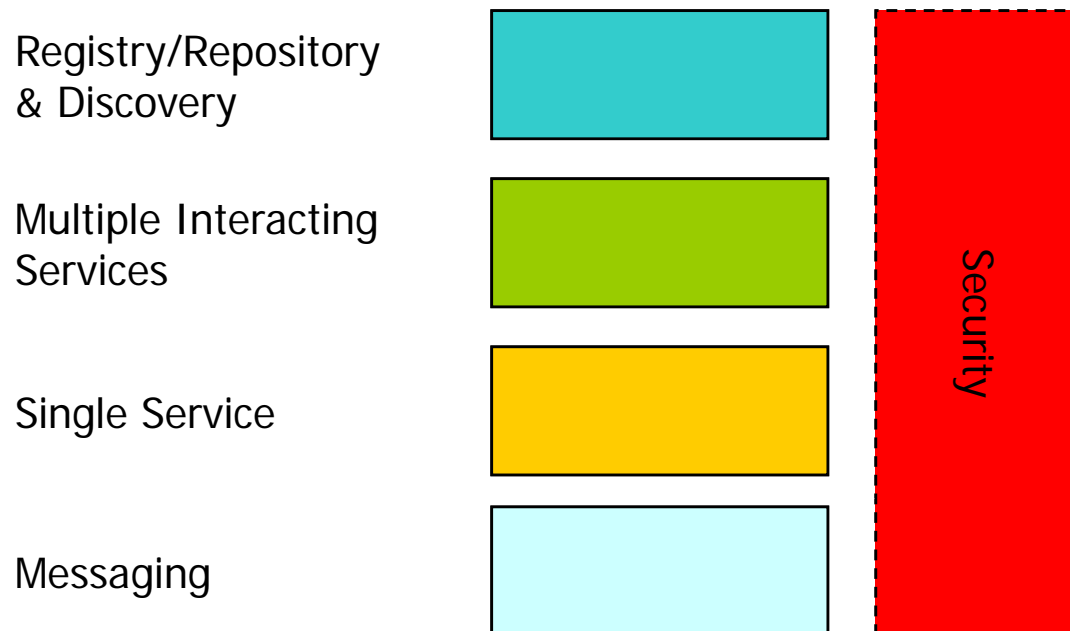




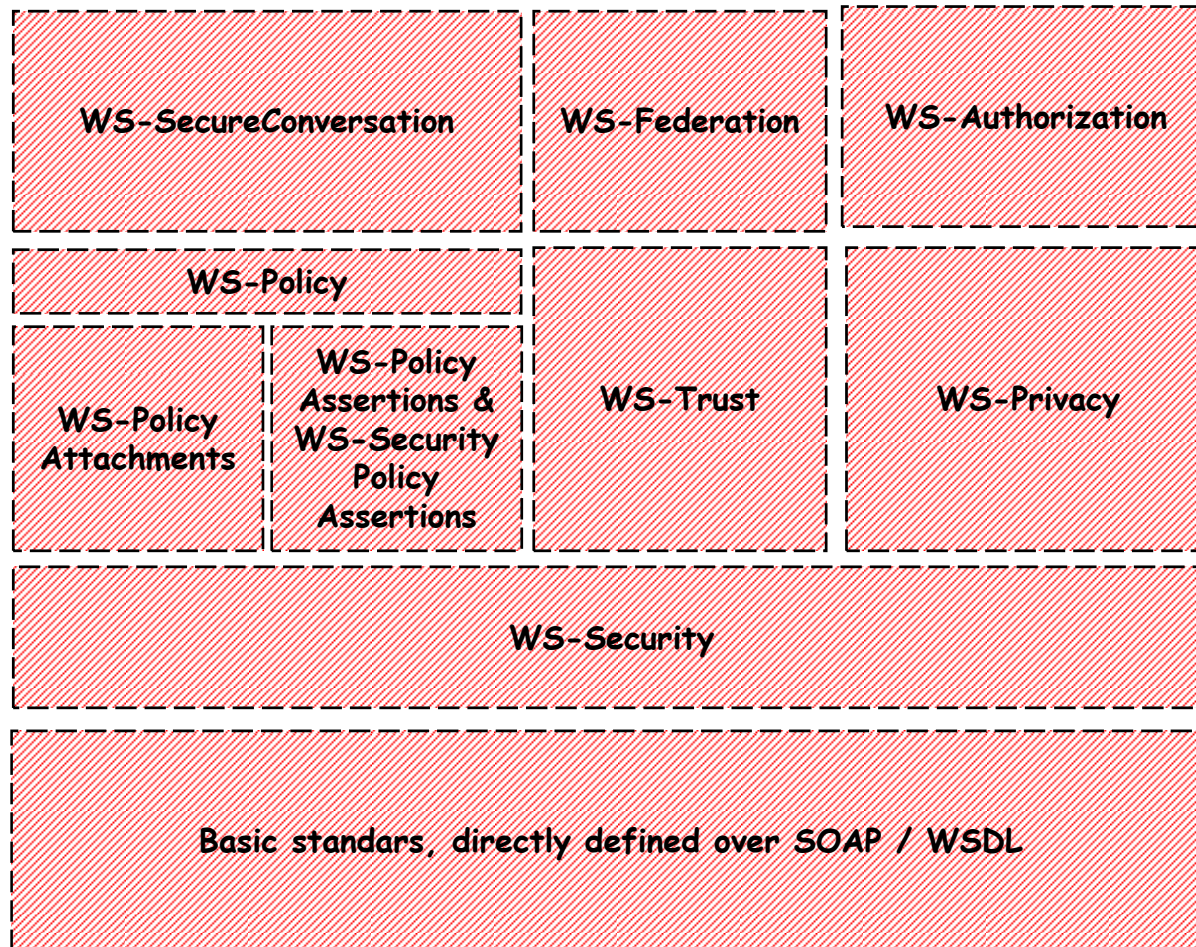
# Security



- Standardization bodies are trying to define every facet of SOC, even security



# Security Standards (1)



# *Security Standards (2)*



- *"Secure" Channel*

- Provides the abstraction of "secure & confidential" communication channel

- XML Digital Signature: to sign parts of XML documents (and therefore parts of the messages exchanged between client and WS)
- XML Encryption: to cypher parts of XML Documents
- XML Key Management Services: the interface of basic Web Services for processing and management of keys based on PKI
- WS-Security: integrity and confidentiality end-to-end of messages

# Security Standards (3)

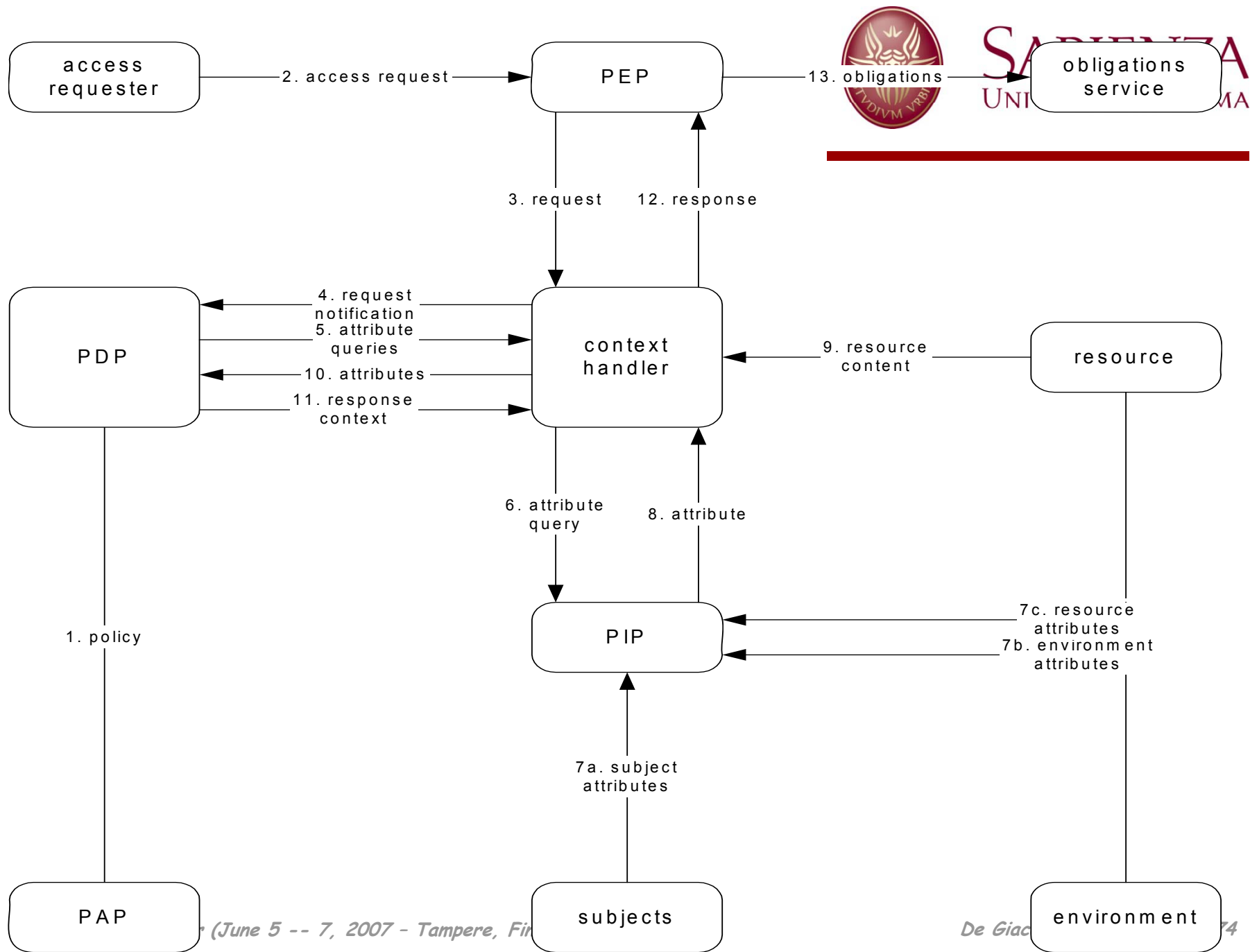


- **Description of Features**
  - The WS-Policy family allows the description of requirements and capabilities of a WS, in order to provide clients with needed information
  - WS-PolicyAttachment: to link assertions to a WS
  - WS-PolicyAssertions and WS-SecurityPolicyAssertions are the languages for expressing such assertions
- **Trust**
  - WS-Trust defines a model for establishing trust between client and WS, based on third parties (Security Token Services - to be realized as infrastructural services)
  - Definition of protocols and interfaces for verifying authenticity and freshness of the tokens presented by the subjects
- **Secure Conversations**
  - WS-SecureConversation: mechanisms for establishing and exchanging security contexts, to be used during exchanges of messages belonging to the same conversation
- **Others (not yet mature)**
  - WS-Privacy
  - WS-Authorization
  - WS-Federation

## *Security Standards (4)*



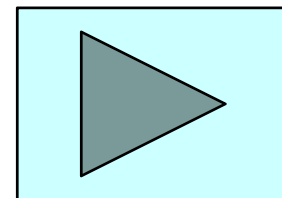
- SAML (Security Assertion Markup Language)
  - XML framework for exchanging authentication and authorization information for securing Web services
- XACML (eXtensible Access Control Markup Language)
  - XML framework for specifying access control policies for Web-based resources



# *Trust-aware Composition*



- Service in a community may not trust each other at the same level
- They may pose conditions on credentials presented by clients
- Taking care of these aspects during composition is feasible



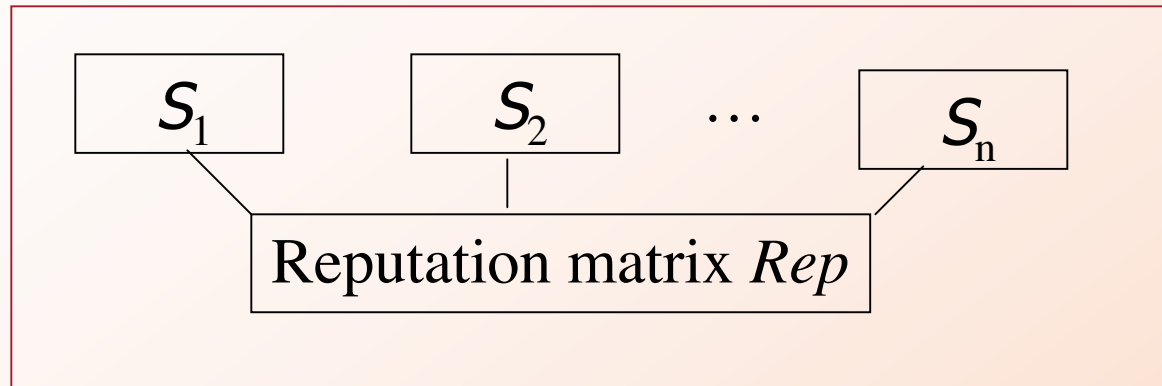
# Example

- We consider a community of services for searching and listening mp3 files
- To use these services some credentials are needed



