Software architectures: how components can go politely social

Paola Inverardi

Software Engineering and Architecture Group
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
University of L'Aquila, Italy

Software Architectures

- SA serve many purposes - see [TMD2009]
  - My favorite view
    - glue/connectivity *nature* that allows subsystems/ components to interact, correctly
  - Define the system structure in terms of components/subsystems, their interactions in terms of functional and non functional behavior, either local and global

[TMD2009] Richard N. Taylor, Nenad Medvidovic, Eric Dashofy, Software Architecture: Foundations, Theory, and Practice

What is Social?

- Merriam-Webster **social**: *relating to or involving activities in which people spend time talking to each other or doing enjoyable things*…

- **Meet and enjoy yourself doing things together → no bad things happen, be polite!**

SA behavioral: A Historical Perspective

1990	• The birth of Software Architecture
1995	• Software Architecture analysis
1997	• Architectural Connectors
2000	• Checking Assumptions & SA Synthesis
2000	• Service Oriented Architecture
2010	• NS, Interoperability, mediators
2020	• Integration means for Services, Apps, Cloud

Software Architecture & Politeness

- SA defines structure/components and interactions
- The global behavior of the SA can be analyzed and checked to assess politeness, e.g., absence of deadlock, desired behavior, i.e., *after a receive always an ack*
- Early 90ties Kramer& Magee, Inverardi et al., Luckam et al, Allen&Garlan
  - Formal descriptions automata, process calculi, Cham, po sets (influence from concurrency theory)
  - Complexity and scalability

System: vehicles and crosspoint

Structuring interactions: protocols and connectors

- SA defines structure/components and interactions
  - Interactions are the *observable* actions at the interface level
  - Interactions are performed by following *protocols*, i.e., given ordering in the way interface operations need to be executed
  - Connectors are architectural elements that define how components' protocols match together

From components behaviors to protocols

- Easy the behavioral proof by introducing “sort of behavioral types”
  - Allen & Garlan roles and ports
  - Inverardi & Wolf & Yankelevich assumption and behaviors
  - Yellin & Strom protocol conversions
  - In PL Behavioral types: session types, behavioral contracts, …

Connectors to improve global analysis

- Connectors can be exploited to separate concerns, decompose system analysis → localize and gain efficiency (?)

- Wright:

  connector C-S-connector =

  role Client = (request!x → result?y → Client) ΓI §

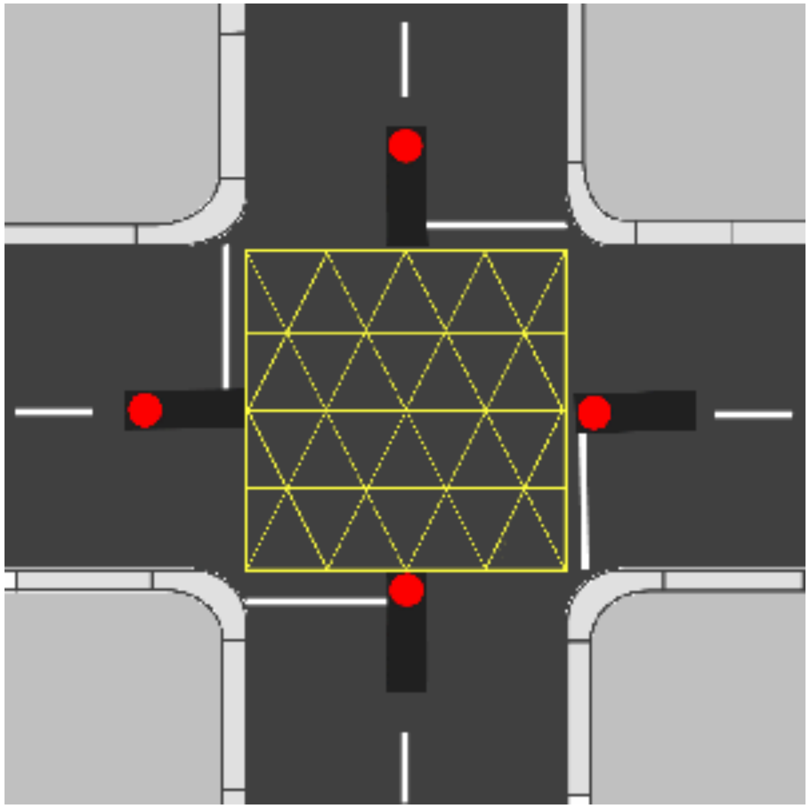
  role Server = (invoke?x → return!y → Server) [] §

  glue = (Client.request?x → Service.invoke!x → Service.return?y → Client.result!y → **glue**)

  [] §

- Analysis reduces to check connectors' behavior and to check compatibility between ports and roles locally, *no efficiency gain*

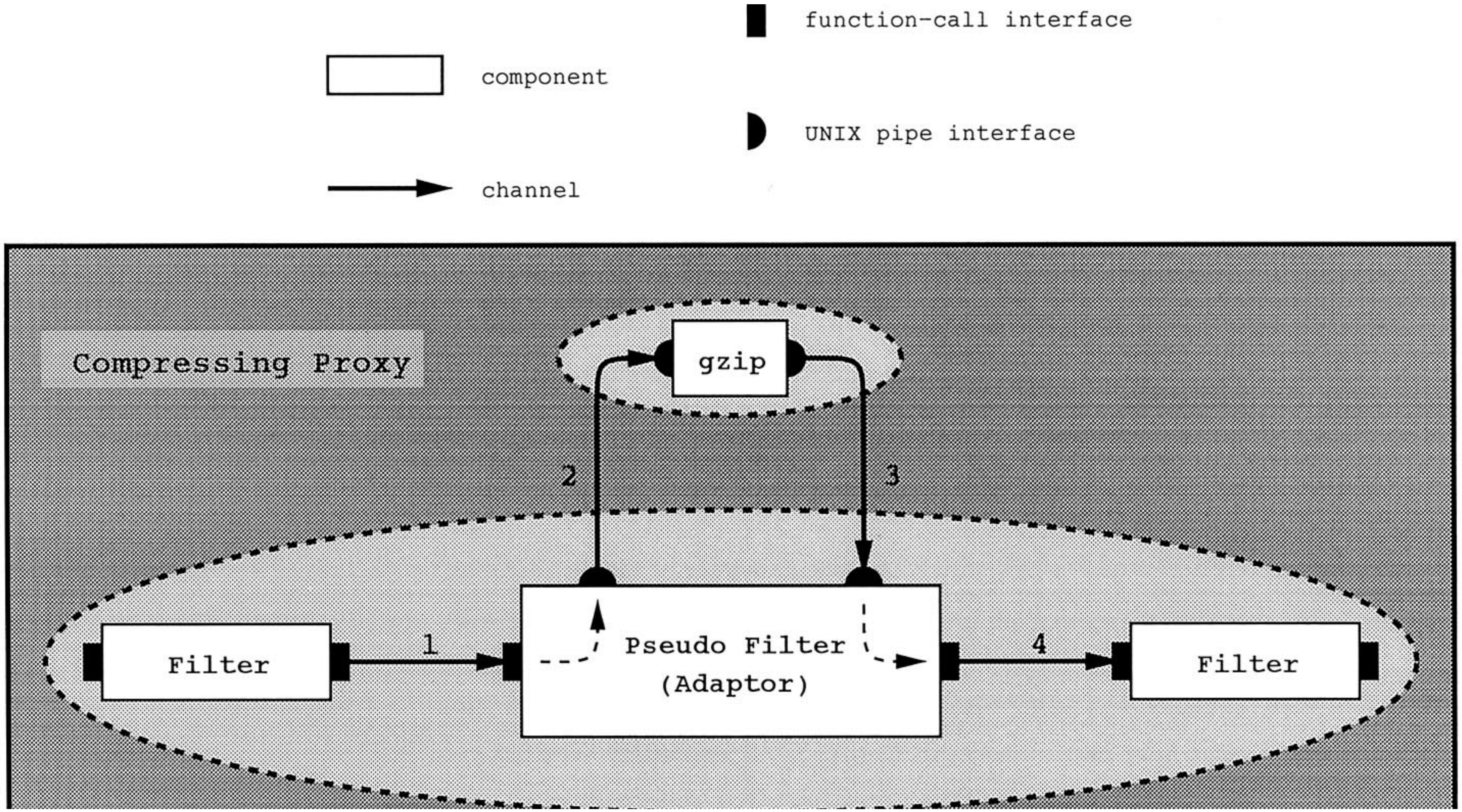
Crosspoint with traffic light connector



Roles and assumptions

- Roles represent explicit assumptions on the environment, i.e., the connector and/or components
- What if we start from components and SA only?
- Inverardi, Wolf, Yankelevich propose a method to generate automatically assumptions out of components and SA and check (efficiently) for compatibility, i.e., deadlock freedom
- The SA component is a sort of connector plus+ (more in the style of Darwin)

Compressing Proxy example



Modeling Gzip

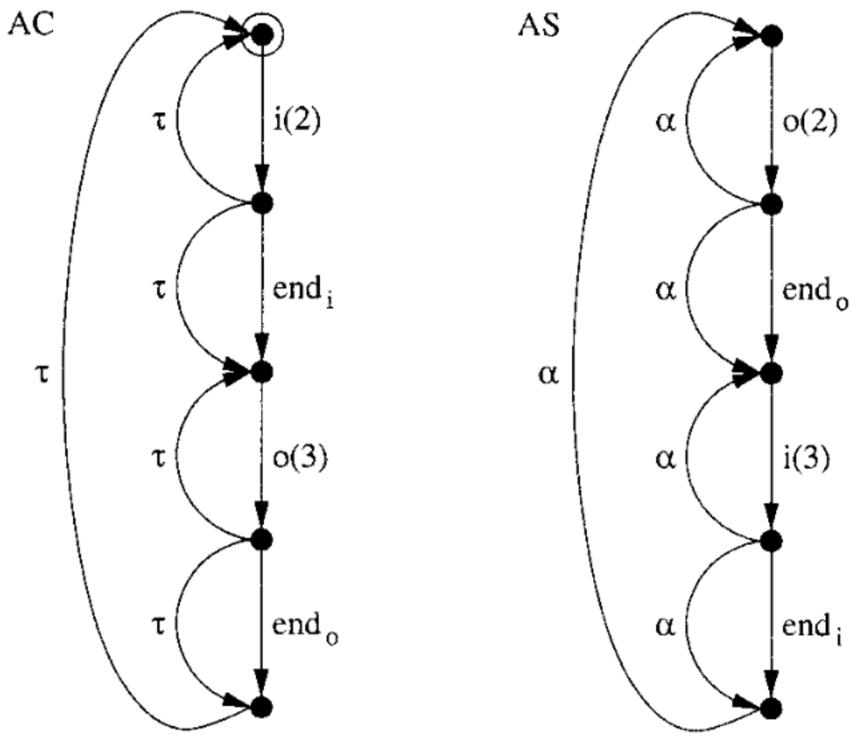


Fig. 3. AC and AS graphs for **gzip.**

Modeling the Adaptor

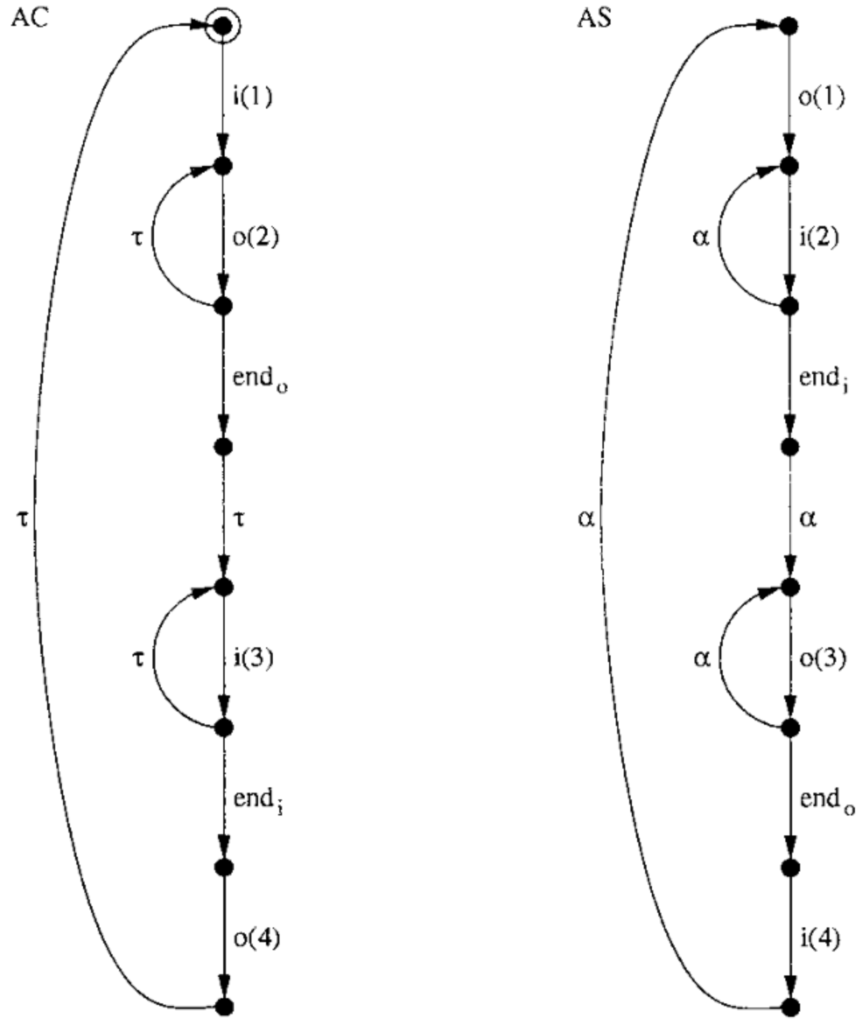


Fig. 4. AC and AS graphs for the adaptor.

Mismatch signaling deadlock

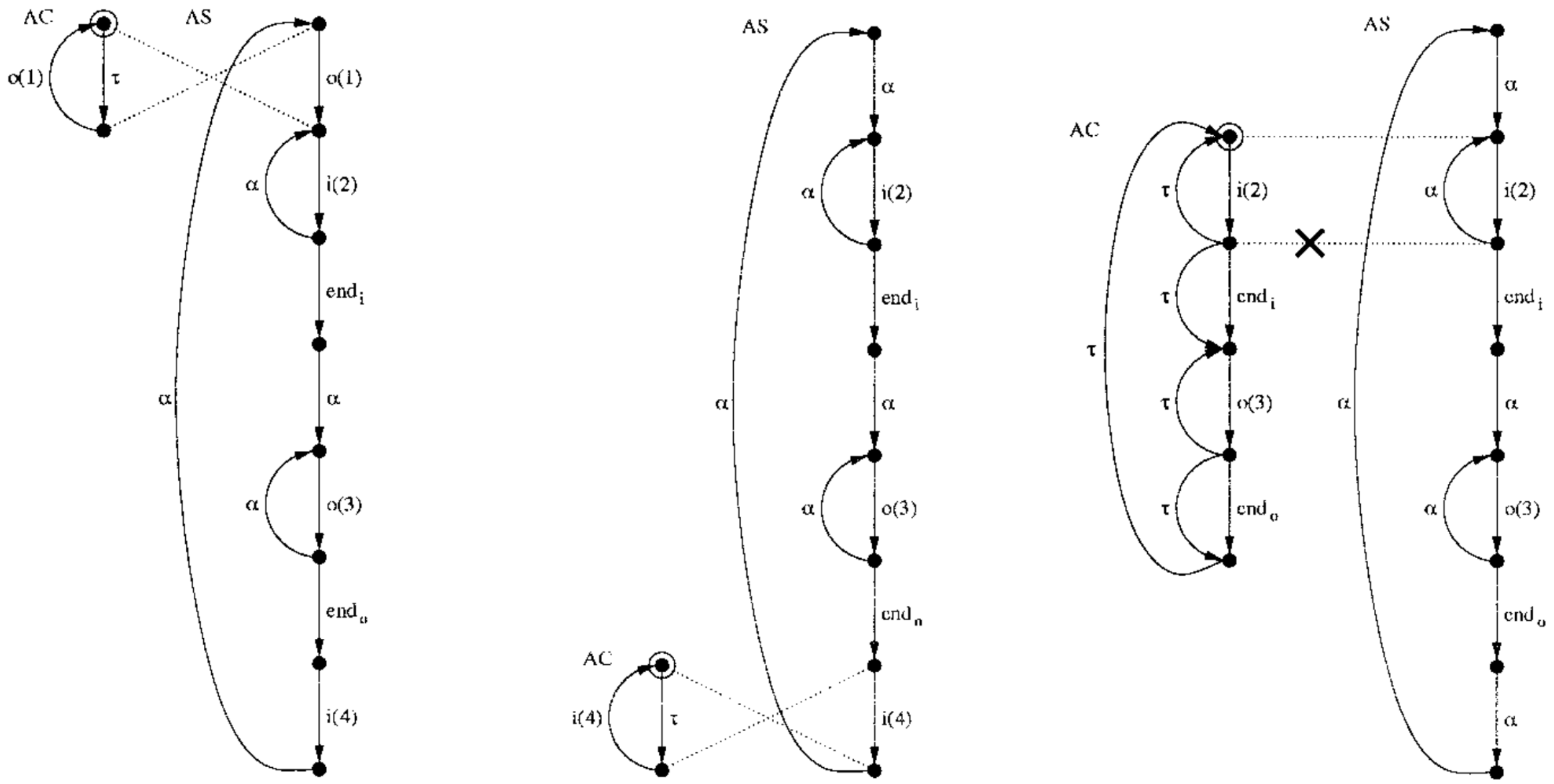


Fig. 5. Mismatch in actual and assumed behavior leading to deadlock.

Changing the Adaptor

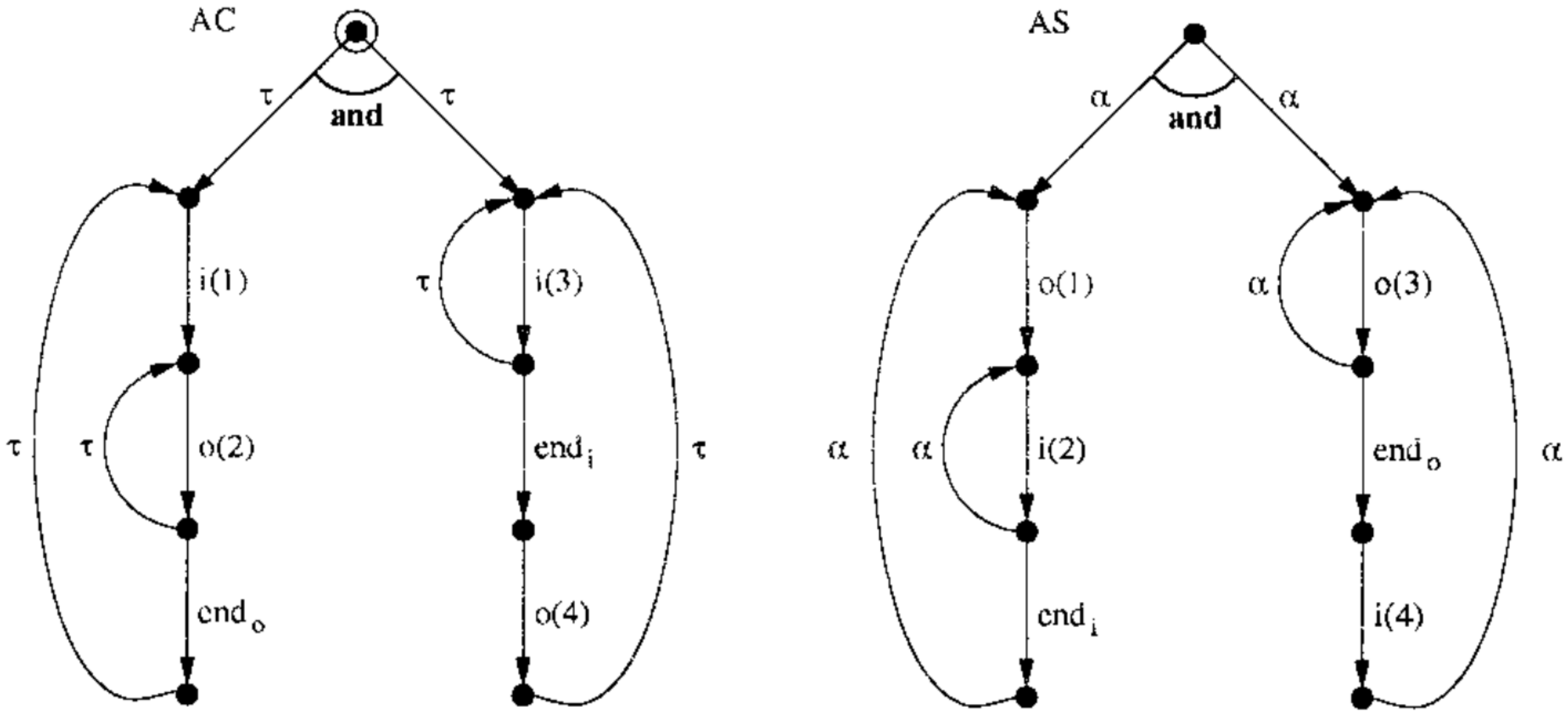


Fig. 6. AC and AS graphs for the modified adaptor.

Successful match

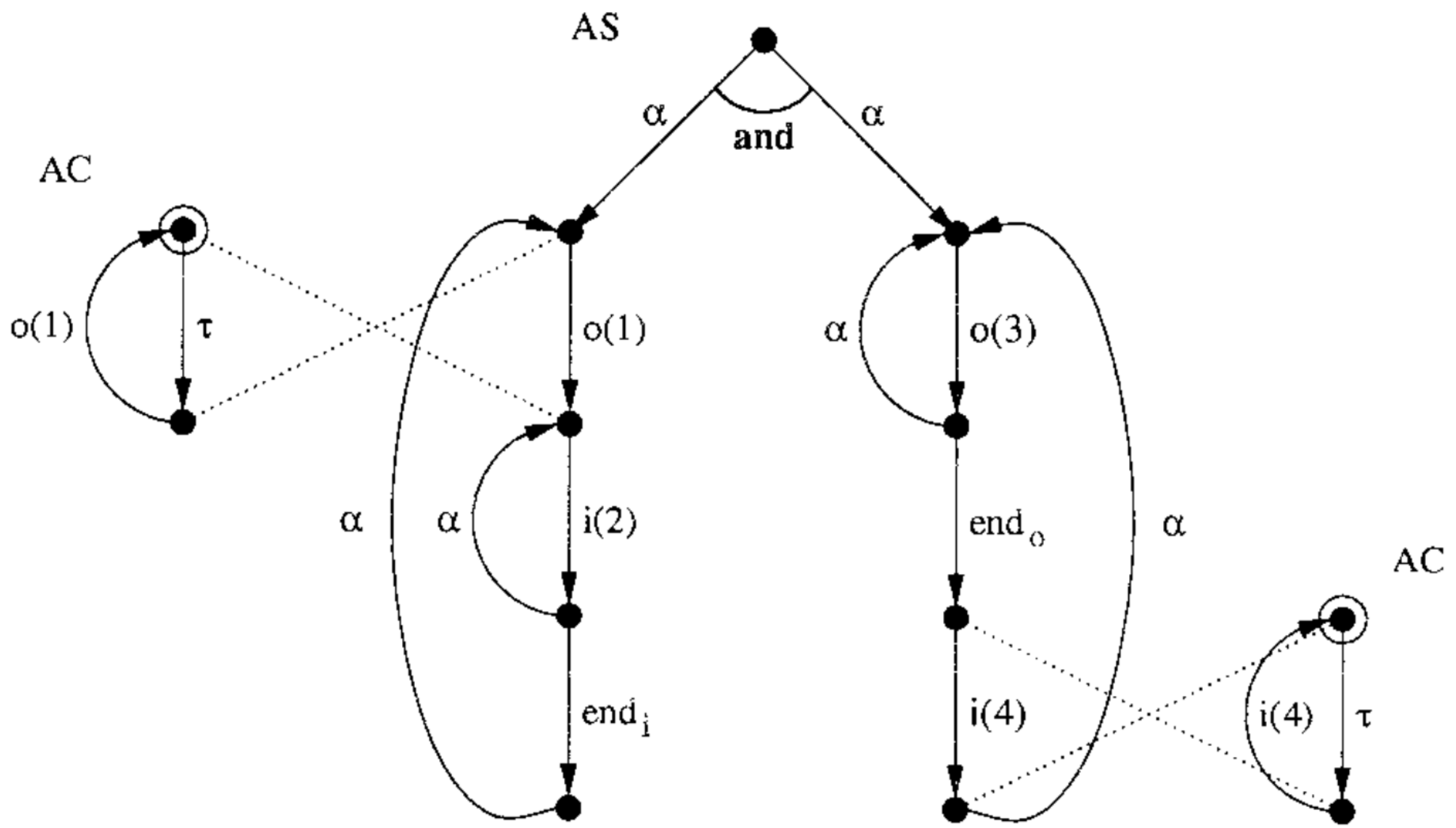


Fig. 7. Successful match of filter AC graphs against adaptor AS graph.

Successful match

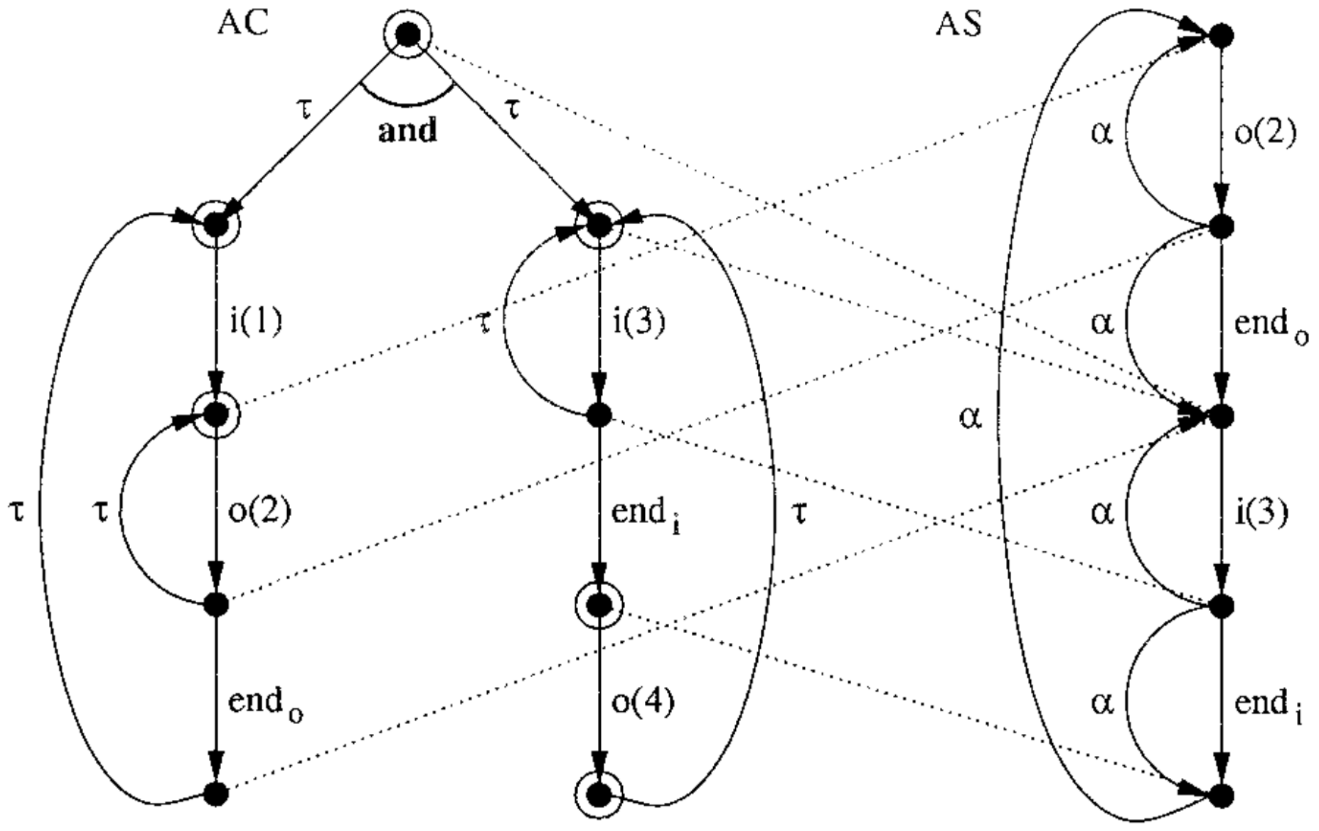


Fig. 8. Successful match of adaptor AC graph against **gzip** AS graph.

Successful match

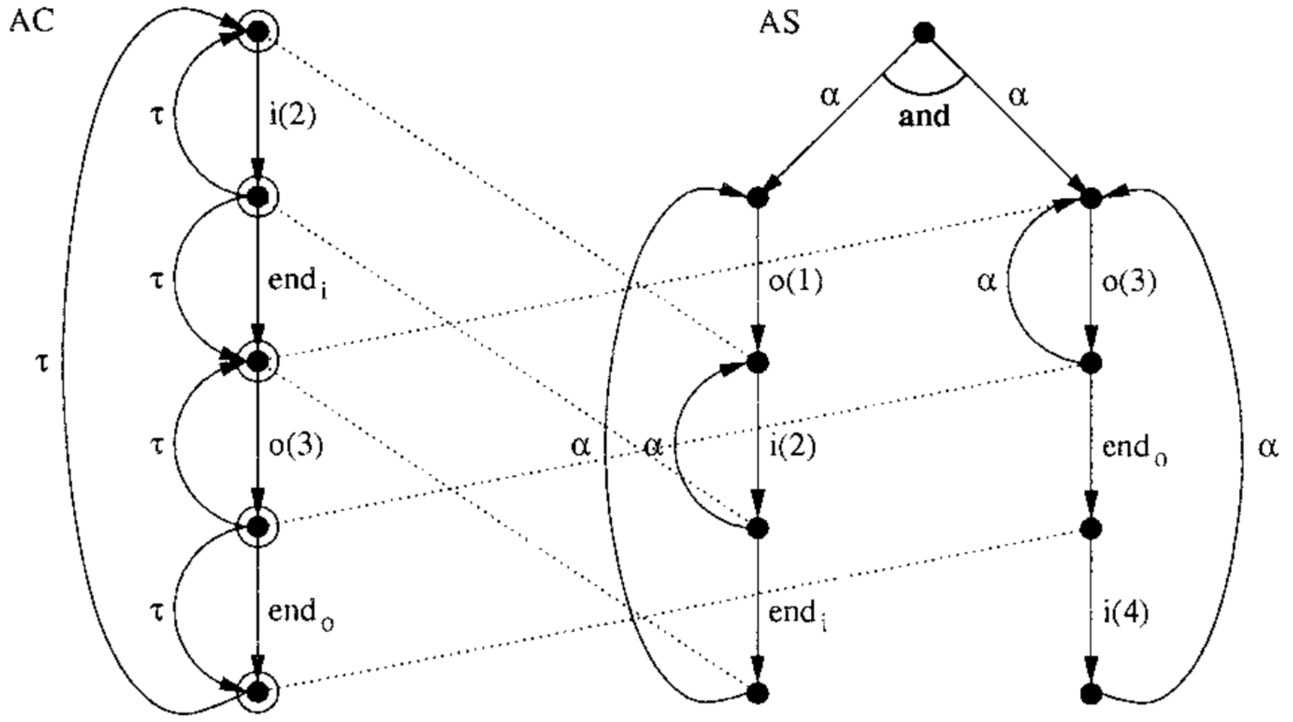


Fig. 9. Successful match of **gzip** AC graph against adaptor AS graph.