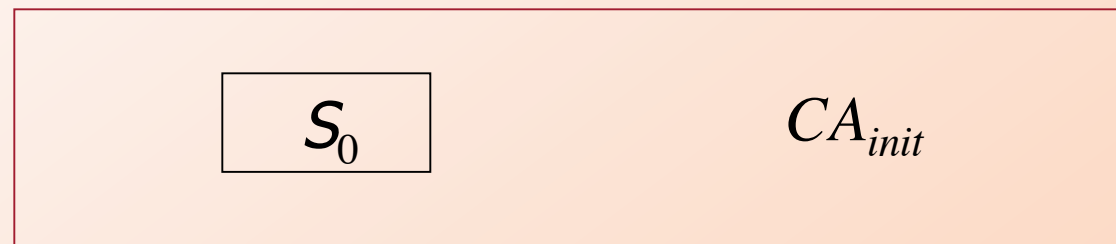
# General view

Community  $S$



Credentials  
 $C = \{c_1, \dots, c_m\}$

Client



# Community

- A community  $S$  is formed by a finite set of available services  $\{S_1, \dots, S_n\}$  that share the same set of actions  $A$
- Reputation matrix  $Rep : Rep(i,j)$  represents the reputation level that the service  $S_i$  has on the service  $S_j$

# Example

- We consider a community  $S = \{S_1, S_2, S_3\}$

<i>Rep</i>	$S_1$	$S_2$	$S_3$
$S_1$	5	0	5
$S_2$	0	5	0
$S_3$	0	0	5

# Credentials

- Assertions about the client, issued by a given service
- $C = \{c_1, \dots, c_m\}$ : finite set of credentials associated to the client
- $c_h = (Attr, Issuer)$

*Attr*: Attribute variable ranging over  $\Delta$

*Issuer*: Issuer variable ranging over  $I$

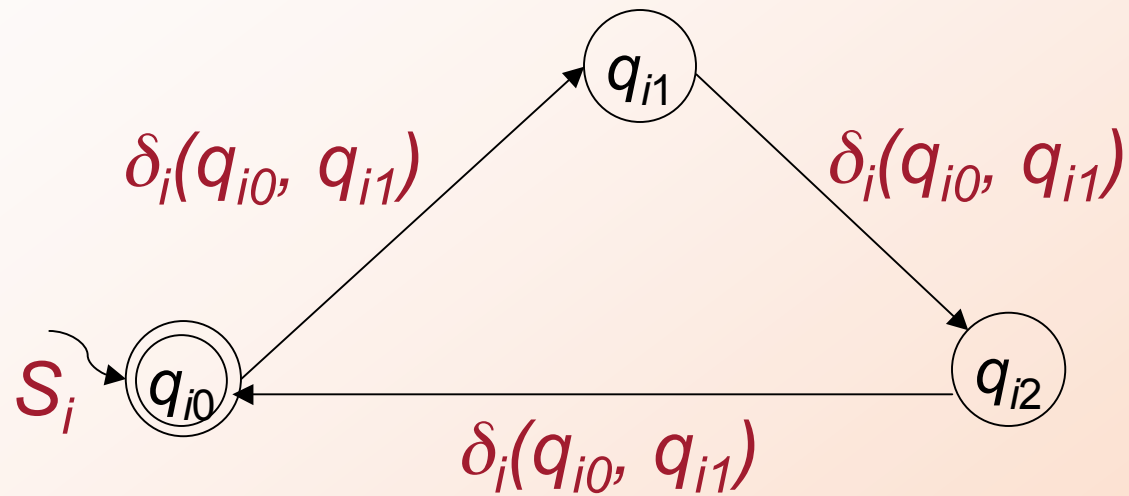
# Example

- $C = \{c_1\}$
- $c_1 = (\textit{Issuer}, \textit{Inscribed})$

# Available Services

- An available service  $S_i$  is defined in terms of a transition system  $TS_i$
- $TS_i = (Q_i, q_{i0}, G_i, \delta_i, F_i)$ 
  - $Q_i$  : finite set of states
  - $q_{i0}$  : single initial state
  - $G_i$  : set of guards
  - $\delta_i$  : transition function
  - $F_i$  : set of final states

# Available Services



# Available Services

- Transition function

$$\delta_i(s_i, s_i') \subseteq G_i \times A \times \Gamma$$

- Guards

- Atomic guards:

$$g := \text{Rep}(i, c_h.\text{Issuer}) \leq v \mid c_h.\text{Attr} \leq v$$

$c_h$ : credential of the client

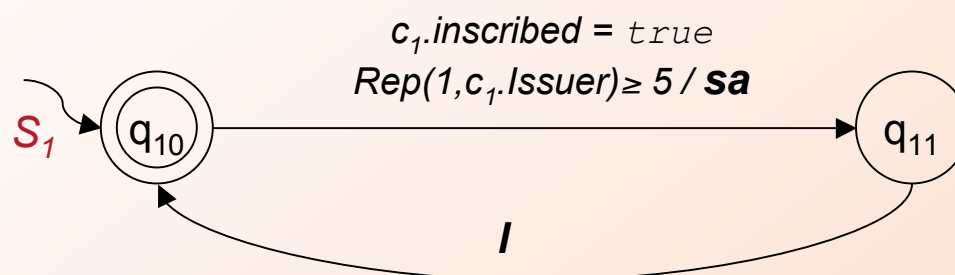
$v$ : value in  $\Delta$

-  $\psi$  : set of closed FOL formulas

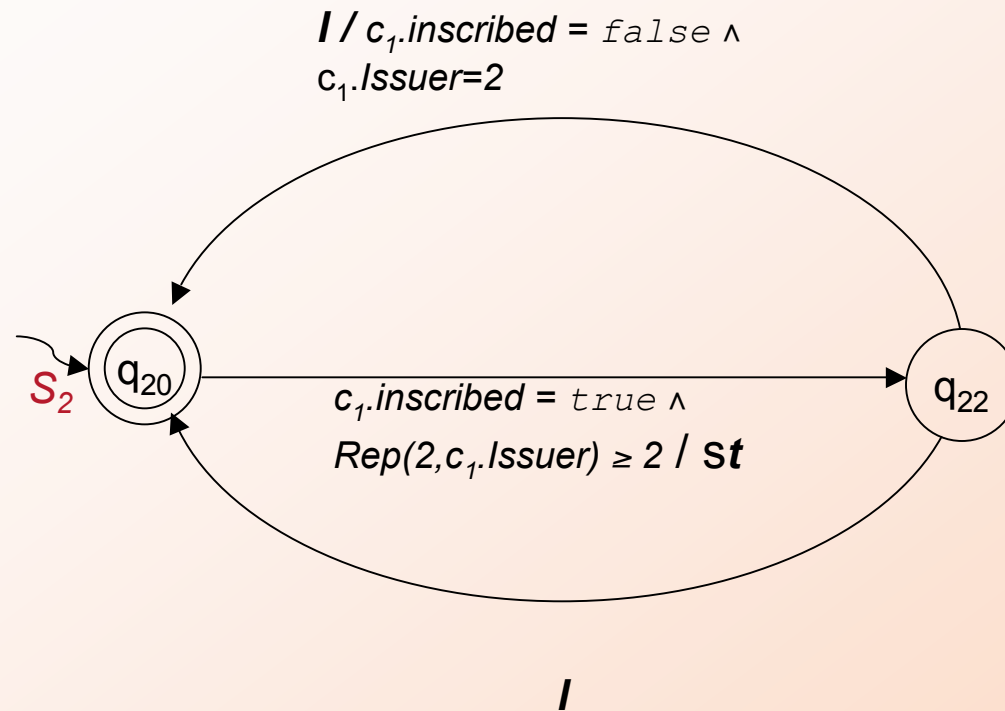
- $\Gamma$  : reassignment for credential variables



# Example



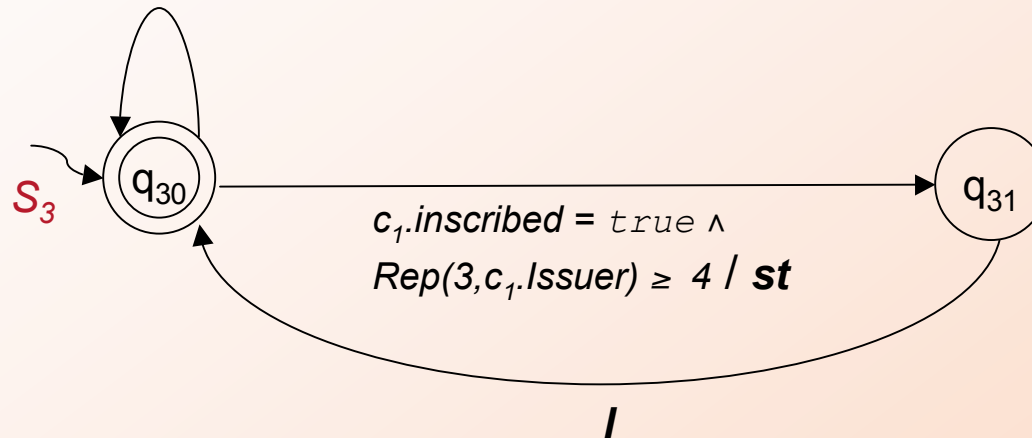
# Example



# Example

*i /*

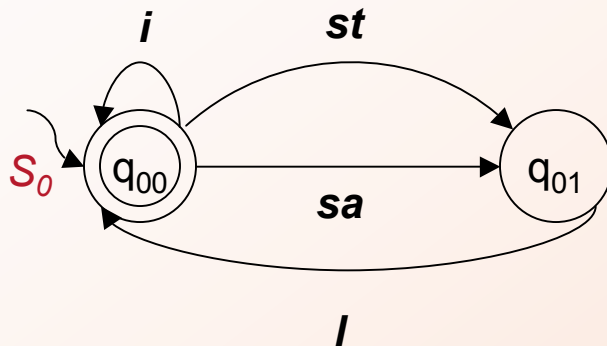
$c_1.inscribed = true \wedge c_1.Issuer = 3$



# Target Service

- $TS_0 = (Q_0, q_{00}, G_0, \delta_0, F_0)$
- Transition are not labeled by reassignments
- Guards can refer only to attribute variables of credentials
- Transitions are deterministic
- Initial assignment  $CA_{init}$

# Example



$$CA_{init}(c_1.Issuer)=S_3$$

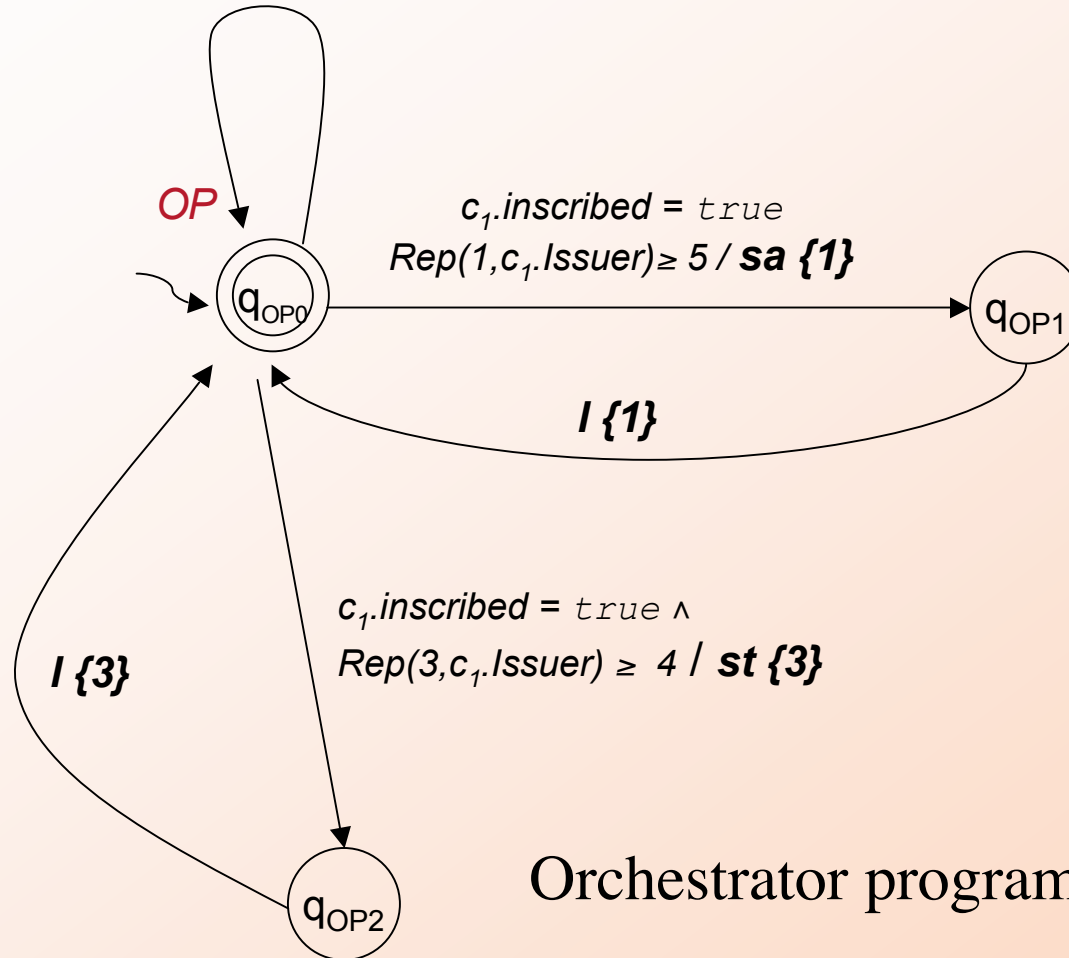
$$CA_{init}(c_1.Inscribed)=false$$

# Orchestrator Programme

- $OP: \mathcal{H} \times A \rightarrow \{1, \dots, n\}$
- $\mathcal{H}$ : set of all histories  $h$
- $h = (q_1^0, \dots, q_n^0, CA^0) \cdot a^1 \cdot (q_1^1, \dots, q_n^1, CA^1) \dots a^j \cdot (q_1^j, \dots, q_n^j, CA^j)$
- $CA^0 = CA_{init}$
- $t = a^1 \cdot a^2 \dots a^j$ : trace of a target service
- $q_i^0 = q_{i0}$ ,  $i = 1, \dots, n$
- for one  $i$ ,  $(q_i^k, g_i, a, \gamma, q_i^{k+1}) \in \delta_i$ ,  $g_i = true$  in  $CA^k$  and  $CA^{k+1} = CA^k \circ \gamma$ ,  $1 \leq k \leq l$
- $q_j^{k+1} = q_j^k$ ,  $j \neq i$

# Example

$i \{3\} / c_1.inscribed =$   
 $true \wedge c_1.issuer = 3$



# Composition Problem

- **Input:**

target service  $S_0$ , set  $C$  of credentials, initial assignment  $CA_{init}$ , community  $S = \{S_1, \dots, S_n\}$  and reputation matrix  $Rep$

- **Output:**

checking the existence of an orchestrator program for  $S$  that realizes the target service



# Complexity

- Theorem

The composition problem is Exptime-complete

Proof.