Successful match

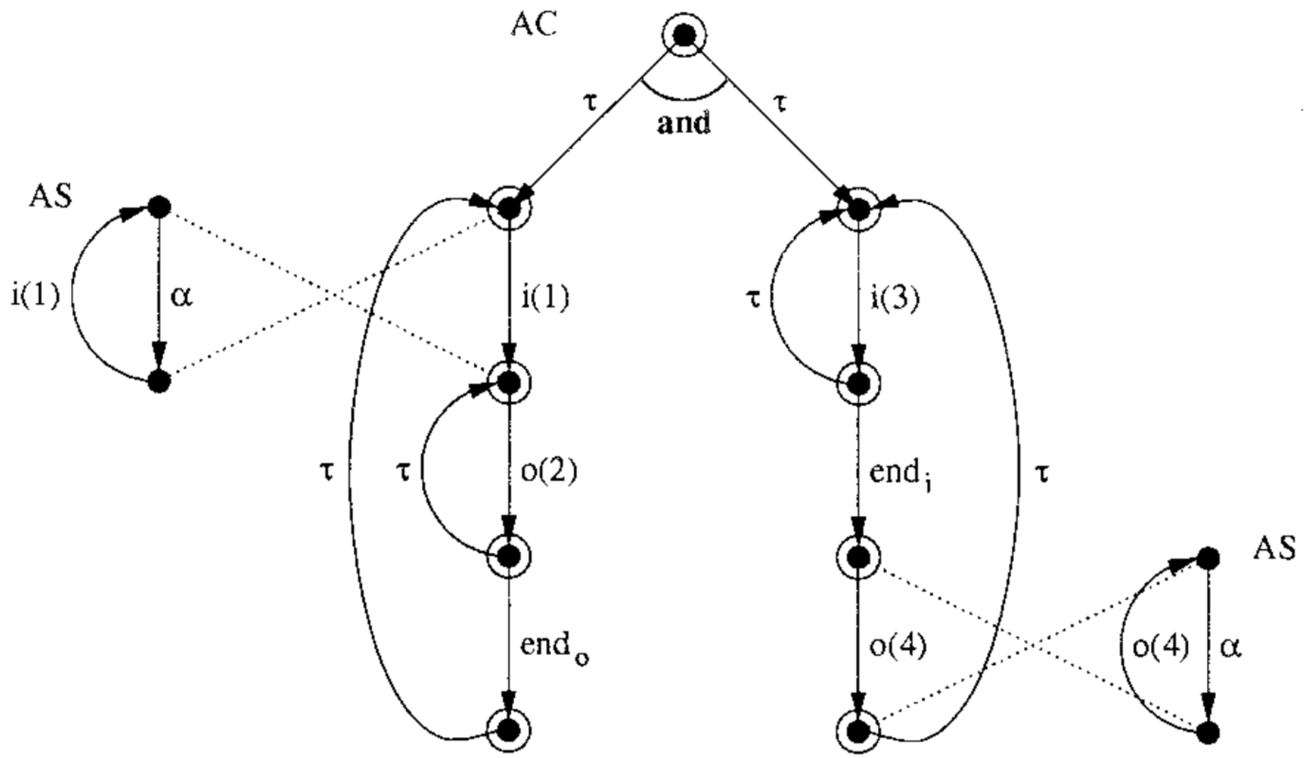
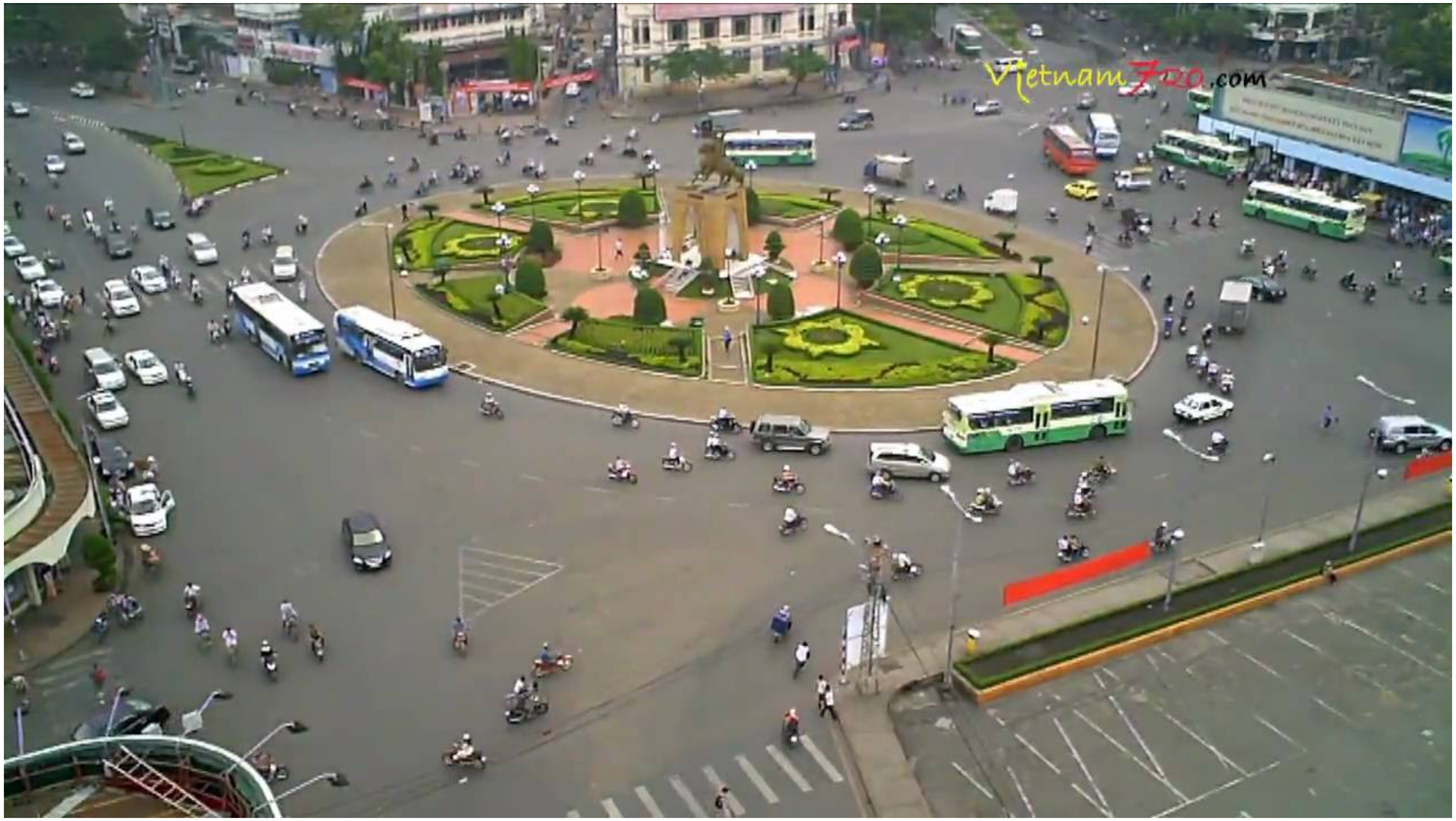


Fig. 10. Successful match of adaptor AC graph against filter AS graphs.

A roundabout



A step further: incomplete knowledge

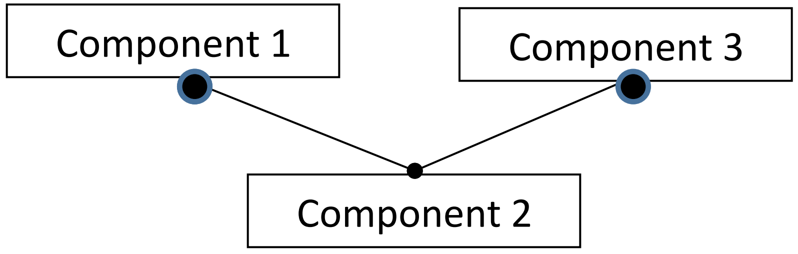
- What if we have only components and no SA/ glue/connector?
- Can we guarantee politeness? Yes if we filter or mask the un-polite behavior (through connectors, mediators, adapters etc.)
- How can we generate such filters? *Synthesis*
- Politeness enforcing can be synthesized more efficiently once we know the SA style we rely on

Putting things together: the Synthesis problem

- Components with protocol behavior
- A desired integration SA style/pattern
- Synthesize the glue so that the components correctly integrate, if possible

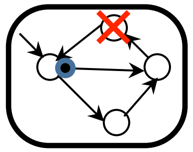
An example: Coordinator concept

communicating black-box components

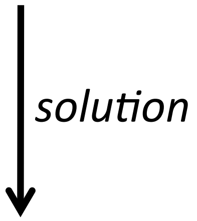


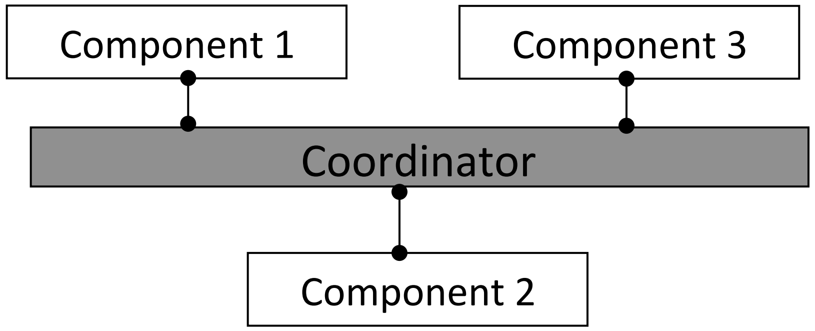
their interaction may deadlock…

…or violate a specified desired behavior



desired behavior specification





the coordinator is an additional component/ connector synthesized so as to intercept all component interactions in order to prevent deadlocks and those interactions that violate the specified desired behavior

Automatic Synthesis of Centralized FFC – Modeling –

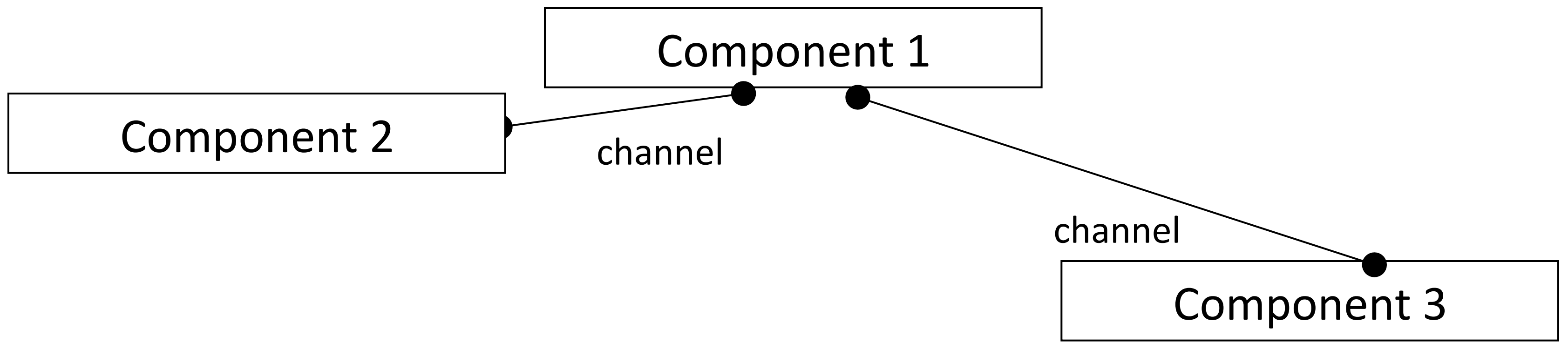
- Component behaviour modeled using finite state machines
- Integration through parallel composition of component models
- In the context of CCS:
  - Components: sequence and choice operators

    E.g. Server = call.service.return.Server
  - Integration: parallel composition and restriction operators

    E.g. $(C_1 \mid C_2 \mid \ldots \mid C_n) \setminus L$
- Deadlock: $(C_1 \mid C_2 \mid \ldots \mid C_n) \setminus L$ can reach a state where no actions are possible

Automatic Synthesis of Centralized FFC – Modeling–

- **C**oordinator **F**ree **A**rchitecture (CFA)

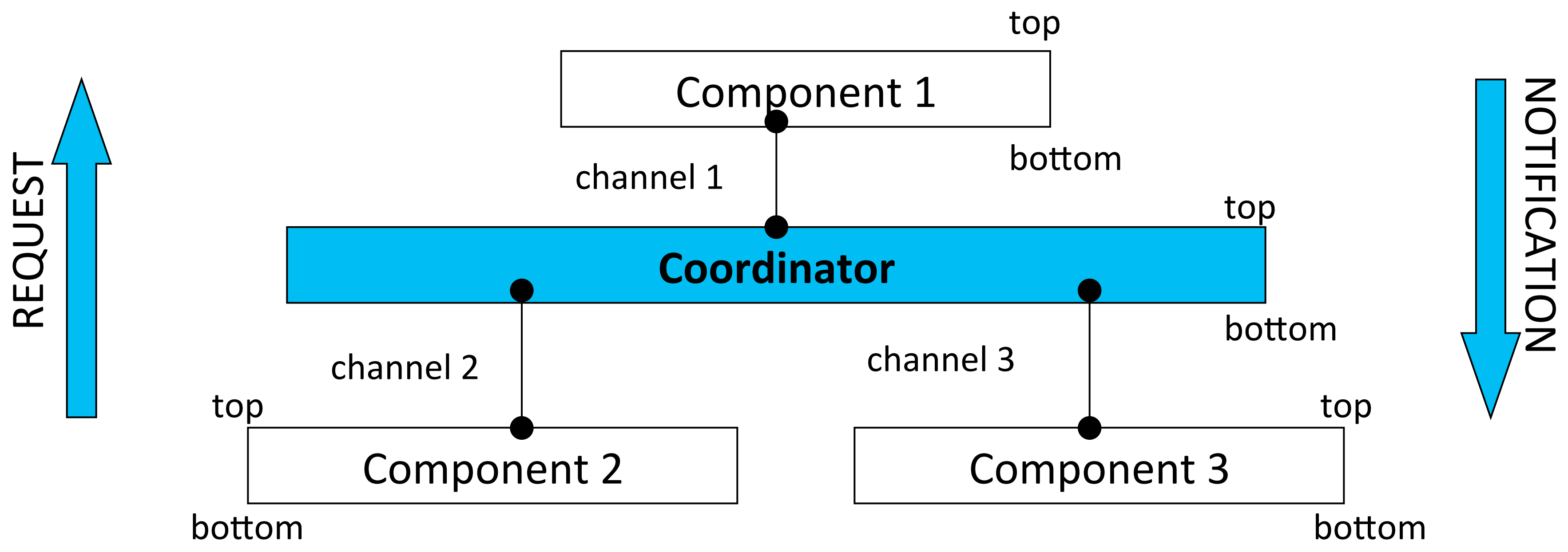


a Coordinator Free Architecture (CFA) is a set of components directly connected in a synchronous way

in CCS : $(C_1 \mid C_2 \mid \dots \mid C_n) \setminus \bigcup_{i=1..n} Act_i$

Automatic Synthesis of Centralized FFC – Modeling –

- **C**oordinator **B**ased **A**rchitecture (CBA)



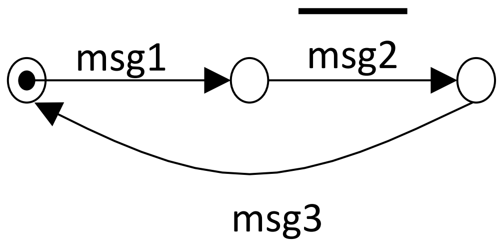
in CCS:

$(C_1[f_1] \mid C_2\,[f_2] \mid \ldots \mid C_n[f_n] \mid K) \setminus U_{i=1..n}\, Act_i[f_i]$

K is the synthesized connector, f_i suitable relabeling function

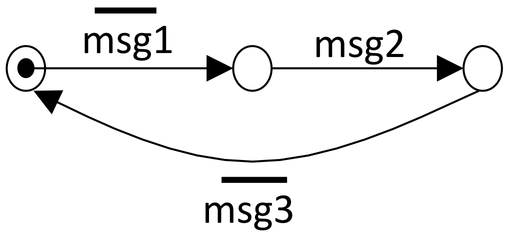
Automatic Synthesis of Centralized FFC – Component Local Views –

AC-Graph: the component protocol



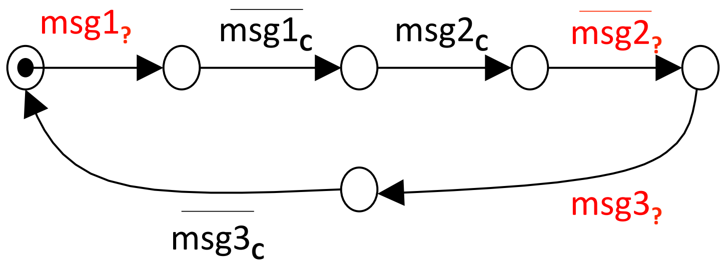
Knowing the composition mechanism and the property

AS-Graph: assumptions on the environment



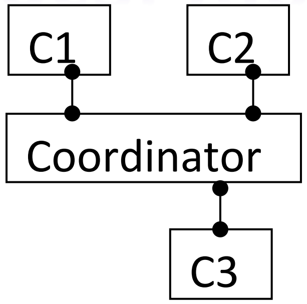
Knowing the characteristics of the environment i.e. the coordinator

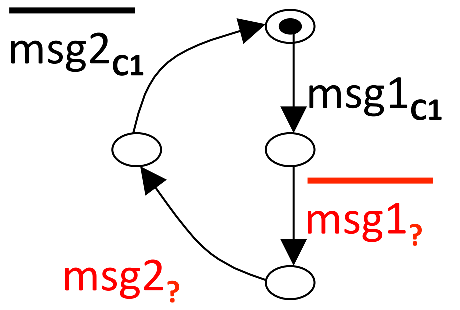
EX-Graph: assumptions on the coordinator

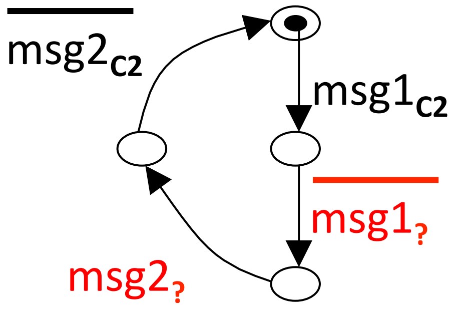


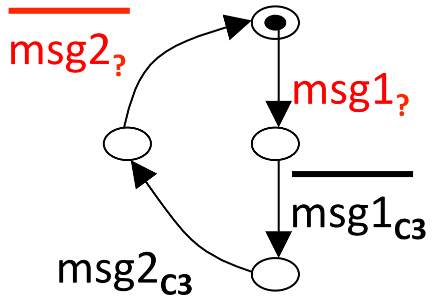
Automatic Synthesis of Centralized FFC – Component Local Views Unification –

based on a usual first-order unification algorithm

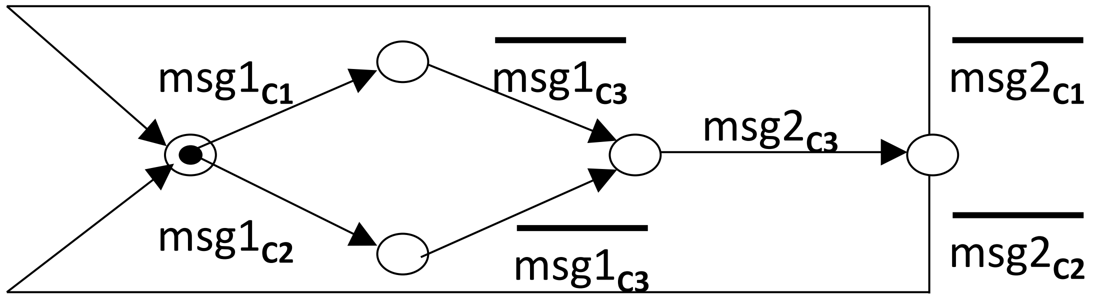




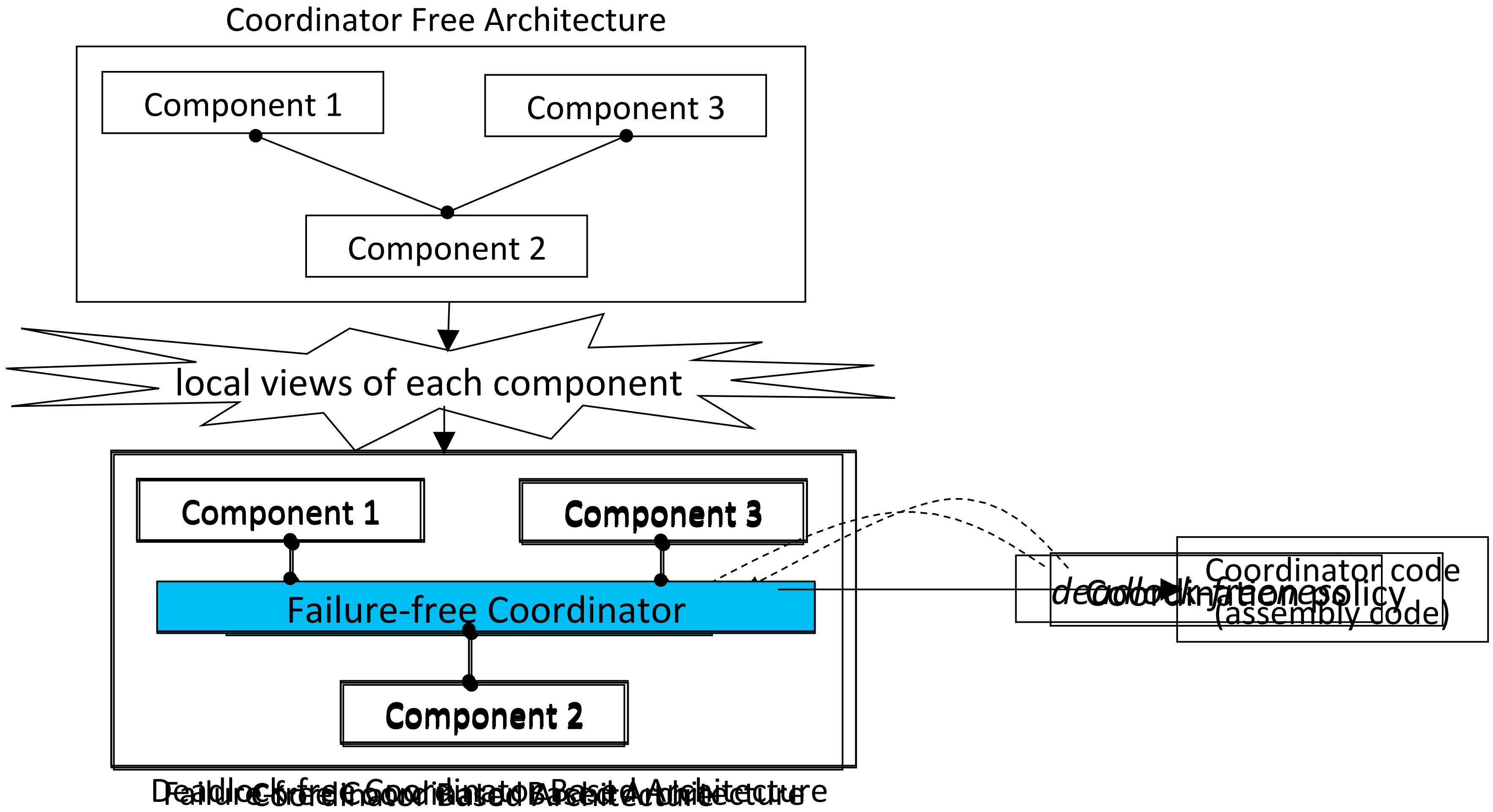




coordinator graph obtained by the unification of the EX-Graphs

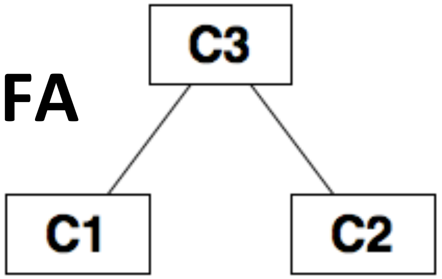


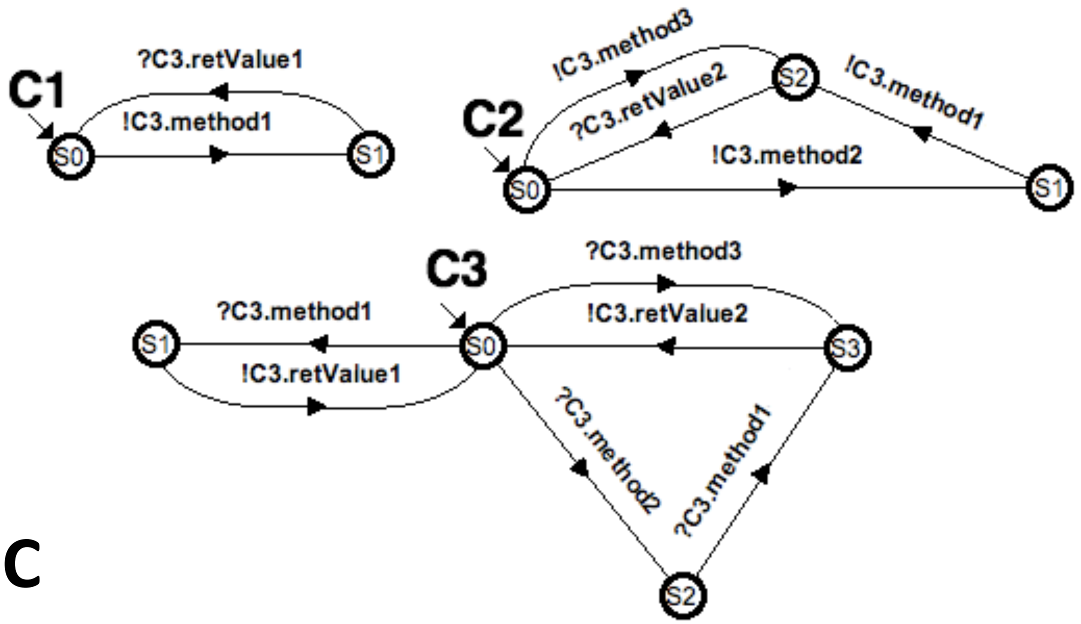
Automatic Synthesis of Centralized FFC – 3-step method –


Coordinator Free Architecture
Component 1
Component 3
Component 2
local views of each component
Component 1
Component 3
Failure-free Coordinator
Component 2
deadlock-freeness
Coordination policy
Coordinator code (assembly code)
Deadlock-free Coordinator Based Architecture
Failure-free Coordinator Based Architecture

Automatic Synthesis of **Centralized** FFC
– *running example: inputs* –

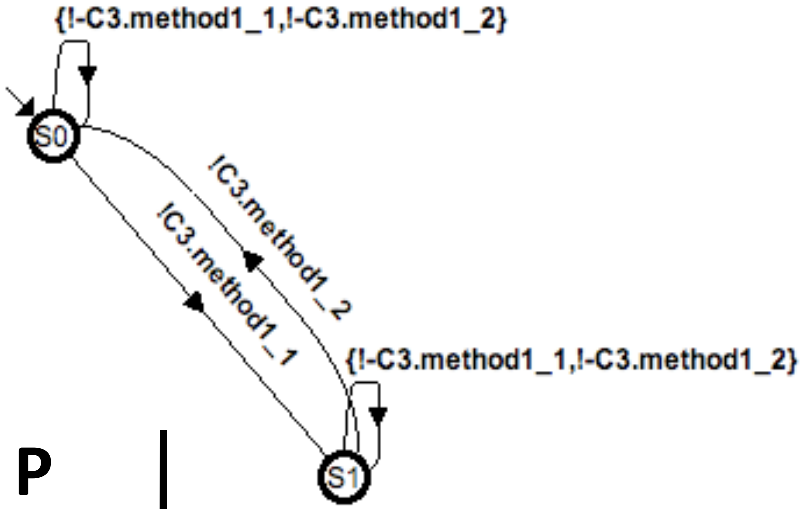
CFA





C

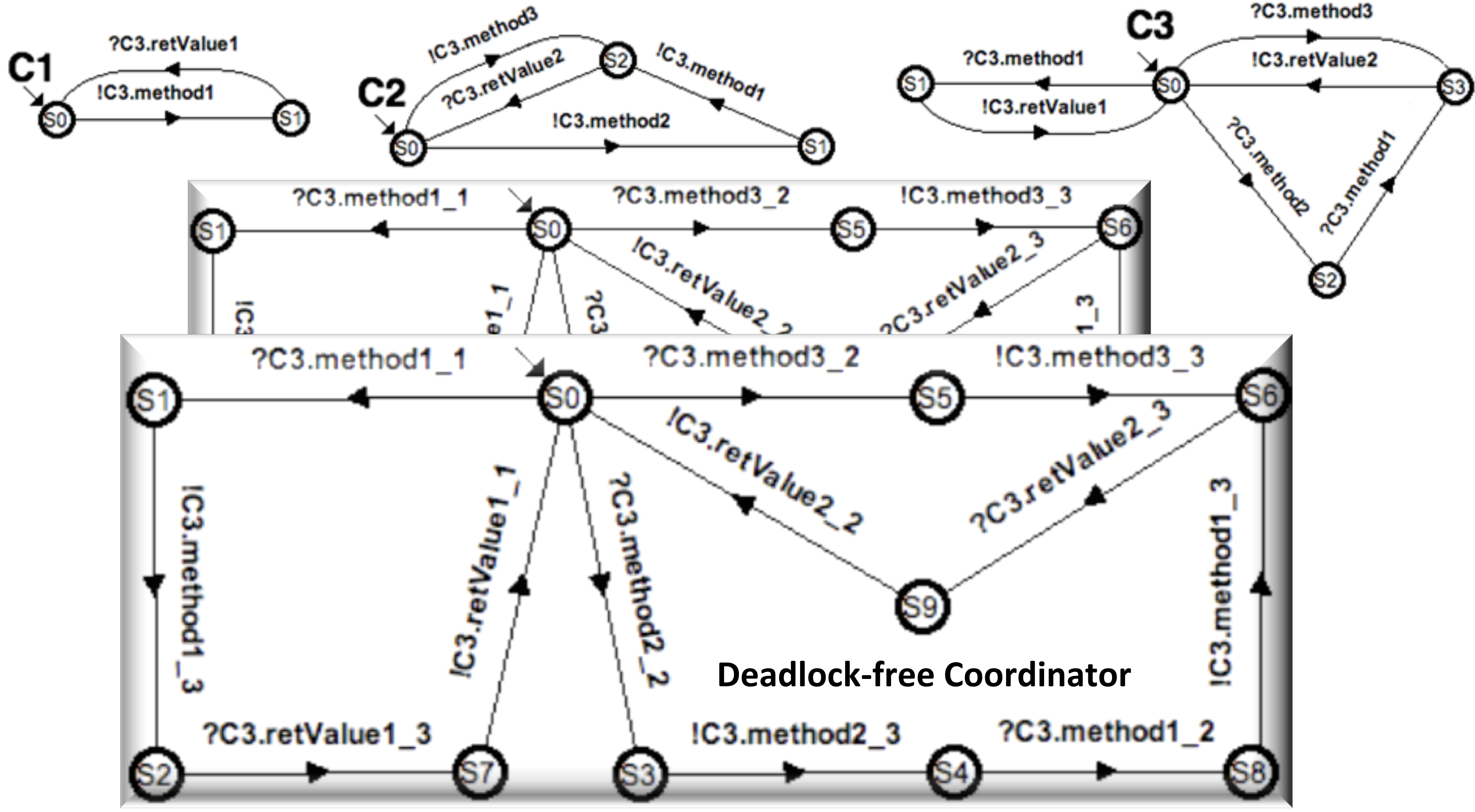
P



{!-C3.method1_1,!-C3.method1_2}
!C3.method1_2
!C3.method1_1
{!-C3.method1_1,!-C3.method1_2}
S0
S1

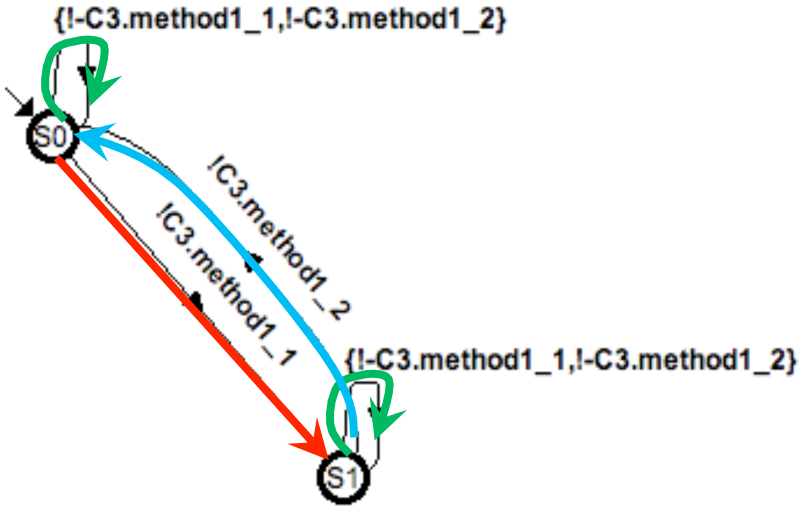
C1 and C2 has to interact by following an Alternating Interaction Protocol

Automatic Synthesis of **Centralized** FFC
– running example: enforcing deadlock freedom –

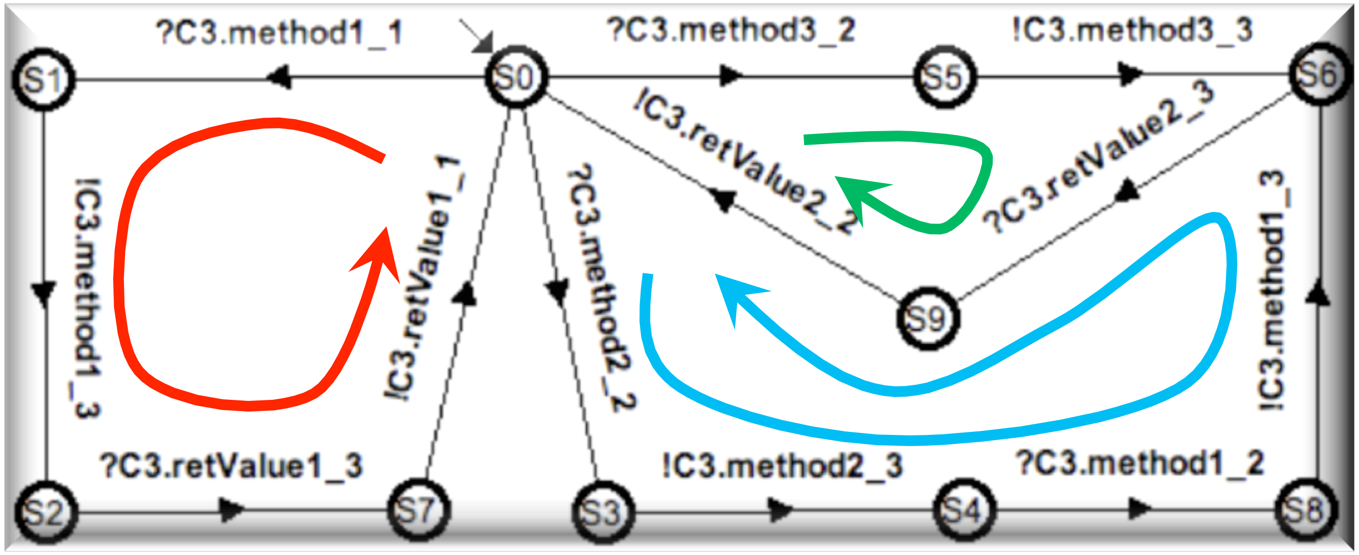


Automatic Synthesis of **Centralized** FFC
– running example: enforcing failure-freeness –