1- We reduce our problem to satisfiability in Propositional Dynamic Logic (upper bound)

2- The problem with deterministic services and without credentials is Exptime-hard (lower bound)

# *Distributed Composition Scenario (1)*



- Emergency management based on MANETs
  - Each team member is typically equipped with handheld devices (PDAs) and communication technologies
  - through the interplay with the software running on the device, can execute specific actions
- The team member and his device offer a service towards the other members, and an overall workflow coordinates the actions of all the services

# *Distributed Composition Scenario (2)*

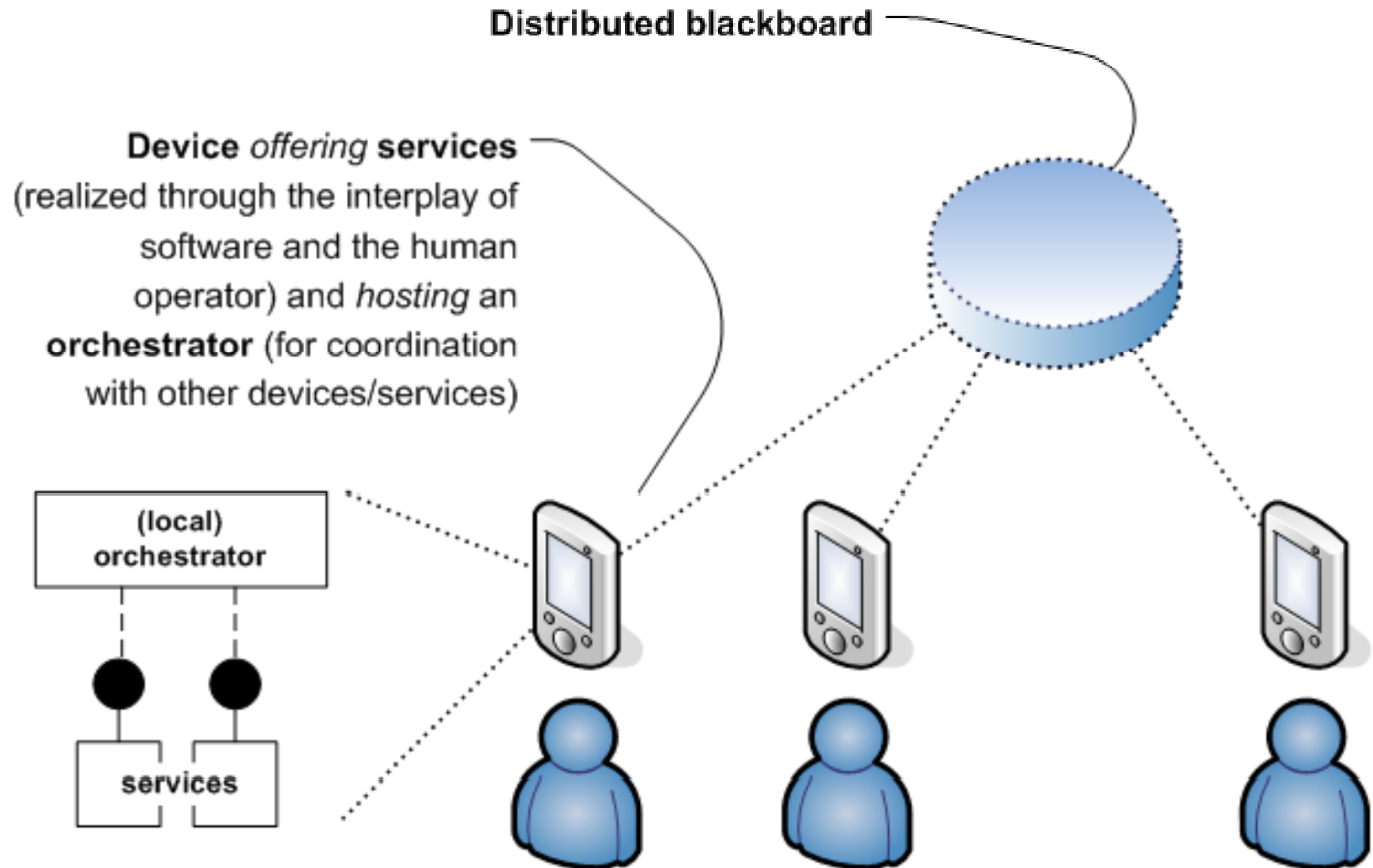


- Actions offered by such mobile services are typically constrained
  - if a service A is instructed to take some photos, then it needs to be instructed to forward them to another storage device B (and no other photos can be taken until the forwarding is executed), as the device offering A has not enough storage space to keep multiple photos
- The effects of such actions can not be foreseen, but can be observable afterwards

# *Distributed Composition Scenario (3)*



- Generic workflows for the different teams are designed a-priori, and then, just before a team is dropped off in the operation field, they need to be instantiated on the basis of the currently available services offered by the mobile devices and operators effectively composing the team
- The effective workflow to be enacted by the team, through the offered services, cannot be *centrally* orchestrated, as in general devices may not be powerful enough
- Decentralized orchestrators (one for each device/service) should *distributively* coordinate the workflow, through the appropriate exchange of messages, conveying synchronization information and the outputs of the performed actions by the services



# *The Setting (1)*



- Non-deterministic services
- Workflow specified on the basis of a *set of available actions* and a *blackboard*, i.e., a conceptual shared memory in which the services provide information about the output of an action (cfr. complete observability wrt. the orchestrator)
  - Workflow Specification Kit (WfSK)
- Such a workflow is specified a-priori without knowing which effective services are available for its enactment
- How to compose (i.e., realize) such a workflow by suitably orchestrating available services
  - When a team leader, before arriving on the operation field, by observing (i) the available devices and operators constituting the team (i.e., the available services), and (ii) the target workflow the team is in charge of, need to derive the orchestration

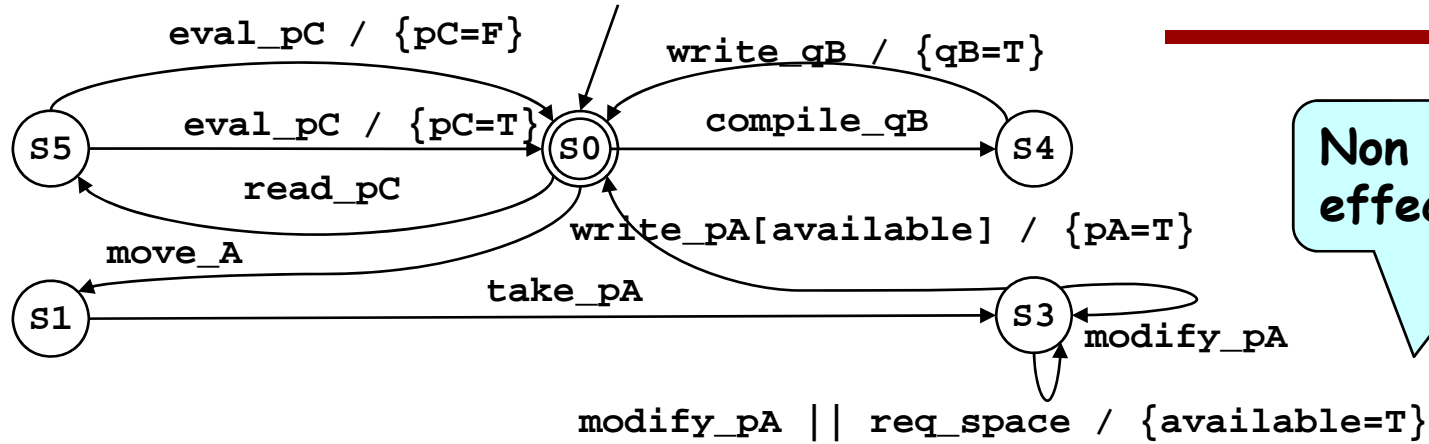
# *The Setting (2)*



- At run-time (i.e., when the team is effectively on the operation field), the orchestrator coordinates the different services in order to enact the workflow.
- The communications between the orchestrator and the services are carried out through appropriate middleware, which offers broadcasting of messages and a possible realization of the blackboard
- The orchestrator is distributed, i.e., there is not any coordination device hosting the orchestrator; conversely, each device, besides the service, hosts also a local orchestrator
- All the orchestrators, by appropriately communicating among them, carry on the workflow in a distributed fashion
- Also the blackboard, from an implementation point of view, is realized in a distributed fashion