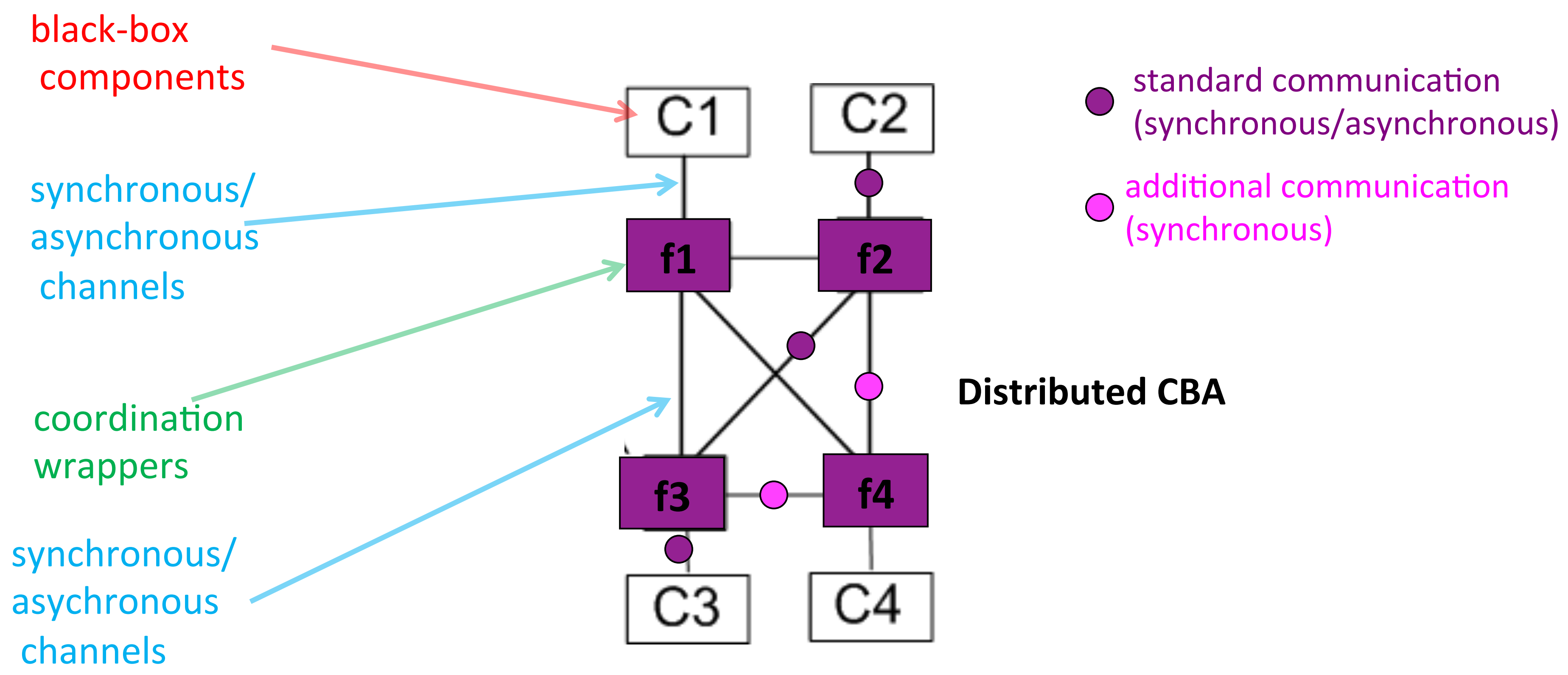
The Failure-free Coordinator is obtained by performing synchronous product between P and the deadlock-free coordinator



{!-C3.method1_1,!-C3.method1_2}
S0
!C3.method1_2
!C3.method1_1
{!-C3.method1_1,!-C3.method1_2}
S1



Automatic Synthesis of Distributed FFC – the reference architectural style –


black-box components
synchronous/ asynchronous channels
coordination wrappers
synchronous/ asychronous channels
C1
C2
f1
f2
f3
f4
C3
C4
Distributed CBA
● standard communication (synchronous/asynchronous)
● additional communication (synchronous)

Networked Systems (NS) Interoperability, Mediator Synthesis

- European FET Project Connect 2009-2013
- A more complex scenario, with lack of knowledge and standardization
- NSs that need to cooperate on the fly, to achieve a common goal G
  - Interoperability is *the* problem both at middleware and application level

Application-layer Protocols: Talking a lot

Application-layer protocols (as opposed to middleware-layer protocols)

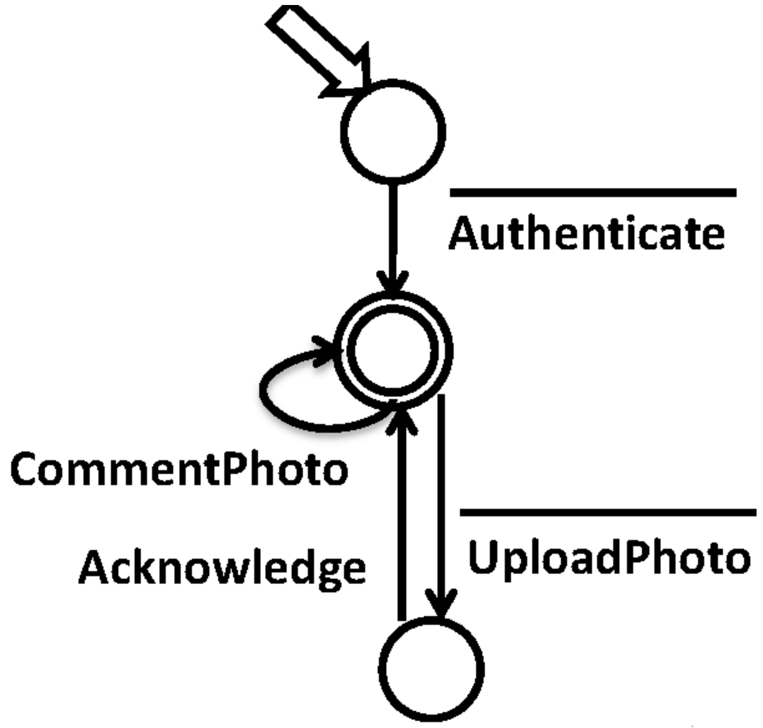
- behavior of a NS in terms of the *sequences of messages at the interface level*, which it may exchange with other systems
- Interactions are performed by following a given ordering in the way interface operations need to be executed
- the notion of protocol abstracts from the content of the exchanged messages, i.e., values of method/operation parameters, return values, etc.

Modeling Application-layer Protocols

- By using Labeled Transition Systems (LTSs) and introducing *final* states

Input actions, e.g., ***Acknowledge***, model
- methods that can be called;
- receiving messages;
- return values.

Output actions, e.g., $\overline{\textbf{\textit{Authenticate}}}$, model
- method calls;
- message transmission;
- exceptions.


$\overline{\text{Authenticate}}$
CommentPhoto
Acknowledge
$\overline{\text{UploadPhoto}}$

A Photo Sharing producer

Interoperability

- The ability of heterogeneous protocols to communicate and correctly coordinate to achieve *system* goal(s) (global property)

- **Communication** expressed as synchronization
  - two protocols communicate if they are able to synchronize on common actions
  - for application-layer protocols, it goes beyond single basic synchronizations and may require a well defined sequence of synchronization to be achieved (a primitive form of coordination)
    - E.g., $\overline{\text{sendItems}}$ <-> receiveItems *(simple case)*
      $\overline{\text{sendItems}}$ <-> receiveItem … receiveItem *(more complex case)*

- **Coordination** expressed as the achievement of a system goal
  - two protocols succeed in coordinating if they interact through synchronization according to the achievement of system goal(s)

- **Goal** usually specified in some automata-based or temporal logic formalism

The Interoperability Problem

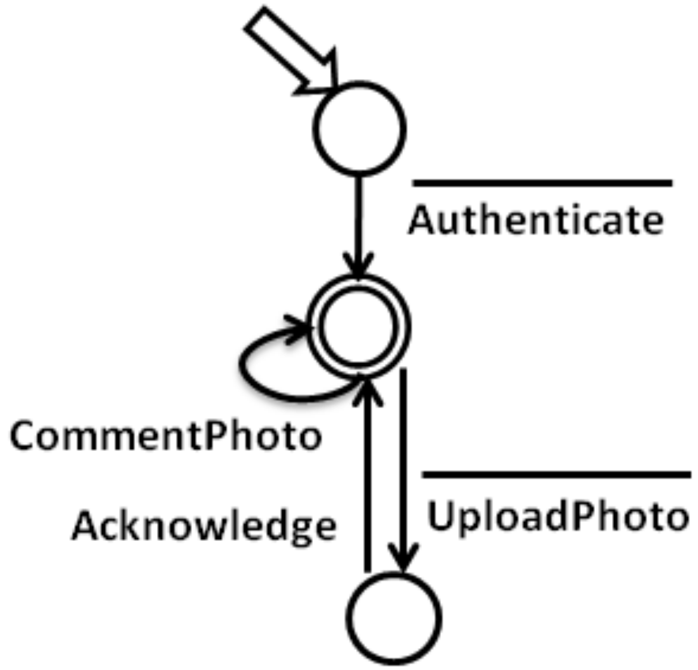
- It concerns the problem of both enabling ***communication*** and achieving ***correct coordination*** (w.r.t. the specified goal)

- Solution: automatic synthesis of ***application-layer connectors/ mediators***

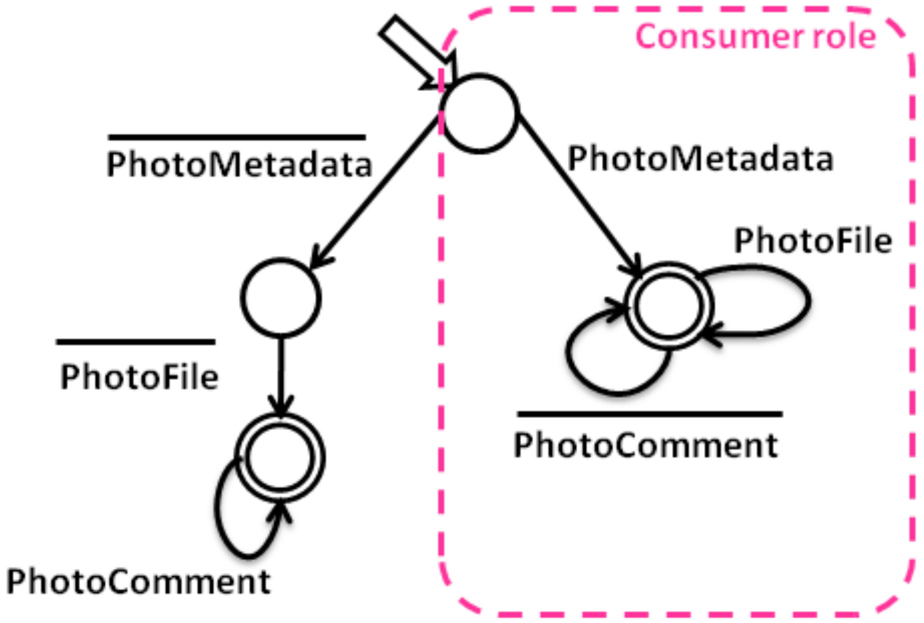
- Automatic ***coordinator*** synthesis (seen before)
  - the main focus is on addressing correct coordination by assuming the communication problem already solved

- Automatic ***mediator*** synthesis (comes next)
  - it focuses on the whole interoperability problem, i.e., addressing communication + correct coordination

The need for Mediators: the Photo Sharing Scenario



Infrastructure-based implementation of Photo Sharing Producer



Peer-to-peer implementation of Photo Sharing

Automatic Synthesis of Mediators

- **Problem**: *interoperability* between *heterogeneous* *protocols*
- **Goal**: to find an *automated solution* to solve the problem
- Compatible or functionally matching protocols: protocols that can potentially communicate by performing complementary sequences of actions
  - "Communication" through (at least one) complementary sequences of actions, i.e., trace
  - "Potentially" because of heterogeneities that can be mediated , i.e. mismatches (e.g. languages, third parties sequences of actions, ...)
- Interoperability: ability of heterogeneous protocols that functionally match to communicate and coordinate, i.e., synchronize to reach their goal(s)

Obtaining Mediators

Protocols

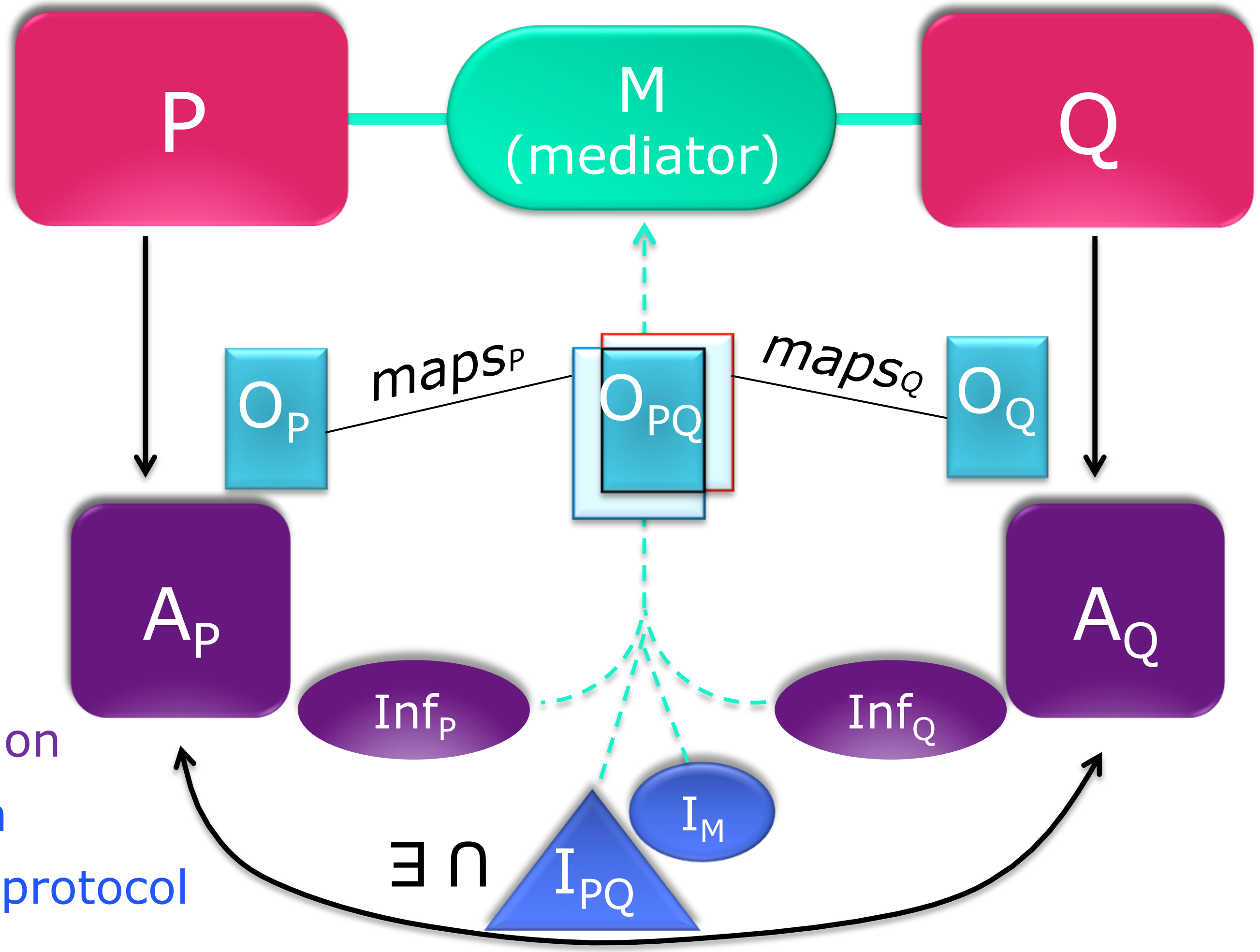
Ontologies

Abstracted protocols

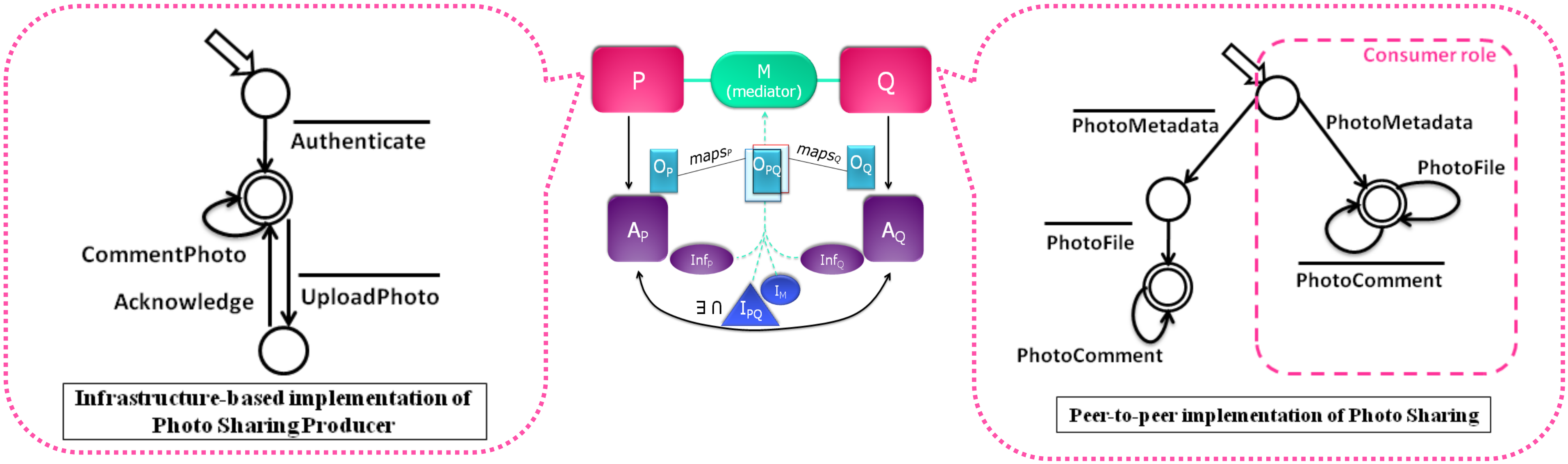
Abstraction information

Matching information

Common abstracted protocol

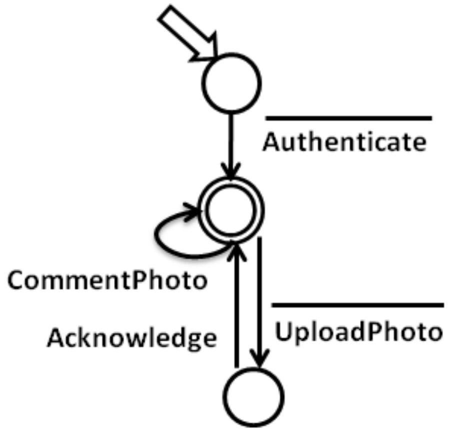


Obtaining Mediators

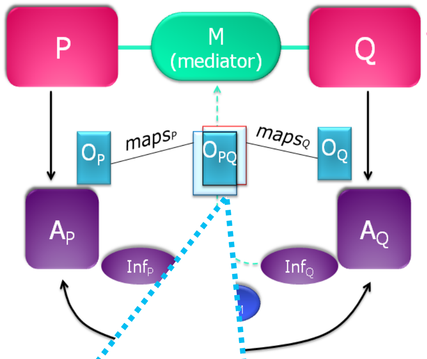

$\overline{\text{Authenticate}}$
CommentPhoto
Acknowledge
$\overline{\text{UploadPhoto}}$
Infrastructure-based implementation of Photo Sharing Producer
P
M (mediator)
Q
O_P
$maps_P$
O_{PQ}
$maps_Q$
O_Q
A_P
A_Q
Inf_P
Inf_Q
I_M
$\exists \cap$
I_{PQ}
Consumer role
$\overline{\text{PhotoMetadata}}$
PhotoMetadata
PhotoFile
$\overline{\text{PhotoFile}}$
$\overline{\text{PhotoComment}}$
PhotoComment
Peer-to-peer implementation of Photo Sharing

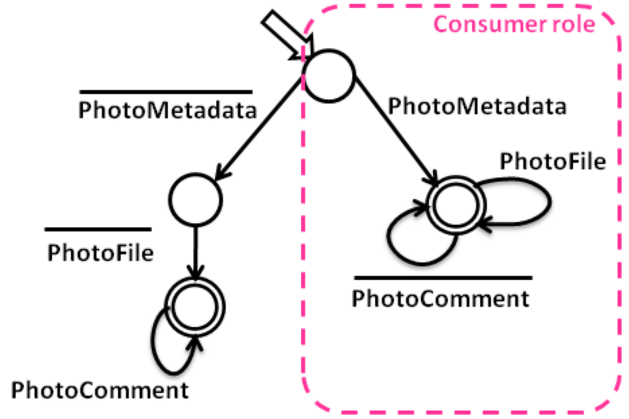
- **Protocols** as Labelled Transition Systems (LTSs)
- Initial state + *final state* define the *coordination policies* (traces)

Obtaining Mediators



Infrastructure-based implementation of Photo Sharing Producer



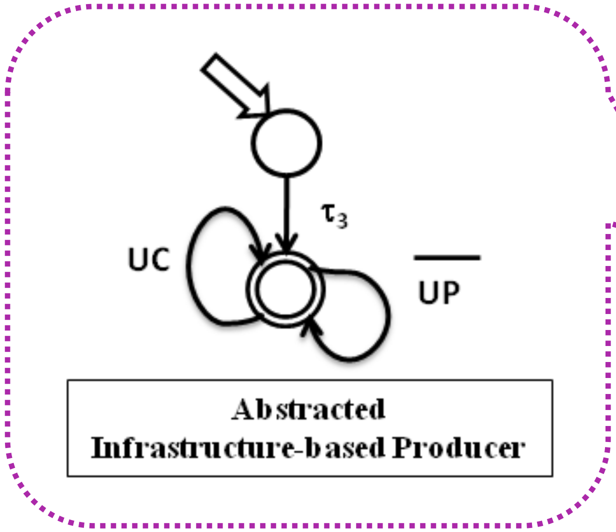


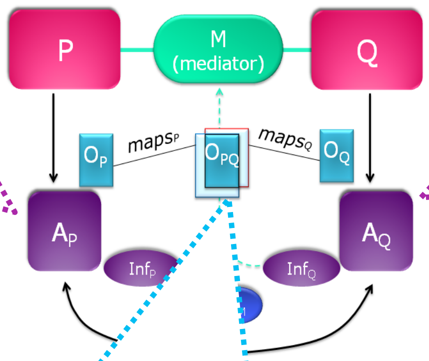
Peer-to-peer implementation of Photo Sharing

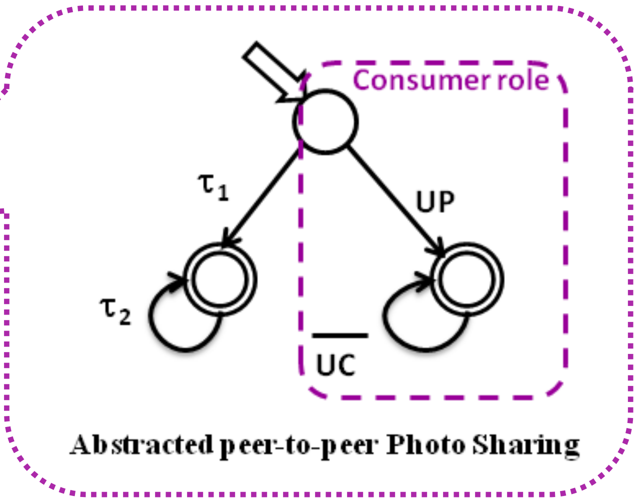
Infrastructure-based Photo Sharing Producer	Common Language		Peer-to-peer Photo Sharing
$\overline{\text{UploadPhoto.}}$ Acknowledge	$\overline{UP}$ *(upload photo)*	*UP* *(download photo)*	PhotoMetadata. PhotoFile
CommentPhoto	*UC* *(download comment)*	$\overline{UC}$ *(upload comment)*	$\overline{\text{PhotoComment}}$
-	-	τ_1	$\overline{\text{PhotoMetadata}}$. $\overline{\text{PhotoFile}}$
-	-	τ_2	PhotoComment
$\overline{\text{Authenticate}}$	τ_3	-	-

- **Ontologies** describing:
  - the *semantics* of the protocols *actions*
  - the *common language*
  - *taus* for *third parties communications*

Obtaining Mediators



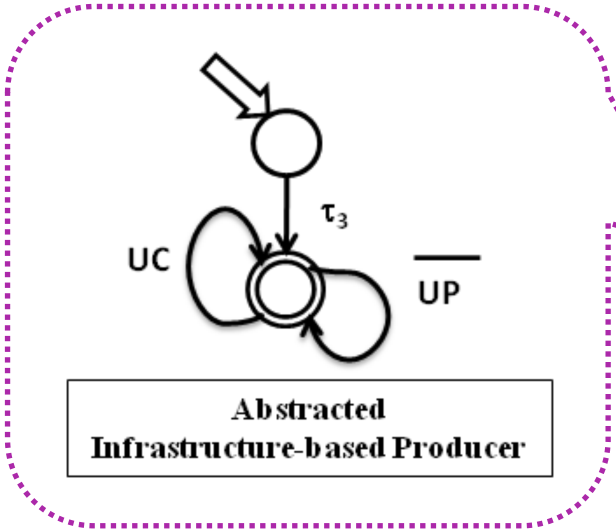


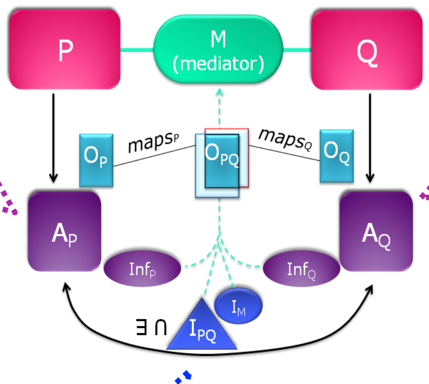


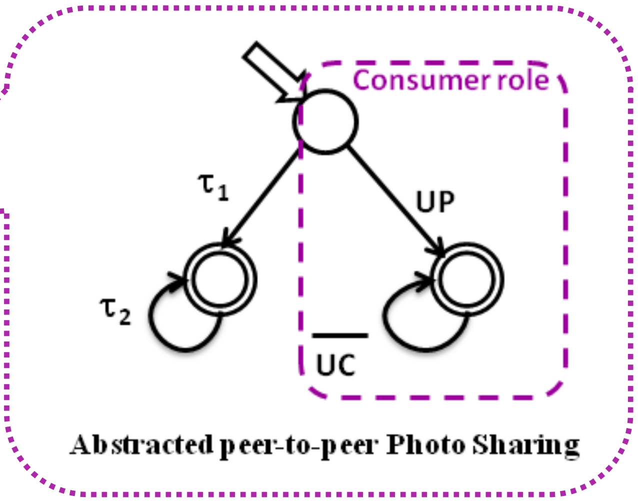
Infrastructure-based Photo Sharing Producer	Common Language		Peer-to-peer Photo Sharing
$\overline{\text{UploadPhoto.}}$ Acknowledge	$\overline{UP}$ *(upload photo)*	*UP* *(download photo)*	PhotoMetadata. PhotoFile
CommentPhoto	*UC* *(download comment)*	$\overline{UC}$ *(upload comment)*	$\overline{\text{PhotoComment}}$
-	-	τ_1	$\overline{\text{PhotoMetadata.}}$ $\overline{\text{PhotoFile}}$
-	-	τ_2	PhotoComment
$\overline{\text{Authenticate}}$	τ_3	-	-

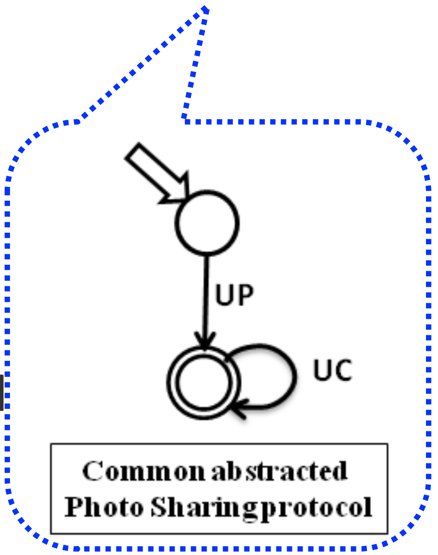
- **Abstraction:**
  - *relabeling* of protocols with *common language and taus*

Obtaining Mediators





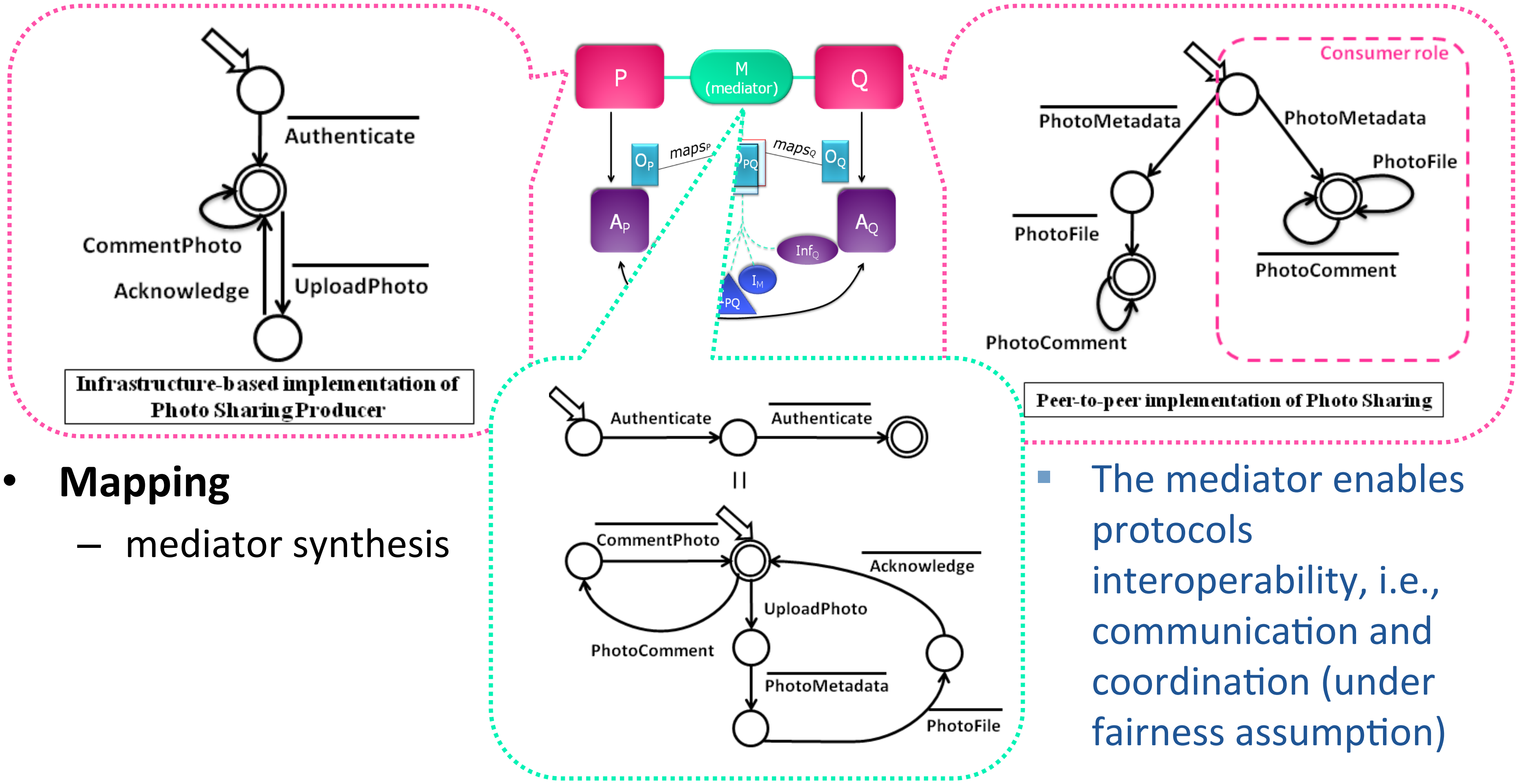




- **Compatibility or Functional matching:**
  - *complementary traces* modulo ***mismatches*** and *third parties communications*

- **Successful matching:**
  - a mediator exists and
  - it can be automatically synthesized

Obtaining Mediators



- **Mapping**
  - mediator synthesis

- The mediator enables protocols interoperability, i.e., communication and coordination (under fairness assumption)

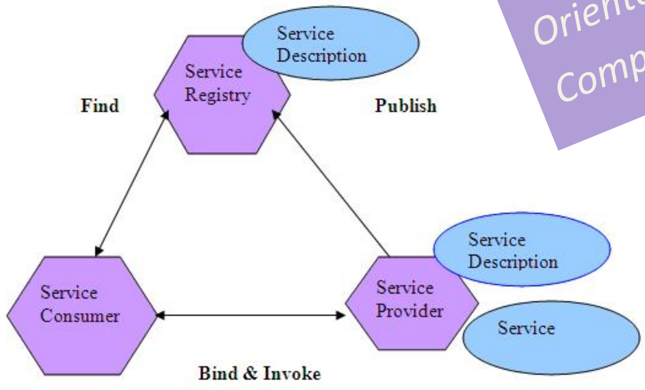
Adding performance concerns

- We build on top of our solution to the *automated synthesis of connectors* to
  - take into account performance concerns during the synthesis proces
  - make the synthesized connectors *(self-)adaptive* with respect to runtime performance requirements changes

- By reasoning on systems' specification, the approach:
  - produces a mediator that satisfies the functional requirements
  - acts on the produced mediator to let it satisfy performance issues and to make it (self-)adaptive

What is the present/future?
Services, Apps and Clouds in the *air*

- A virtually *infinite* number of software applications that provide computational software resources in the *open Digital Space*



Service Oriented Computing

Cloud Computing

Apps Development

Developer as an integrator

- The developer in the digital space is more and more an integrator
- It relies on third party artifacts and it is the owner of the integration code **only**

•How do we achieve confidence in the final system?