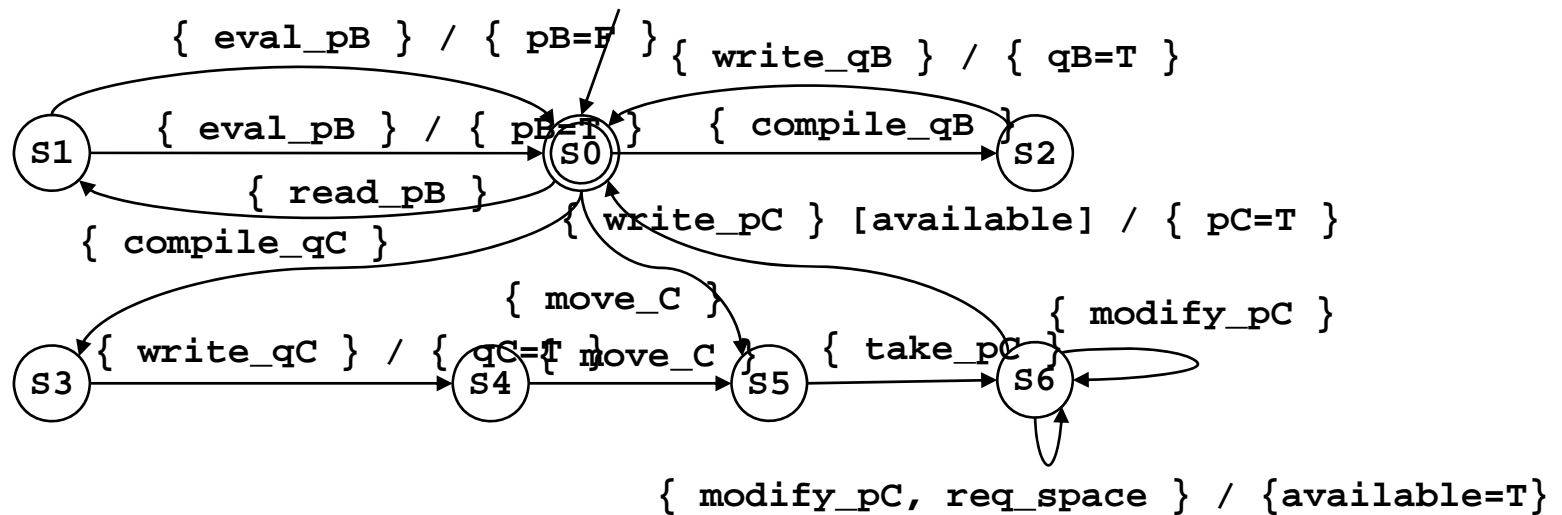


BEHAVIOUR A



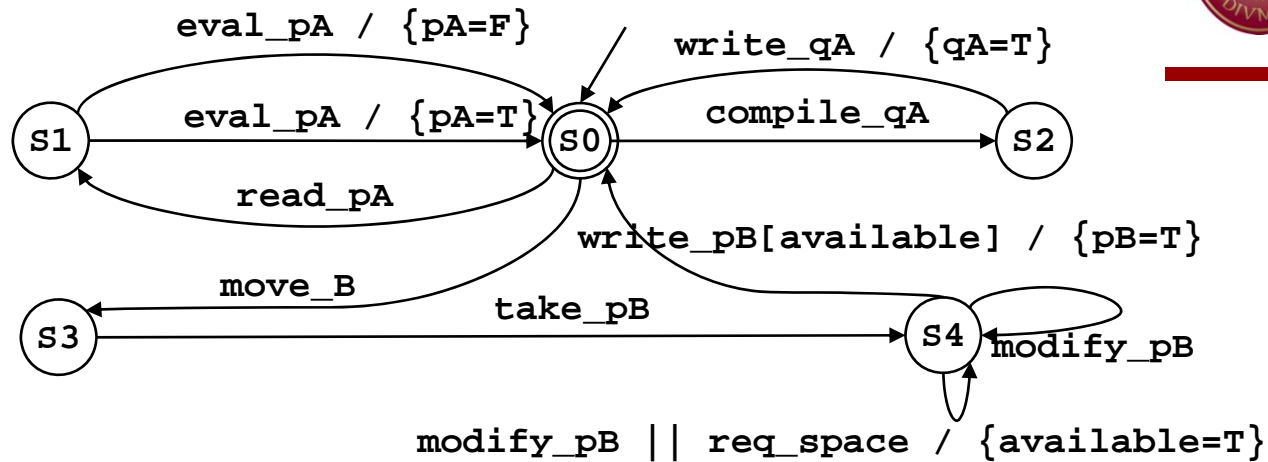
Non deterministic effects

BEHAVIOUR B

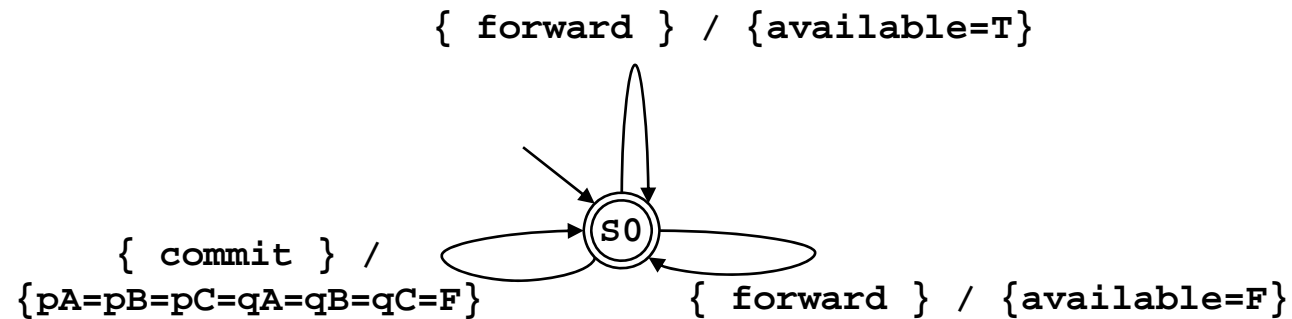




## BEHAVIOR C

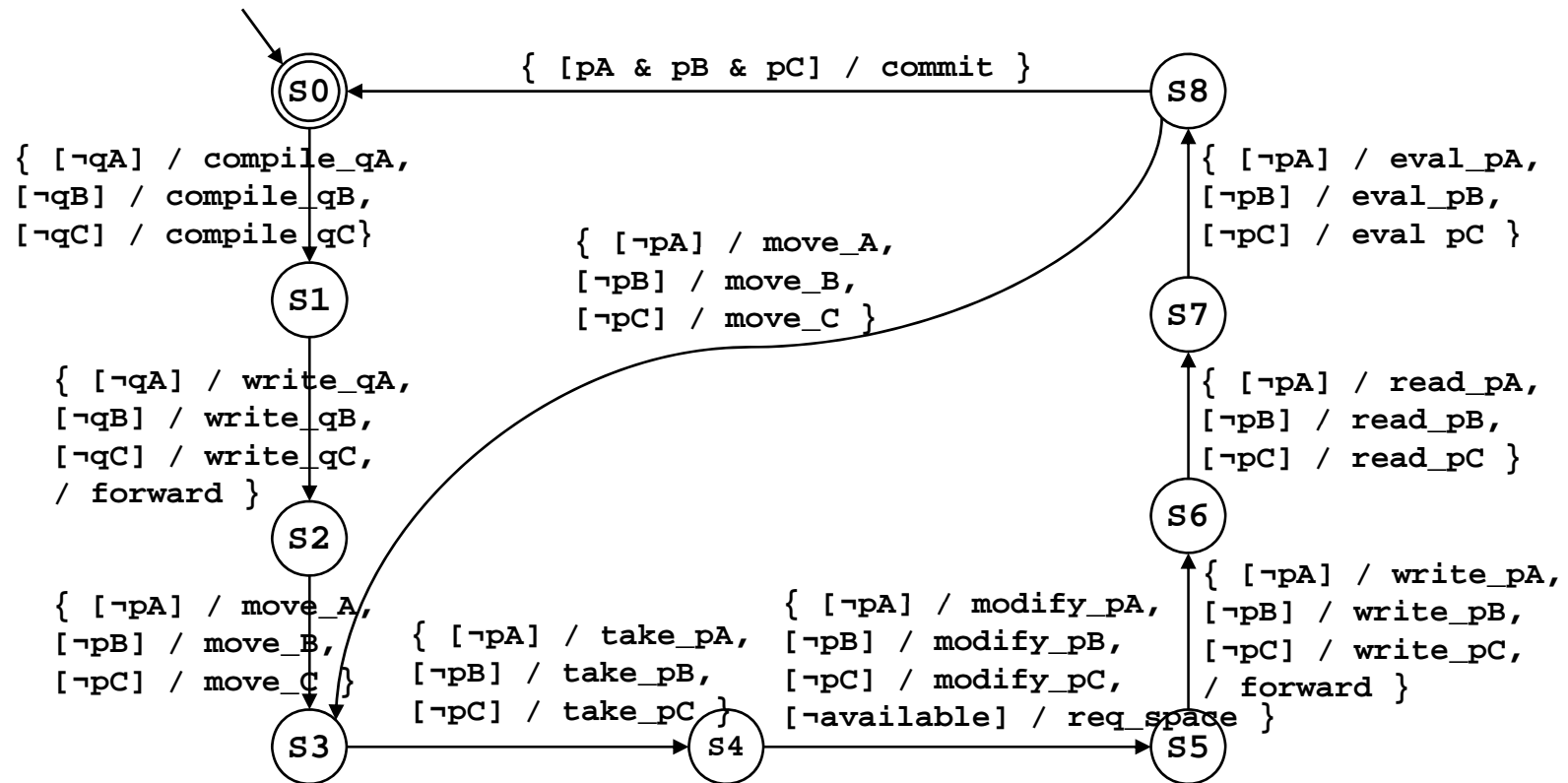


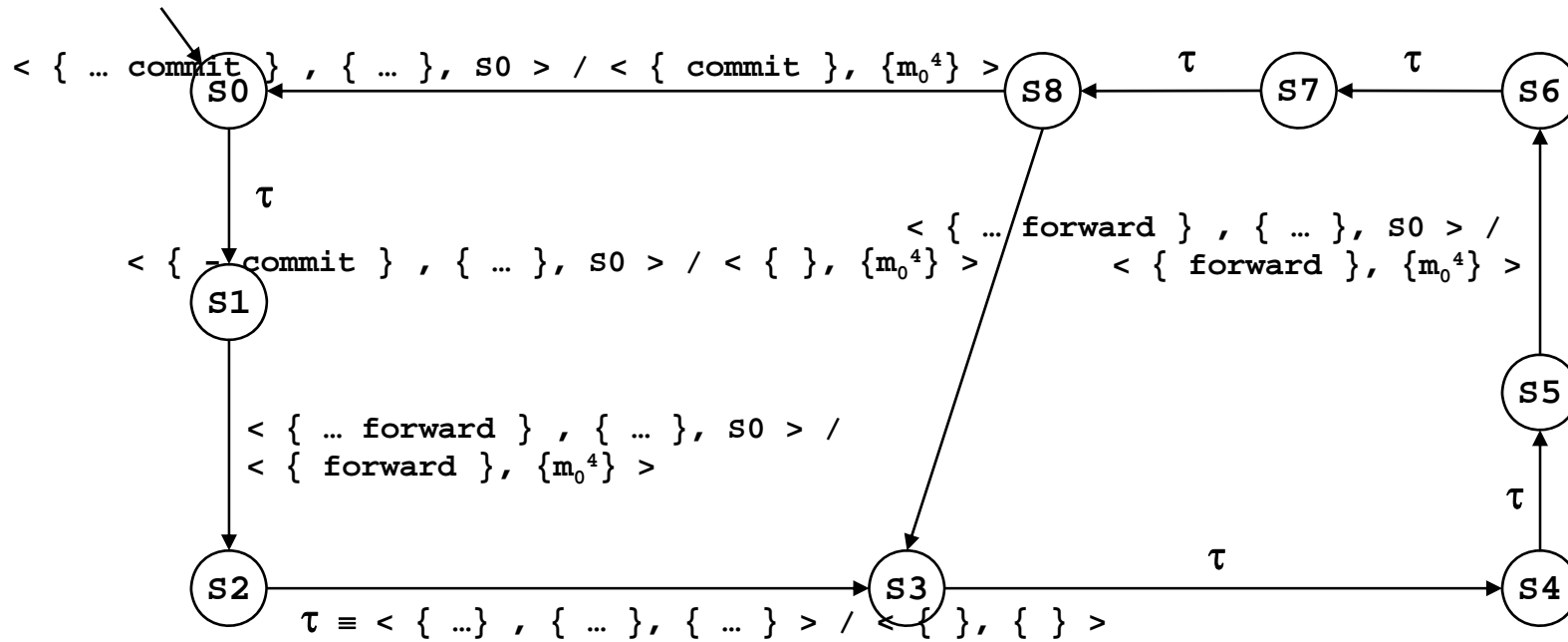
## BEHAVIOUR REPOSITORY





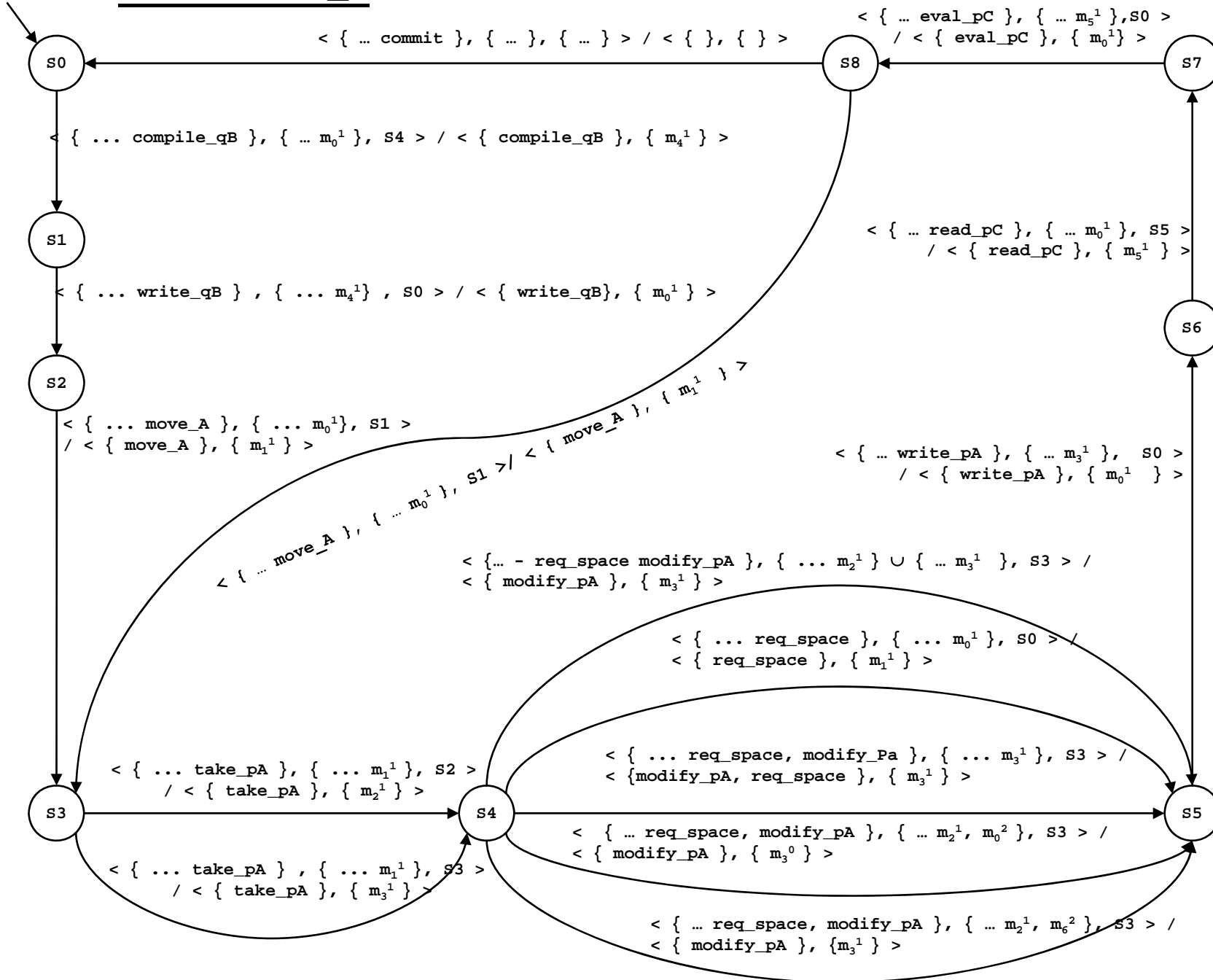
**TARGET**



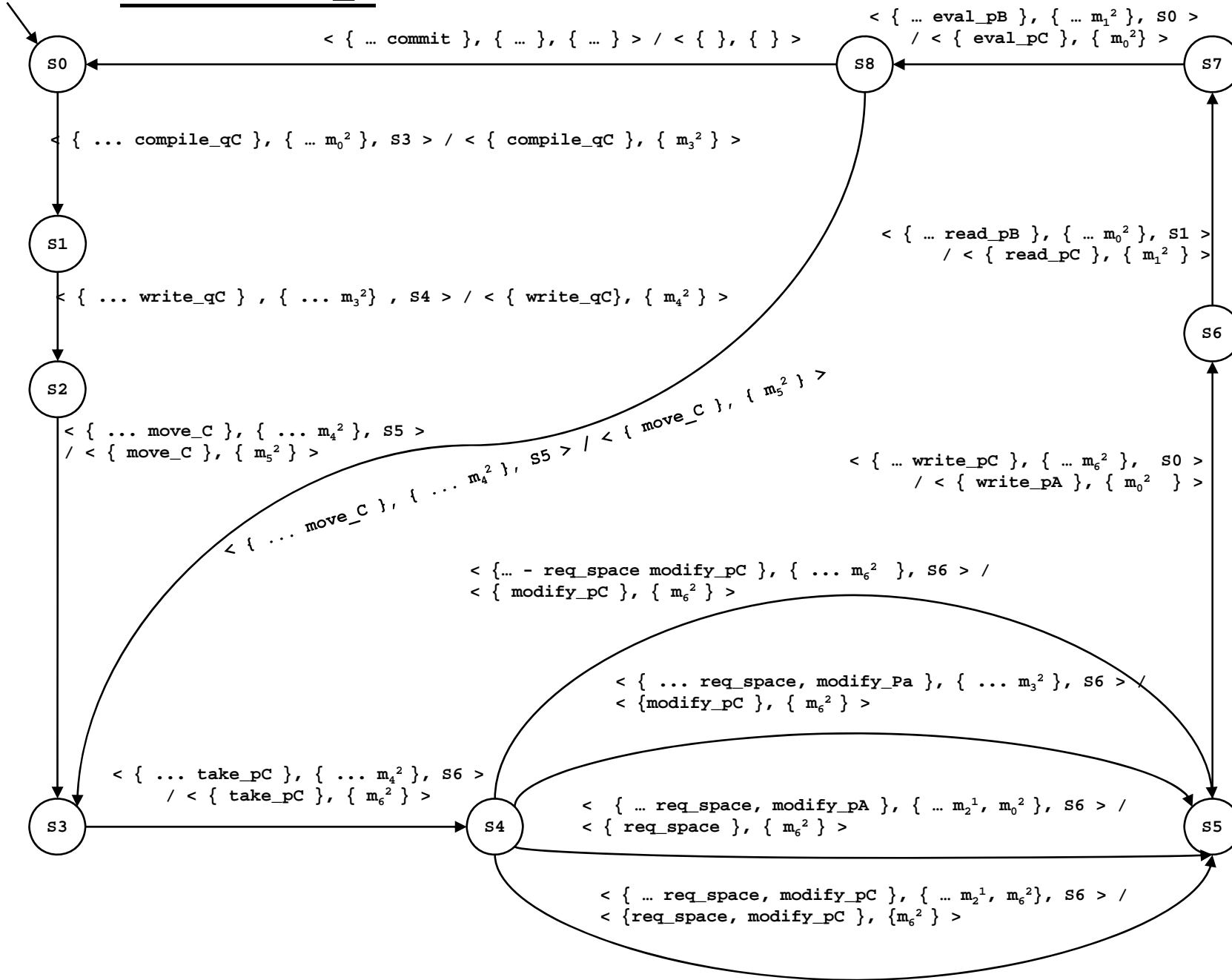


CONTROLLER\_REPOSITORY

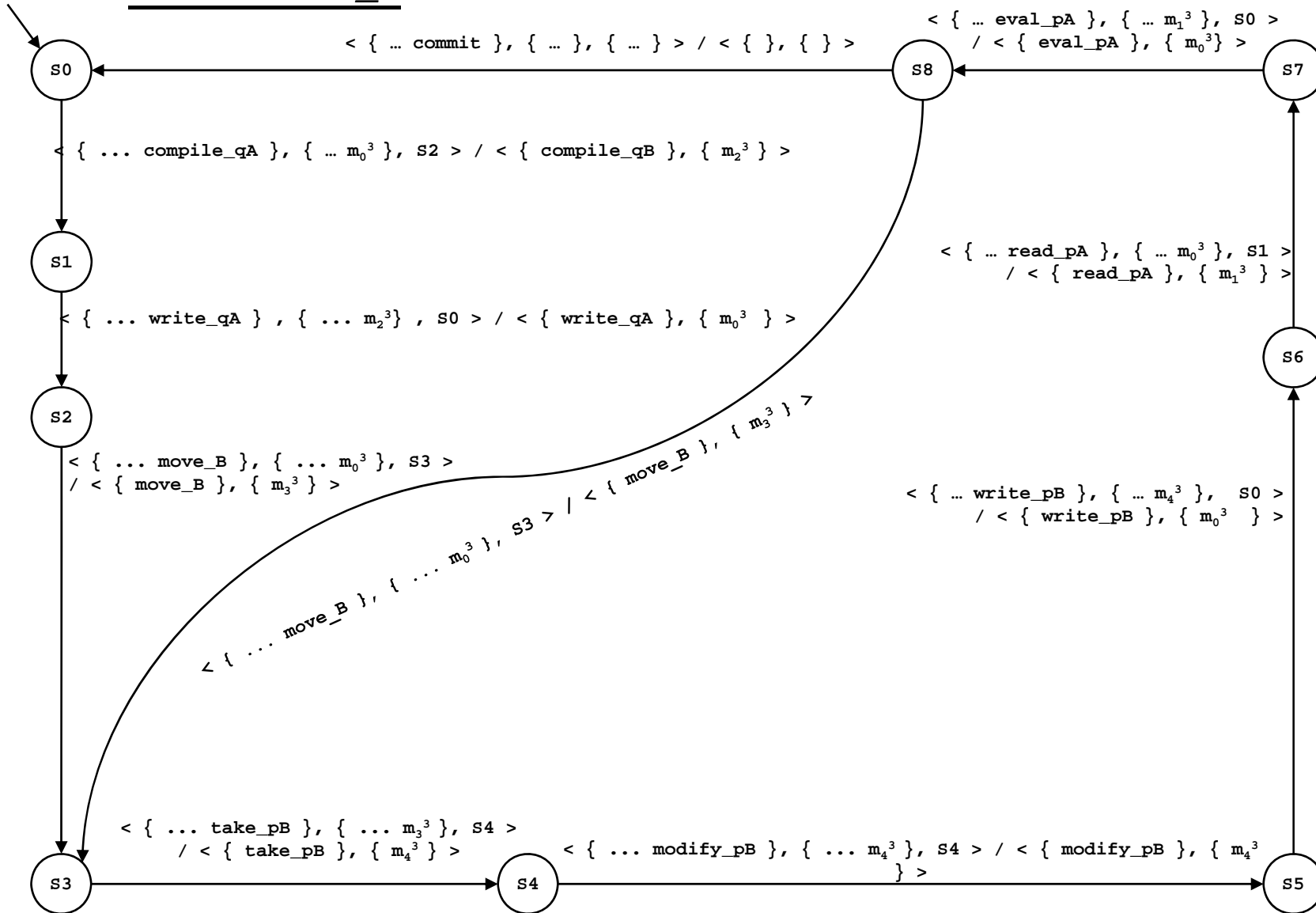
# CONTROLLER\_A



# CONTROLLER\_B



# CONTROLLER\_C



# The Result



[de Leoni, De Giacomo, Mecella, Patrizi @ International Conference on Web Services 2007]

*There exists a sound, complete and terminating procedure for computing a distributed orchestrator  $X = (O_1, \dots, O_n)$  that realizes a workflow  $W$  over a WfSK  $K$  relative to services  $S_1, \dots, S_n$  over  $K$  and blackboard states. Moreover each local orchestrator  $O_i$  returned by such a procedure is finite state and require a finite number of messages (more precisely message types)*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

# *Appendixes*



# OWL-S (formerly DAML-S)



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



[from DAML-S]

# *OWL-S Service Profile*

## *(What it does)*



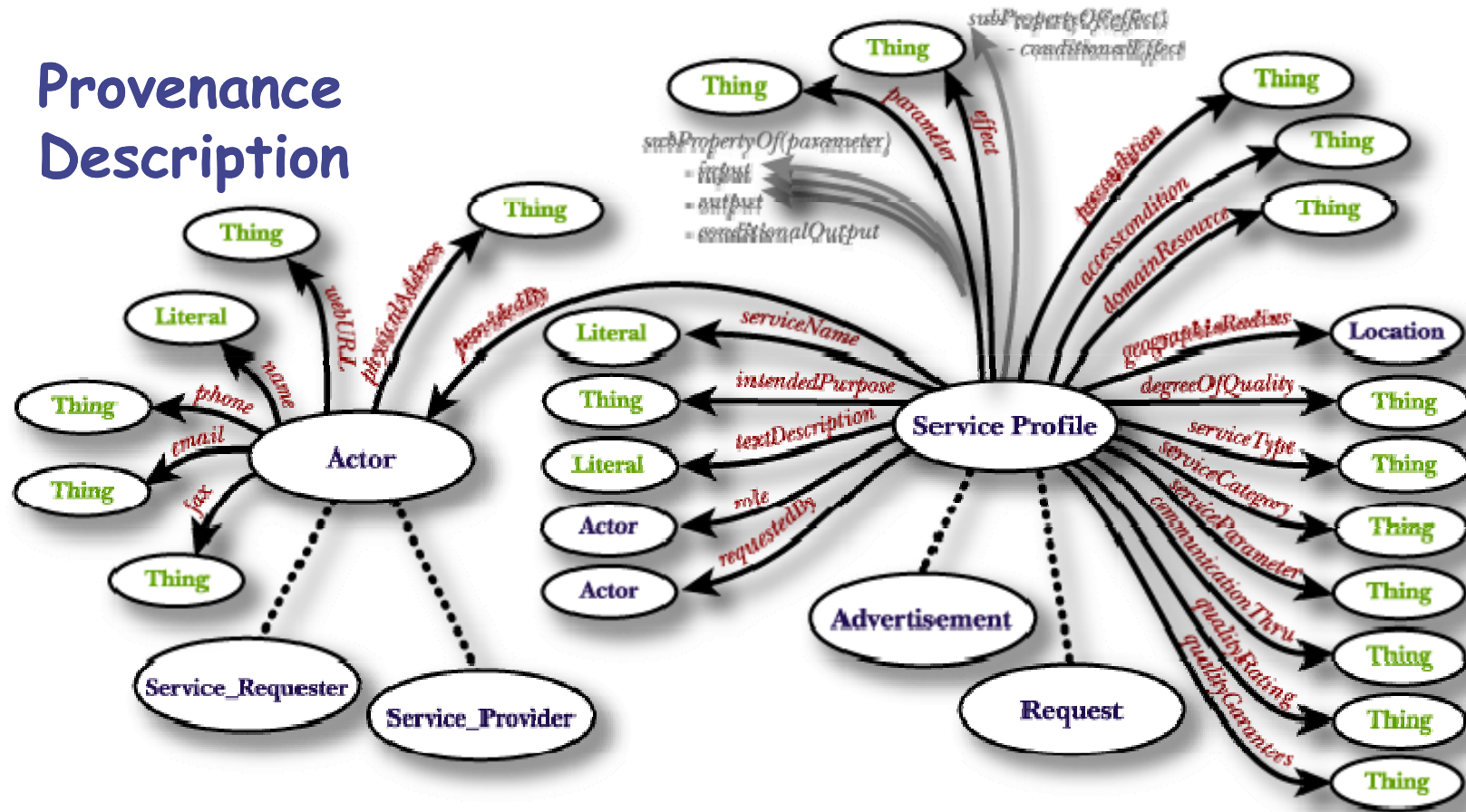
- High-level characterization/summary of a service
  - Provider & participants
  - Capabilities
  - Functional attributes (e.g., QoS, region served)
- Used for
  - Populating service registries
    - A service can have many profiles
  - Automated service discovery
  - Service selection (matchmaking)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