•How do we easy the development process?

Software Production

- Expectations/requirements can be thought as expressing a **goal**
- The integrated software behaves as expected/required both functionally and non functionally

  Integration means: Enterprise integration patterns, connectors, Mediators, adapters, controllers, wrappers, coordinators, Orchestrations, Choreographies

- to foster a correct reuse with respect to a given goal, we should know the **actual functional and non-functional behavior of the software** being reused
- Assessed by means of suitable software ***models***
- (protocol) Models through experimental observation (***mining***)

Another Sign of Science in Computer Science?

- Peter Denning's Viewpoint in June 2013's ACM Communication
- *"There is a growing consensus today that many of the issues we are studying are so complex that only an experimental approach will lead to understanding"*

Empirical Software Engineering

1996 Victor Basili's editorial to ESEE

http://www.cs.umd.edu/projects/SoftEng/tame/ESEEdit.htmldefine

- Galileo's scientific method
  - **Observation : <u>quantitative</u> characterization of the observed phenomenon**
  - **Theory/Model construction of the phenomenon to interpret**
  - **Validation through experimental verification**

Where do models come from?

From *Creationistic View* to *Experimental View*

Creationistic view

- A producer is the owner of the artifact, and with the right tools she can supply any piece of information

Experimental view

- The knowledge of a software artifact is limited to what can be ***observed*** of it
- Theoretical barrier limits the power and extent of observations

The Envisioned Production Process

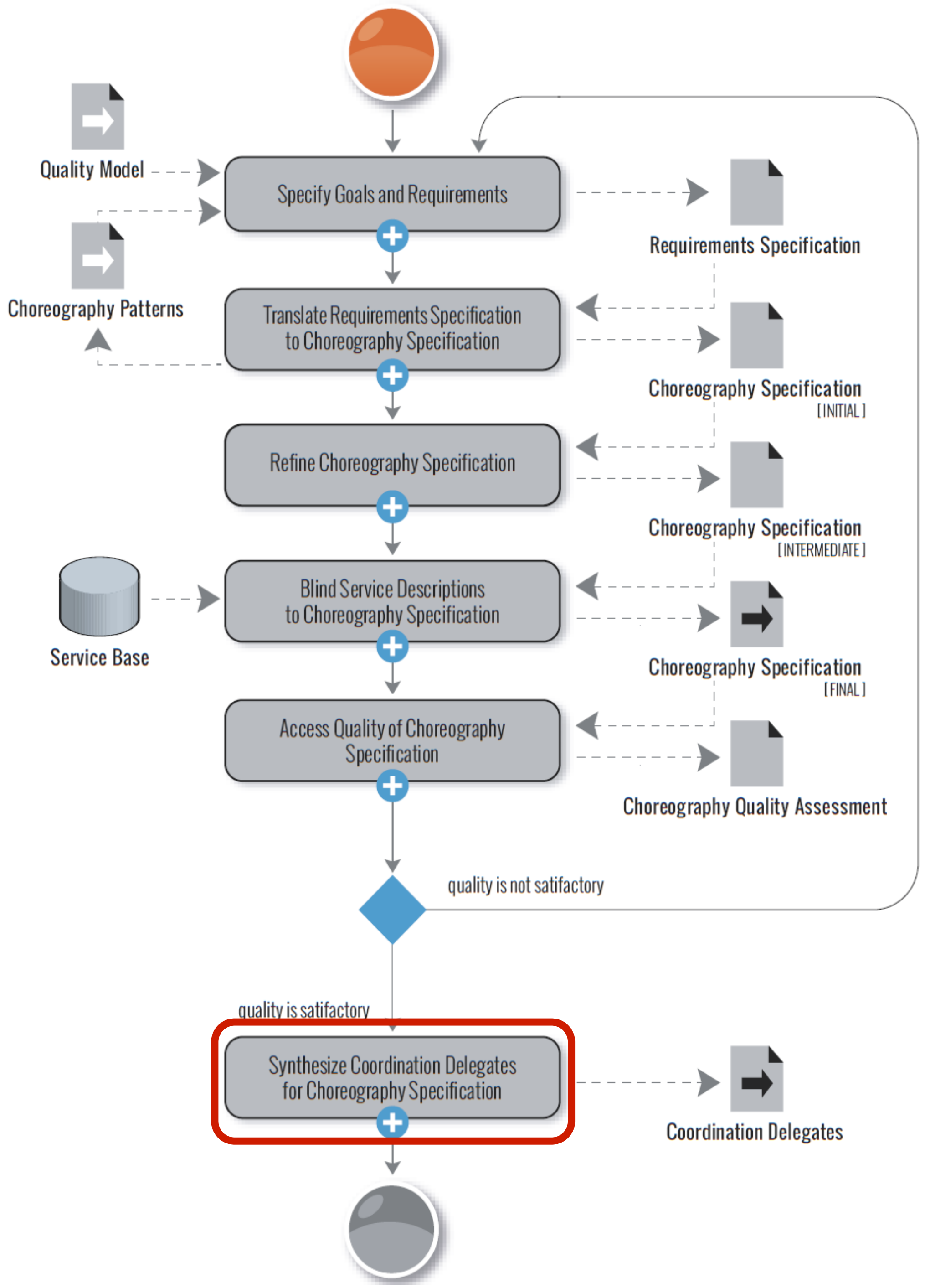
Elicit

- Given a software service **S**, elicitation techniques are used to produce **models** as complete as possible with respect to a goal **G**.
- **Models** may in general exhibit degrees of **incompleteness**, provided that they are accurate enough to allow the development of a correct integration w.r.t. the goal **G**.

Integrate

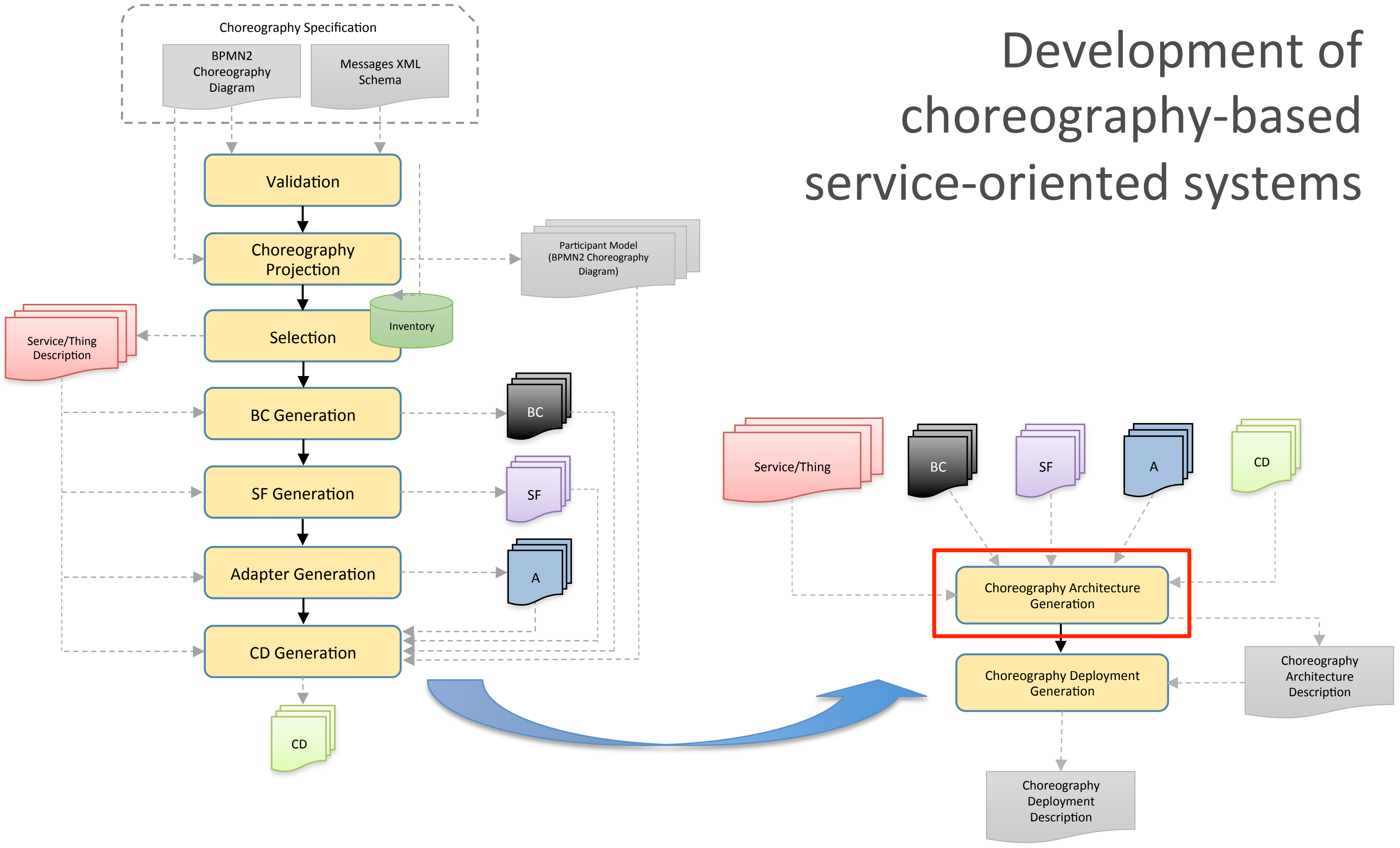
- Assist the developer in creating the appropriate integration means to compose the observed software together in order to produce an application that satisfies the goal **G**.

Development of choreography-based service-oriented systems

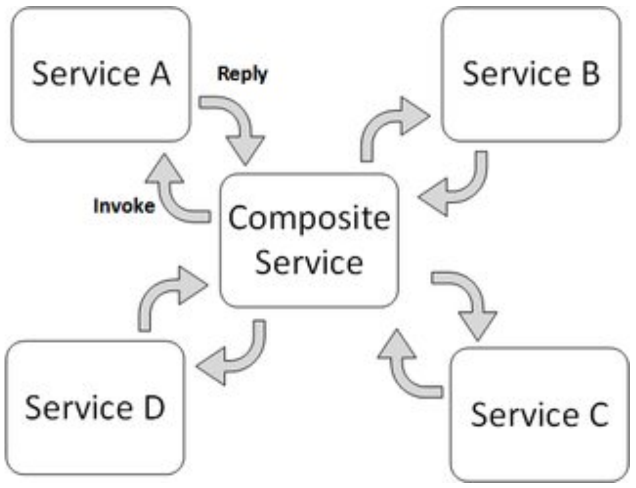


Quality Model
Choreography Patterns
Specify Goals and Requirements
Requirements Specification
Translate Requirements Specification to Choreography Specification
Choreography Specification [INITIAL]
Refine Choreography Specification
Choreography Specification [INTERMEDIATE]
Service Base
Blind Service Descriptions to Choreography Specification
Choreography Specification [FINAL]
Access Quality of Choreography Specification
Choreography Quality Assessment
quality is not satifactory
quality is satifactory
Synthesize Coordination Delegates for Choreography Specification
Coordination Delegates

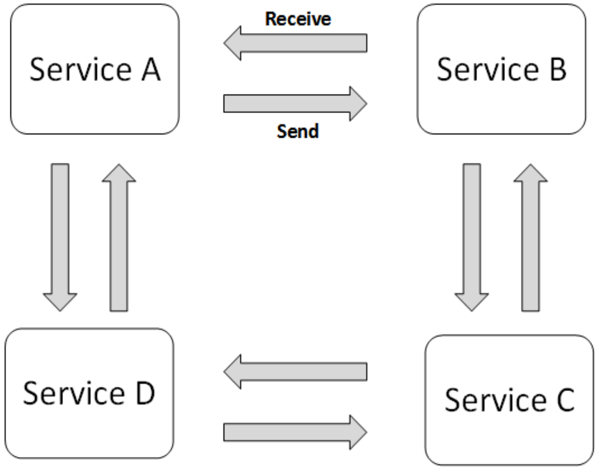
Development of choreography-based service-oriented systems


Choreography Specification
BPMN2 Choreography Diagram
Messages XML Schema
Validation
Choreography Projection
Participant Model (BPMN2 Choreography Diagram)
Selection
Inventory
Service/Thing Description
BC Generation
BC
SF Generation
SF
Adapter Generation
A
CD Generation
CD
Service/Thing
BC
SF
A
CD
Choreography Architecture Generation
Choreography Deployment Generation
Choreography Architecture Description
Choreography Deployment Description

Composition approaches



Orchestration (centralized)



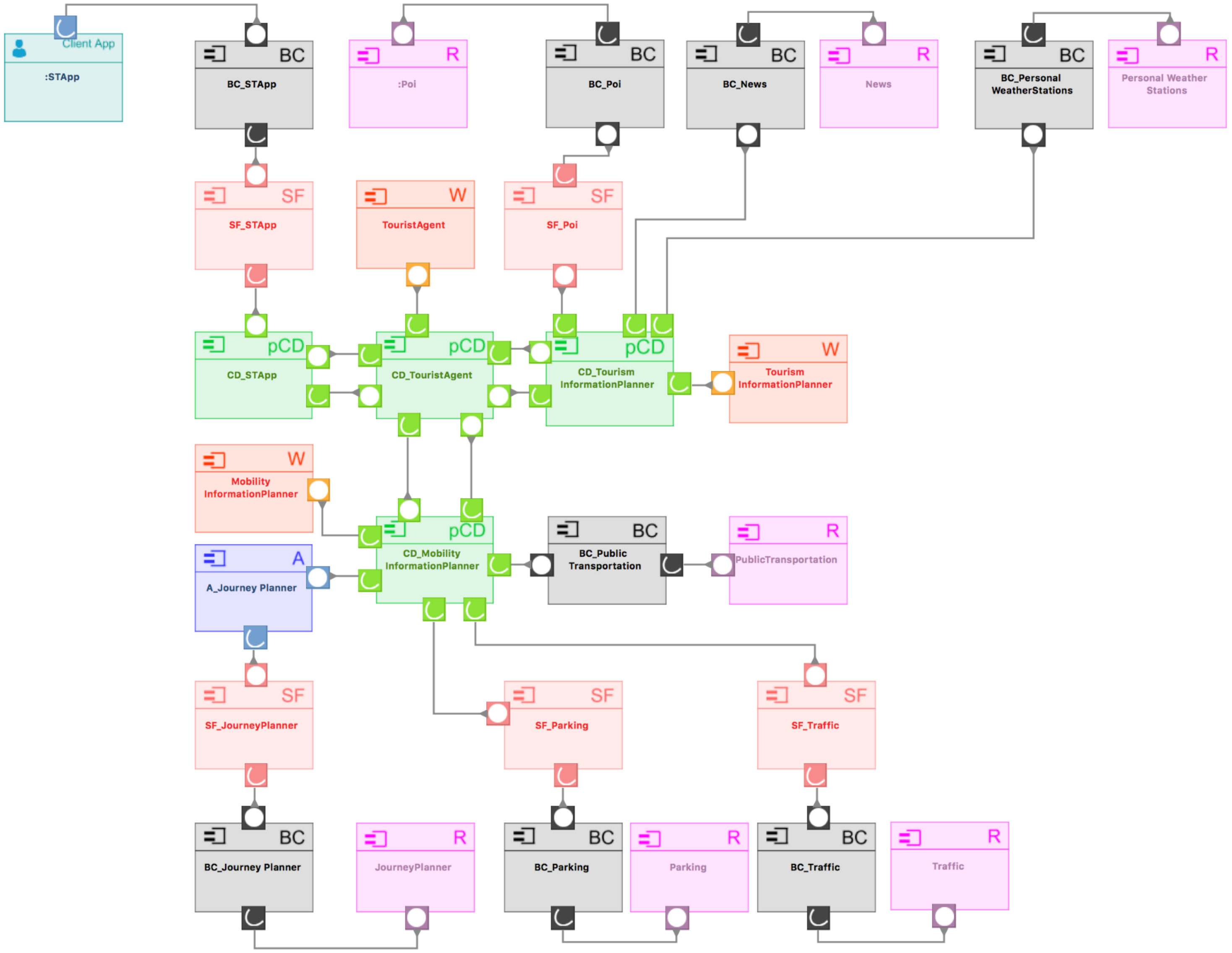
Choreography (fully distributed)