# OWL-S Service Profile



## Capability Description

## Provenance Description



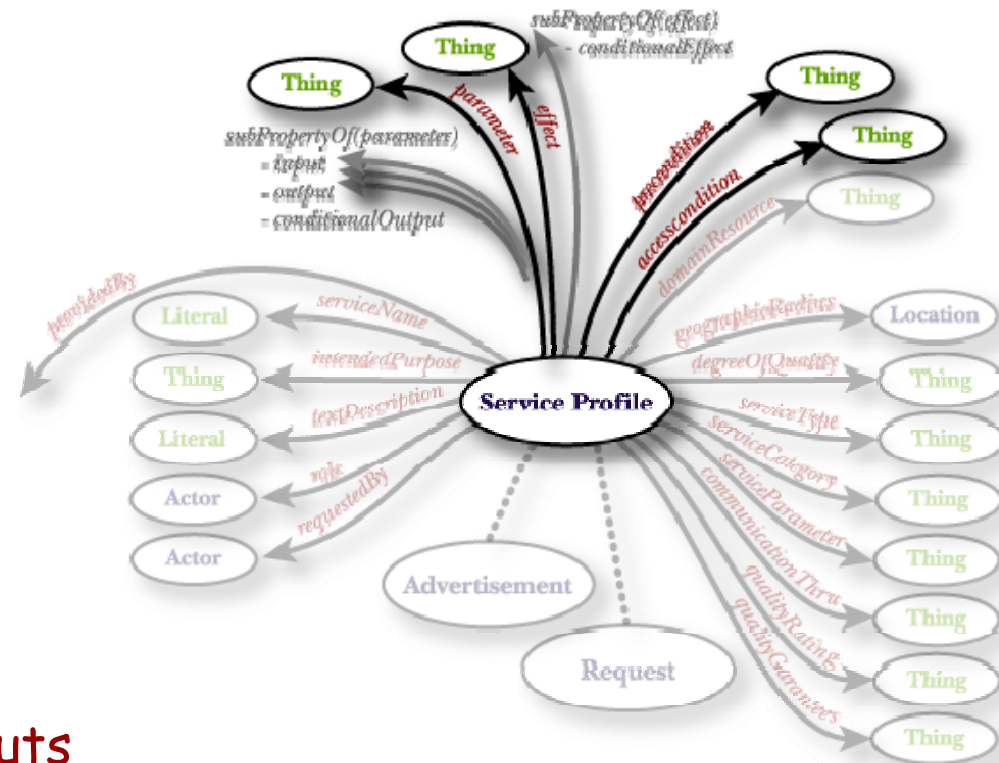
## Functional Attributes

[from DAML-S]

# Capability Description



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



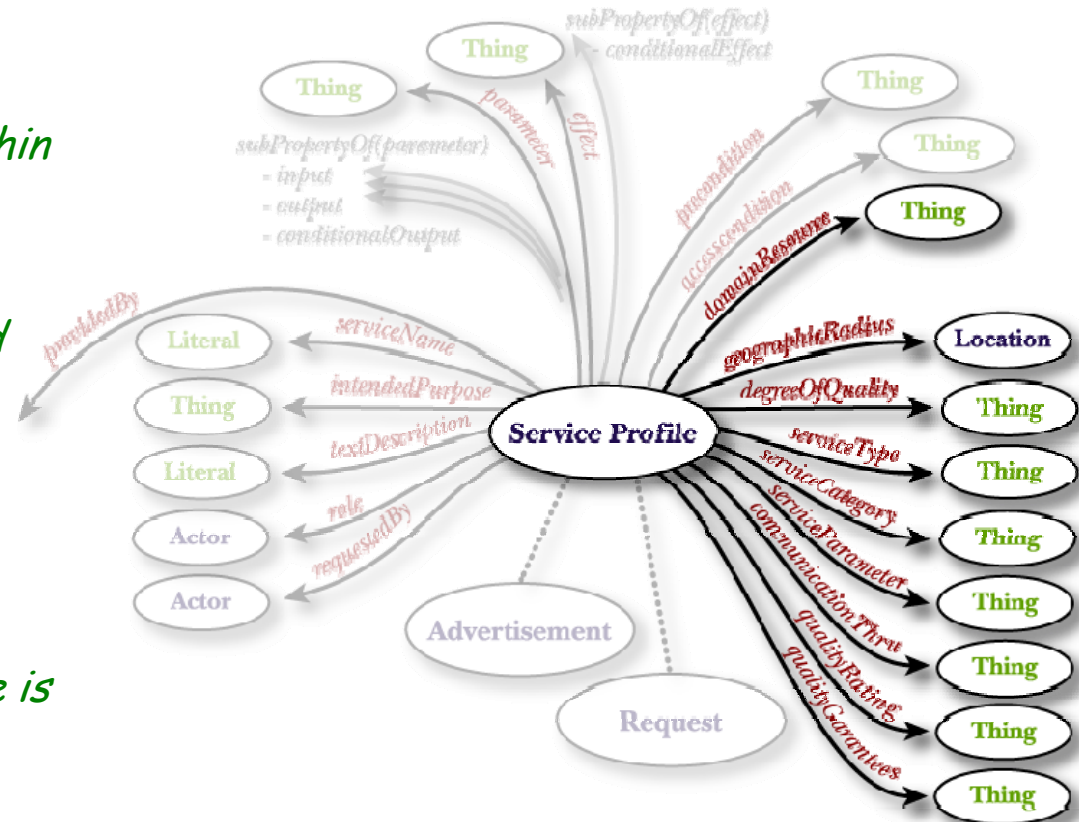
- **Inputs**
  - Set of necessary inputs that the requester should provide to invoke the service
- **(Conditional) Outputs**
  - Results that the requester should expect after interaction with the service provider is completed
- **Preconditions**
  - Set of conditions that should hold prior to service 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



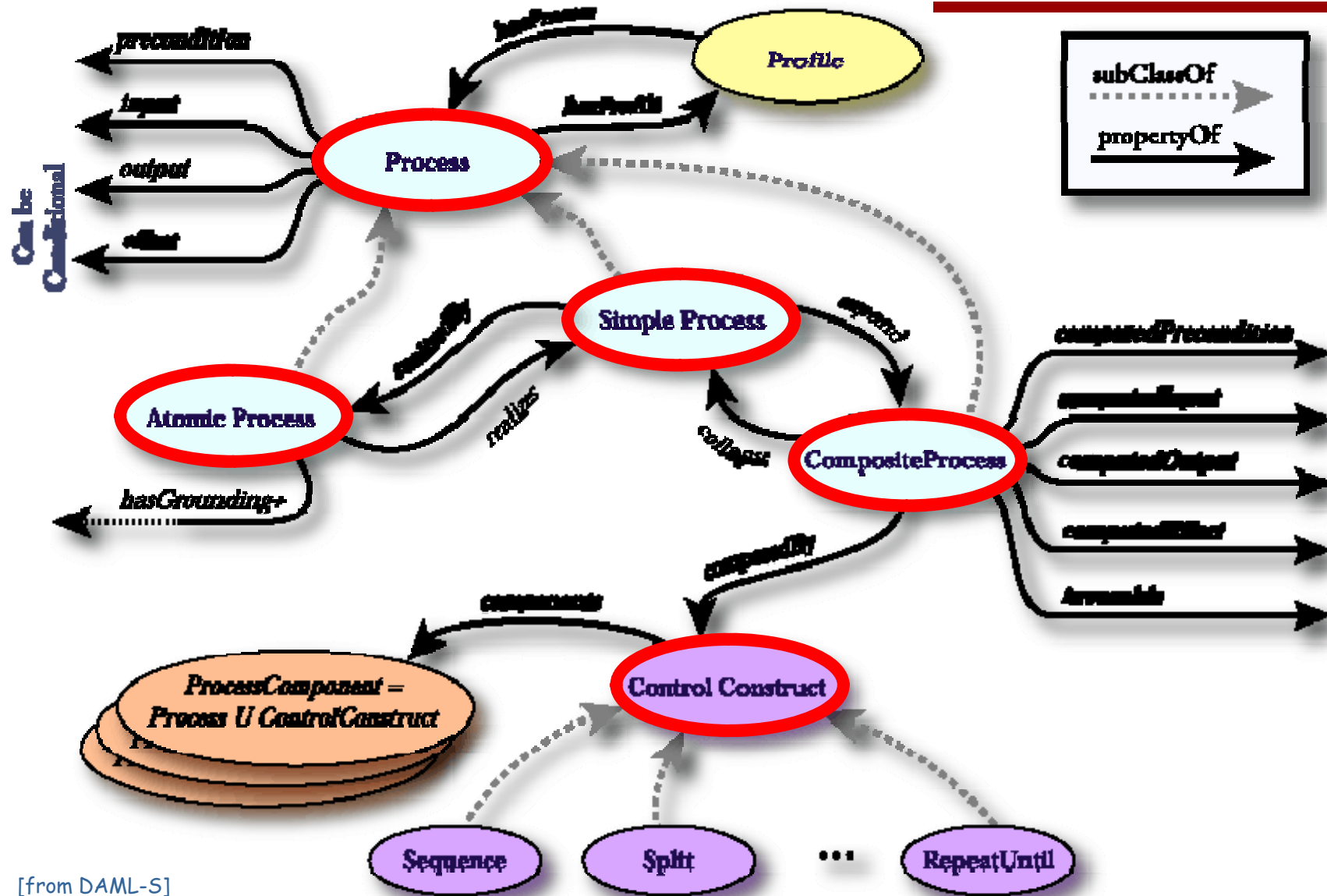
Provide supporting information about the service, including:

- 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 Service Model (How it works)



# *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

# *Enhancements*



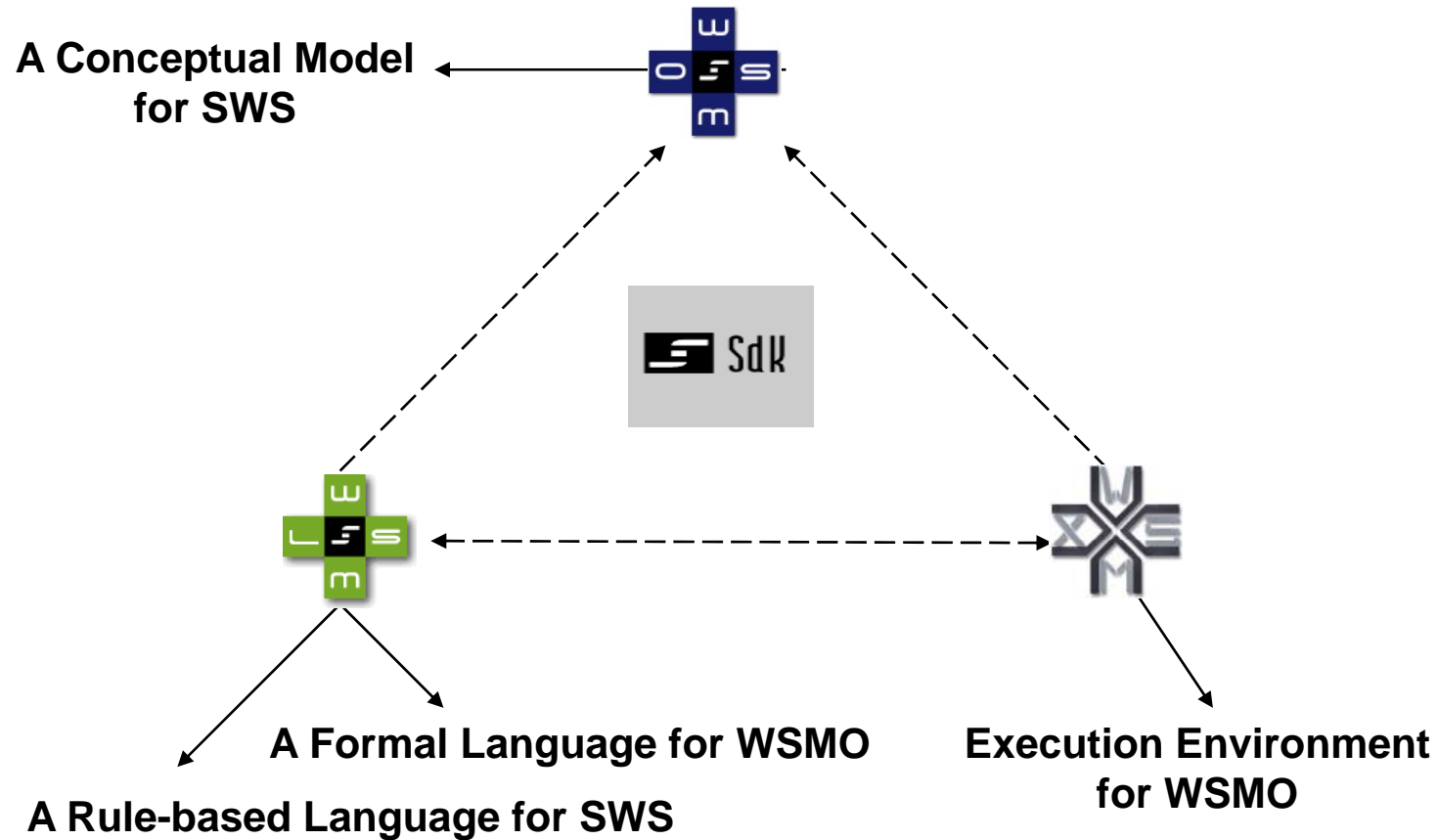
- Recent proposals aim at improving and detailing process modeling and dynamic semantics ...
  - SWSL (Semantic Web Service Language)
- ... work in progress !!
  - <http://www.daml.org/services/swsl/>

- Conceptual model for Semantic Web Services :
  - Ontology of core elements for Semantic Web Services (WSMO)
  - Formal description language (WSML)
  - Execution environment (WSMX)
- ... derived from and based on the Web Service Modeling Framework WSMF
- a SDK-Cluster Working Group  
(joint European research and development initiative)

# WSMO Working Groups



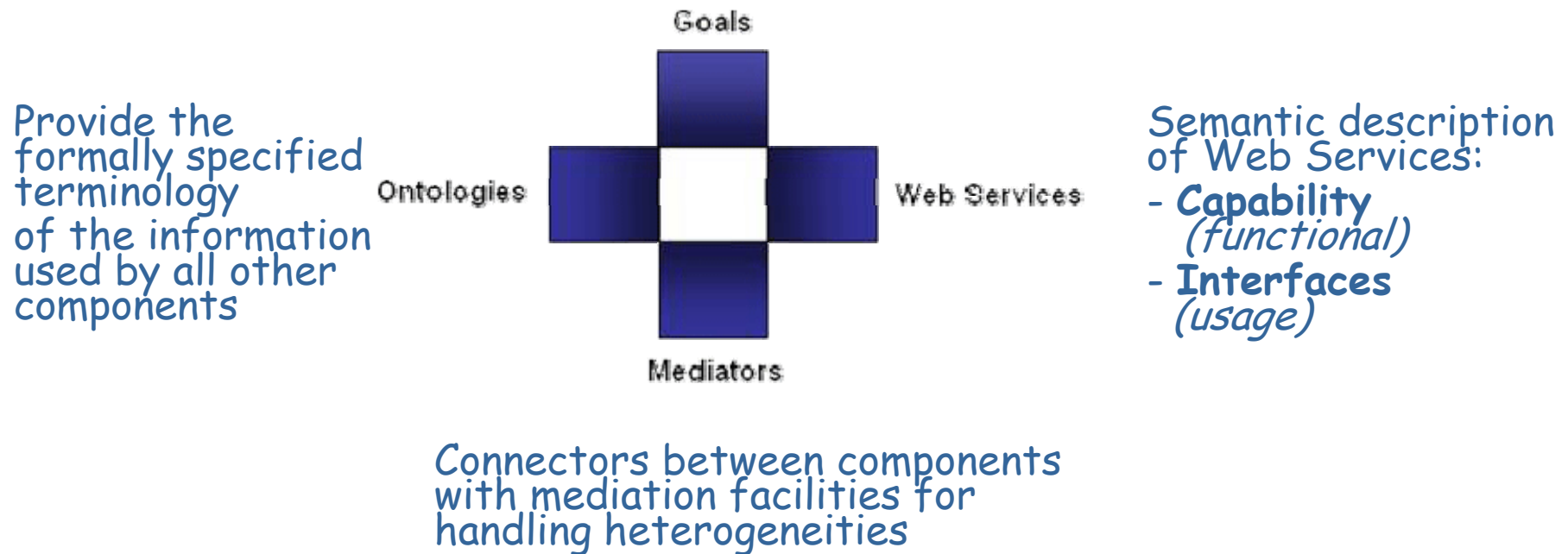
SAPIENZA  
UNIVERSITÀ DI ROMA



# WSMO Top Level Notions



Objectives that a client wants to achieve by using Web Services



# WSMO Web Service Description



- complete item description
- quality aspects
- Web Service Management

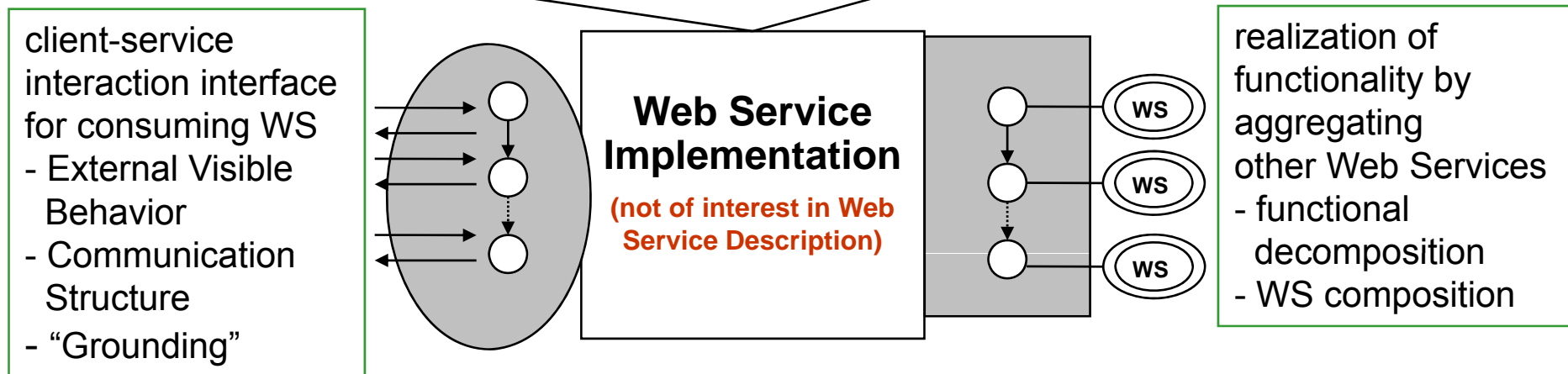
## Non-functional Properties

Dublin Core + QoS +  
version + financial

- Advertising of Web Service
- Support for WS Discovery

## Capability

functional description



Choreography --- Service Interfaces --- Orchestration

# Capability Specification



SAPIENZA  
UNIVERSITÀ DI ROMA

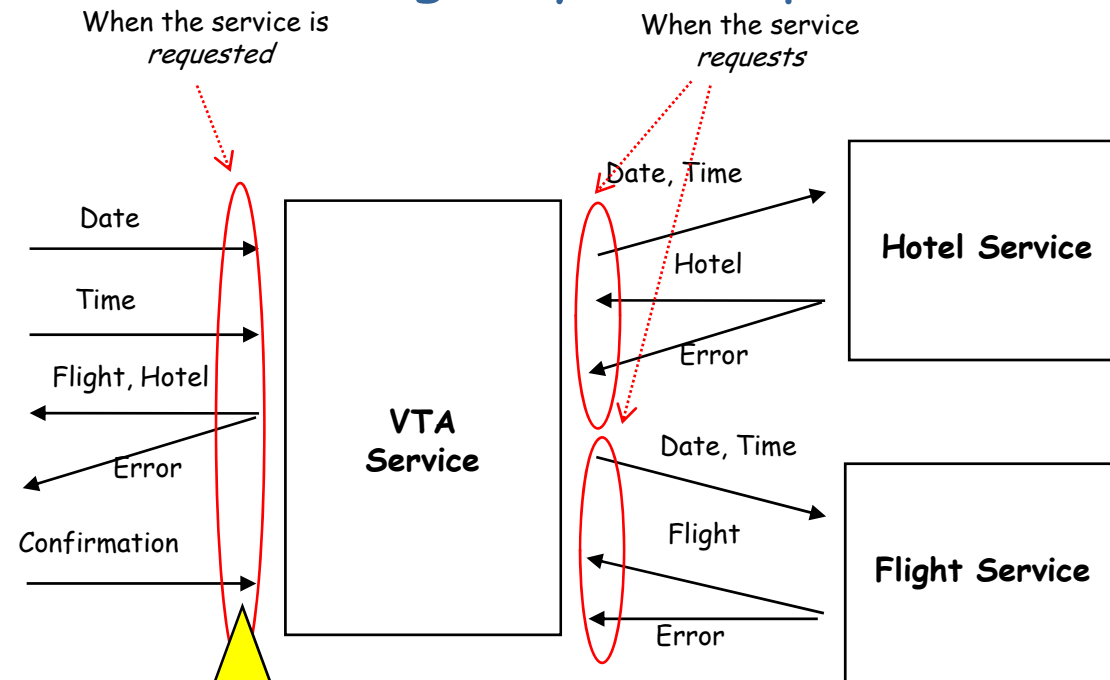
---

- **Non functional properties**
- **Imported Ontologies**
- **Used mediators**
  - *OO Mediator*: importing ontologies with mismatch resolution
  - *WG Mediator*: link to a Goal wherefore service is not usable a priori
- **Pre-conditions**
  - What a web service expects in order to be able to provide its service. They define conditions over the input.
- **Assumptions**
  - Conditions on the state of the world that has to hold before the Web Service can be executed
- **Post-conditions**
  - Describes the result of the Web Service in relation to the input, and conditions on it
- **Effects**
  - Conditions on the state of the world that hold after execution of the Web Service (i.e. changes in the state of the world)

# Choreography & Orchestration



- VTA (Virtual Travel Agency) example:



- **Choreography** = how to interact with the service to consume its functionality
- **Orchestration** = how service functionality is achieved aggregating other Web Services

What previously referred as conversation specification



# *Choreography Aspects*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

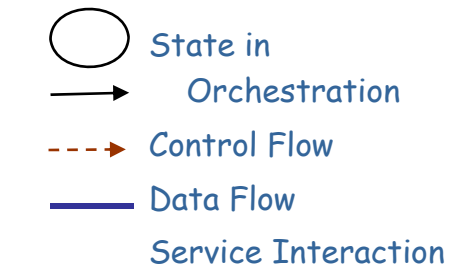
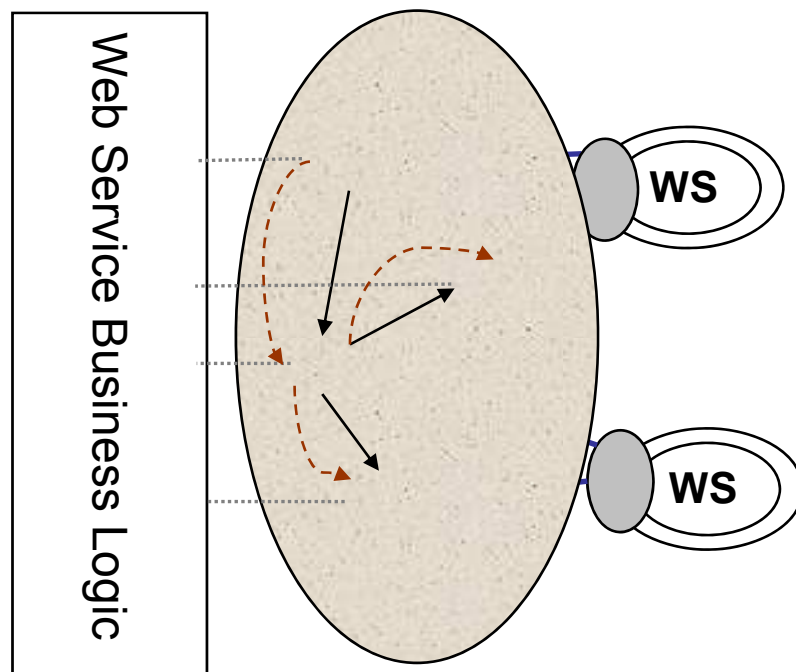
## *Interface for consuming Web Service*

- **External Visible Behavior**
  - those aspects of the workflow of a Web Service where Interaction is required
  - described by workflow constructs: sequence, split, loop, parallel
- **Communication Structure**
  - messages sent and received
  - their order (communicative behavior for service consumption)
- **Grounding**
  - executable communication technology for interaction
  - choreography related errors (e.g. input wrong, message timeout, etc.)
- **Formal Model**
  - reasoning on Web Service interfaces (service interoperability)
  - allow mediation support on Web Service interfaces

# Orchestration Aspects



## Control Structure for aggregation of other Web Services



- decomposition of service functionality
- all service interaction via choreographies

# *WSMO Web Service Interfaces*



- service interfaces are concerned with service consumption and interaction
- Choreography and Orchestration as sub-concepts of Service Interface
- common requirements for service interface description:
  1. represent the dynamics of information interchange during service consumption and interaction
  2. support ontologies as the underlying data model
  3. appropriate communication technology for information interchange
  4. sound formal model / semantics of service interface specifications in order to allow operations on them.

# *Service Interface Description*



- **Ontologies as data model:**
  - all data elements interchanged are ontology instances
  - service interface = evolving ontology
- **Abstract State Machines (ASM) as formal framework:**
  - dynamics representation: high expressiveness
  - core principles: state-based, state definition by formal algebra, guarded transitions for state changes
- **further characteristics:**
  - not restricted to any specific communication technology
  - ontology reasoning for service interoperability determination
  - basis for declarative mediation techniques on service interfaces

# *Service Interface Description Model*



- Vocabulary  $\Omega$ :
  - ontology schema(s) used in service interface description
  - usage for information interchange: in, out, shared, controlled
- States  $w(\Omega)$ :
  - a stable status in the information space
  - defined by attribute values of ontology instances
- Guarded Transition  $GT(w)$ :
  - state transition
  - general structure: *if* (condition) *then* (action)
  - different for Choreography and Orchestration

# Service Interface Example



## Communication Behavior of a Web Service

```
 $\Omega_{in}$  hasValues {  
  concept A [  
    att1 ofType X  
    att2 ofType Y  
  ...]  
}
```

```
 $\Omega_{out}$  hasValues {  
  concept B [  
    att1 ofType W  
    att2 ofType Z  
  ...]  
}
```

### Vocabulary:

- Concept A in  $\Omega_{in}$
- Concept B in  $\Omega_{out}$

State  $\omega_1$

```
a memberOf A [  
  att1 hasValue x  
  att2 hasValue y]
```

received ontology  
instance **a**

Guarded Transition  $GT(\omega_1)$

```
IF (a memberOf A [  
  att1 hasValue x ])  
THEN  
(b memberOf B [  
  att2 hasValue m ])
```

State  $\omega_2$

```
a memberOf A [  
  att1 hasValue x,  
  att2 hasValue y]  
  
b memberOf B [  
  att2 hasValue m]
```

sent ontology  
instance **b**

# *WSMO Future Work*



SAPIENZA  
UNIVERSITÀ DI ROMA

---

- Orchestrazione does not exist in the last version of the WSMO documents
- ASM Graphical representation (possibly through UML Activity Diagrams)

More on Semantic Web Services: ESWC 2005 Tutorial -  
[http://kmi.open.ac.uk/projects/dip/  
resources/eswc2005/SWStutorial-eswc05.ppt](http://kmi.open.ac.uk/projects/dip/resources/eswc2005/SWStutorial-eswc05.ppt)

# ebXML



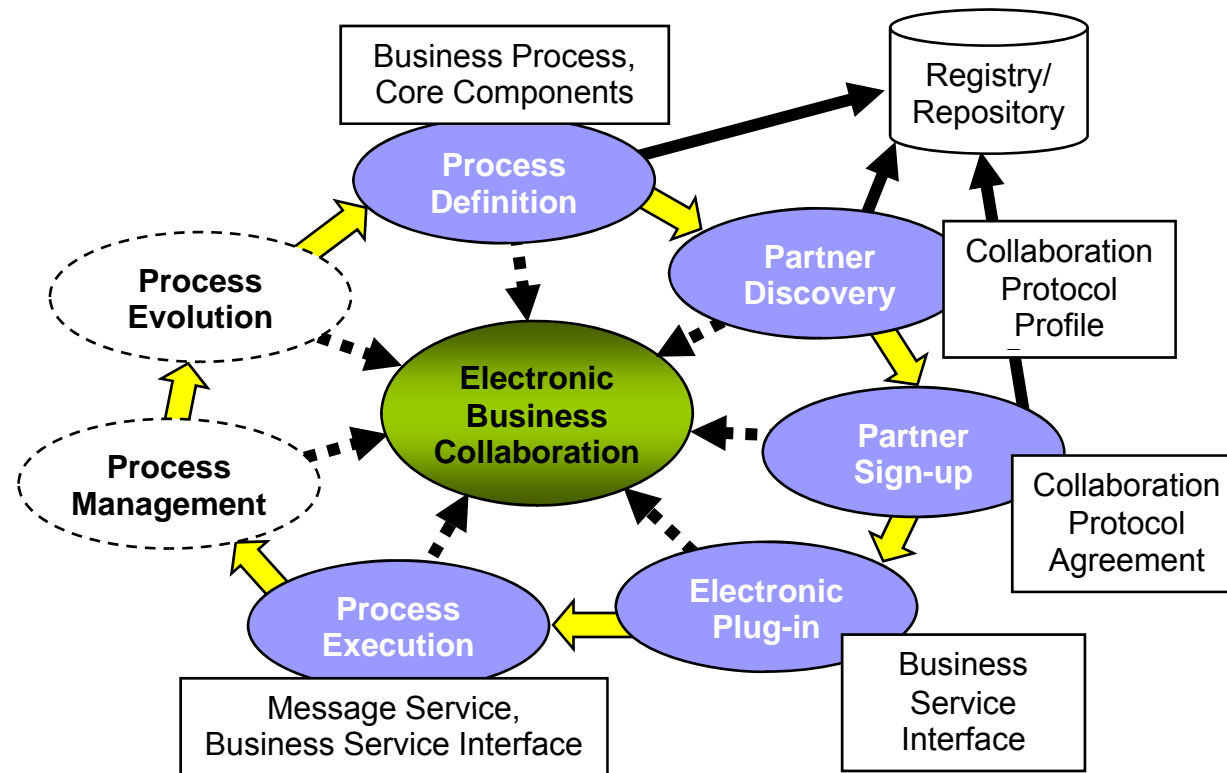
- ebXML is more a standardized “conceptual framework”, a “reference model”, than a real stack of standard technologies
  - *Stable version in 2001/2002*
    - Technical Architecture Specification (v1.04)
    - Business Process Specification Schema (v1.01)
    - Registry Information Model (v2.0)
    - 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

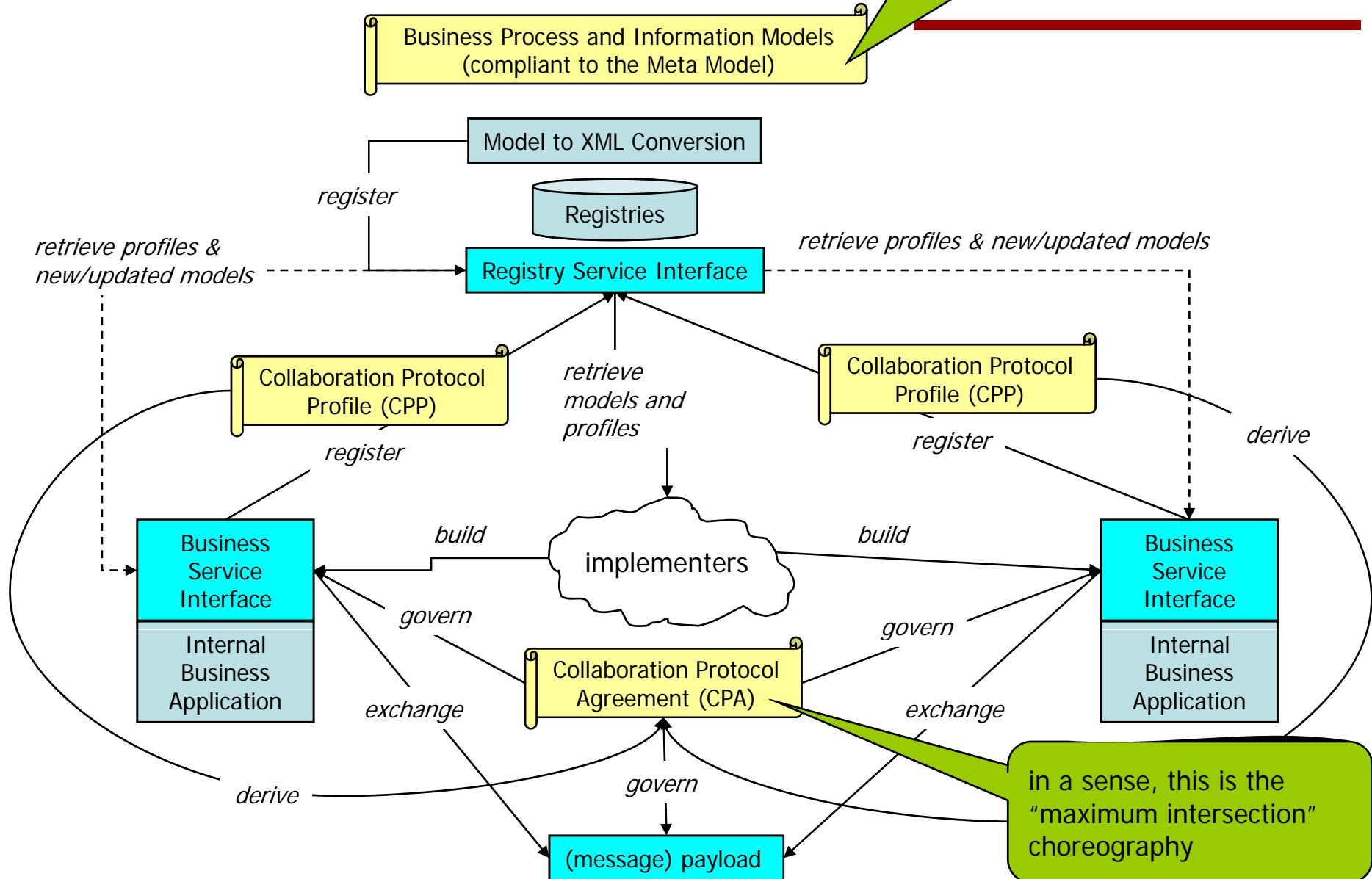


- To define an open & public infrastructure, based on XML, for distributed electronic commerce
  - Special attention to SMEs and developing countries



# 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

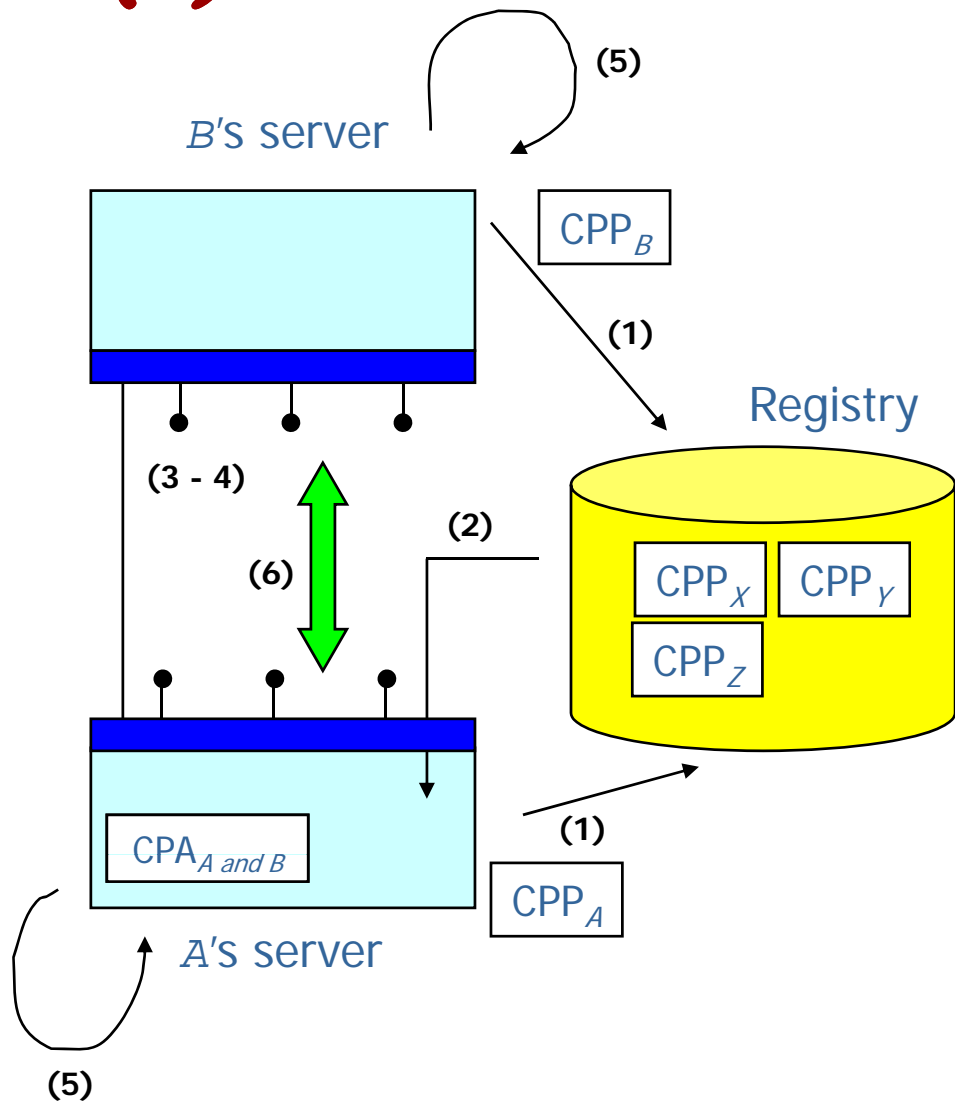
# *ebXML: BPSS, CPP e CPA*

## *(2)*



- *A* wants to make electronic business with *B*; *A* is the acquirer and *B* the vendor; the process underlying the business is already defined in a BPS
- *A* discovers the *B*'s CPP in a registry
- A CPA is created, as the intersection of *A*'s CPP and *B*'s CPP
- On the basis of the CPA, the *A*'s and *B*'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  $A$  discovers  $B$  in the registry and download CPP <sub>$B$</sub>  on its system
3. Partner  $A$  creates CPA <sub>$A$  and  $B$</sub>  and sends it to  $B$
4. After a negotiation (both manual or automatic), both  $A$  and  $B$  register identical copies of the agreed upon CPA <sub>$A$  and  $B$</sub>  in their systems
5. Both  $A$  and  $B$  configure their systems for runtime on the basis of CPA <sub>$A$  and  $B$</sub>
6. Finally  $A$  and  $B$  engage their e-Commerce process

# 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.Tosic, 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 etal 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 etal 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-TE04] - D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella: Post-proc. of VLDB-TE04, 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.ebpm.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/>