Local centralized view from the perspective of one participant

Global decentralized view from a multi-participant perspective

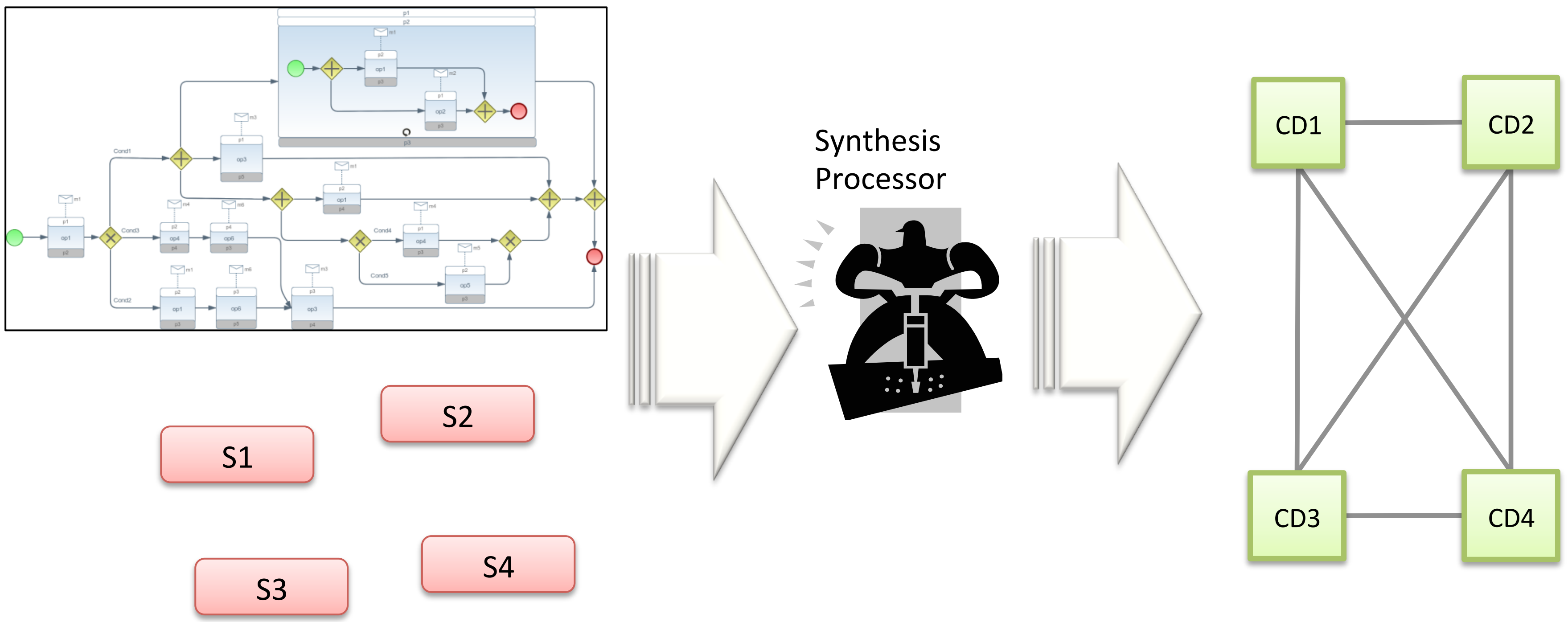
(albeit without a central controller)

Synthesized Choreography Architecture (a sample of)

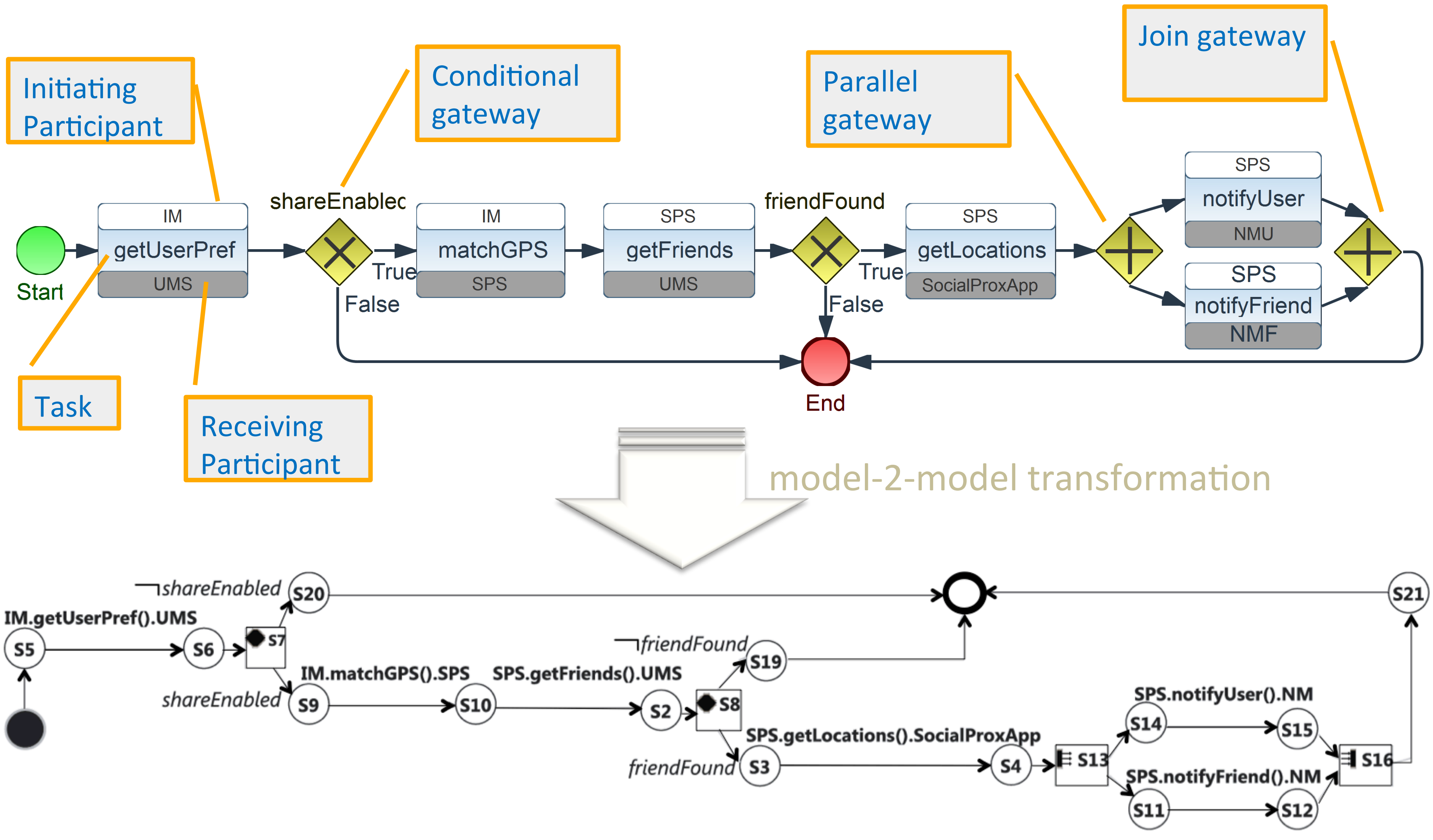


Choreography realizability enforcement

- Given a choreography specification, and
- a set of existing services discovered as “suitable” participants,
- restrict the interaction among them so to fulfill the collaboration prescribed by the choreography specification, hence
- preventing undesired interactions



- BPMN2 is the *standard de facto* for specifying choreographies
- BPMN2 offers a powerful notation called *Choreography Diagrams*

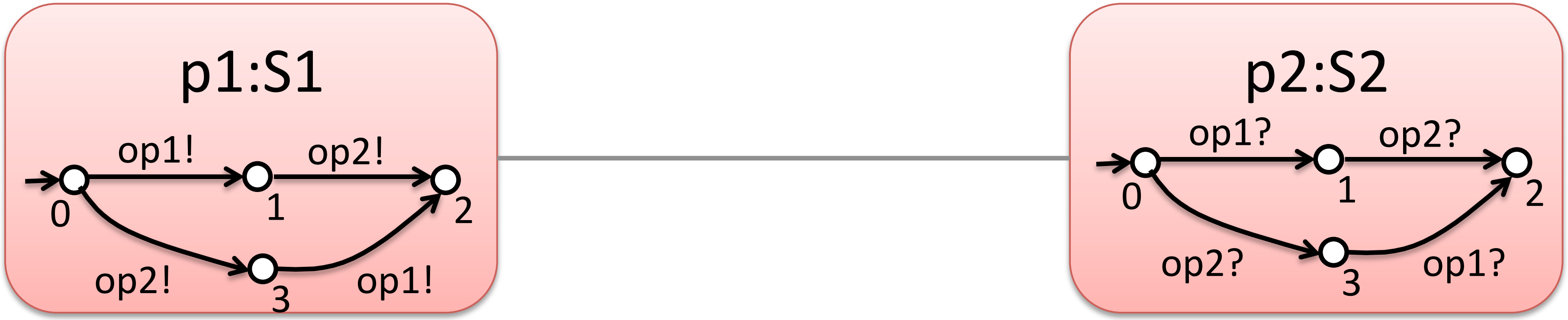


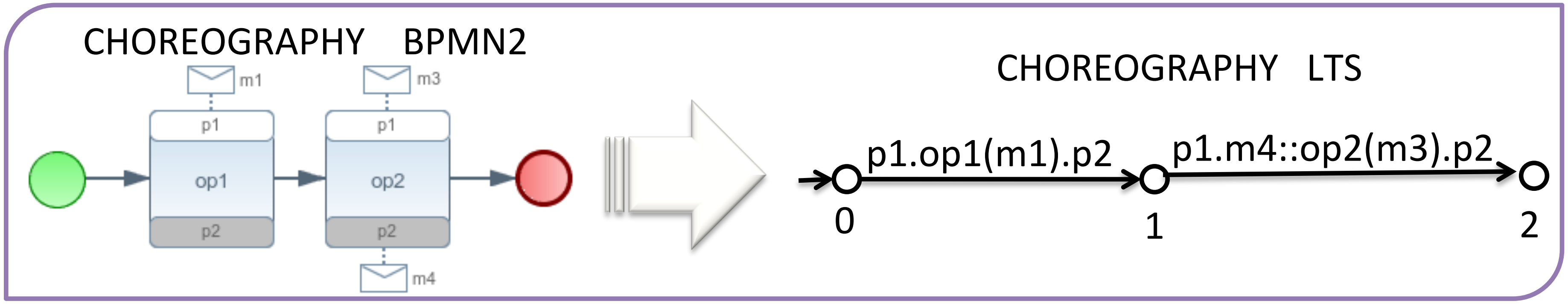
Undesired interactions

Undesired interactions are those interactions that do not belong to the set of interactions modeled by the given choreography and that can happen by letting the discovered services collaborate in an uncontrolled way

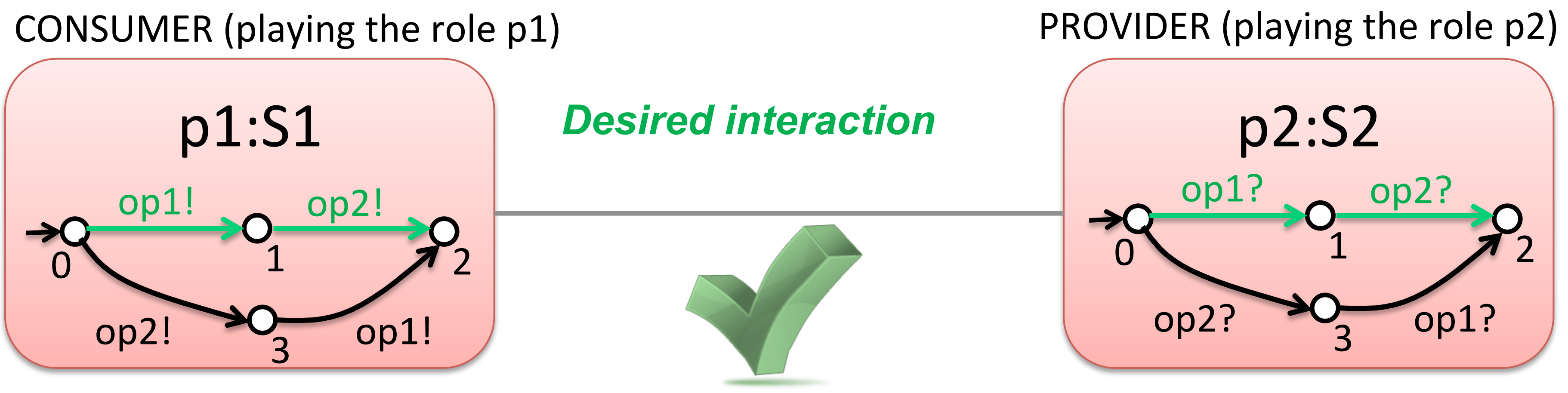
CONSUMER (playing the role p1)

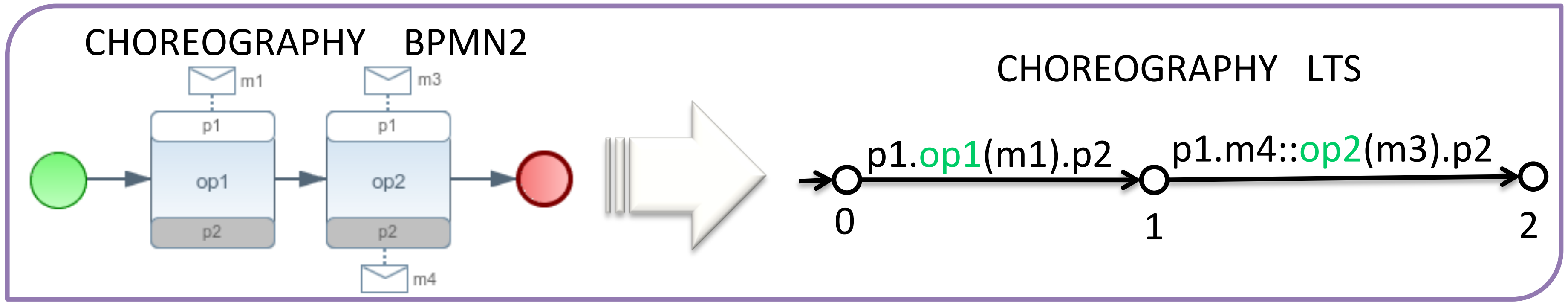
PROVIDER (playing the role p2)



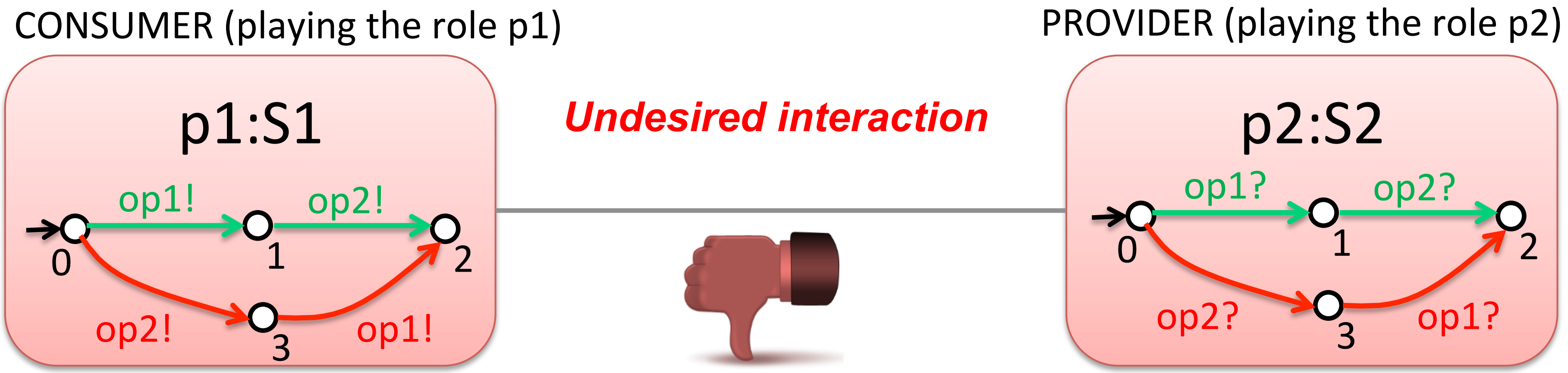


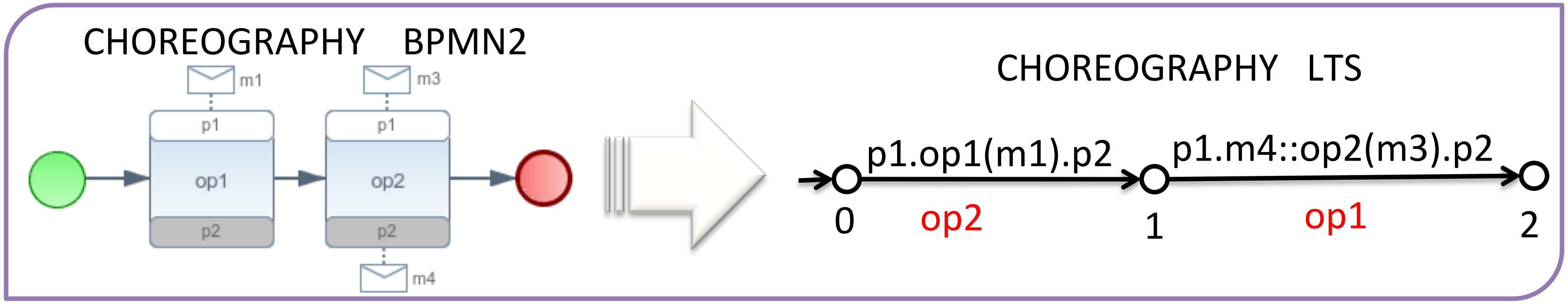
Undesired interactions



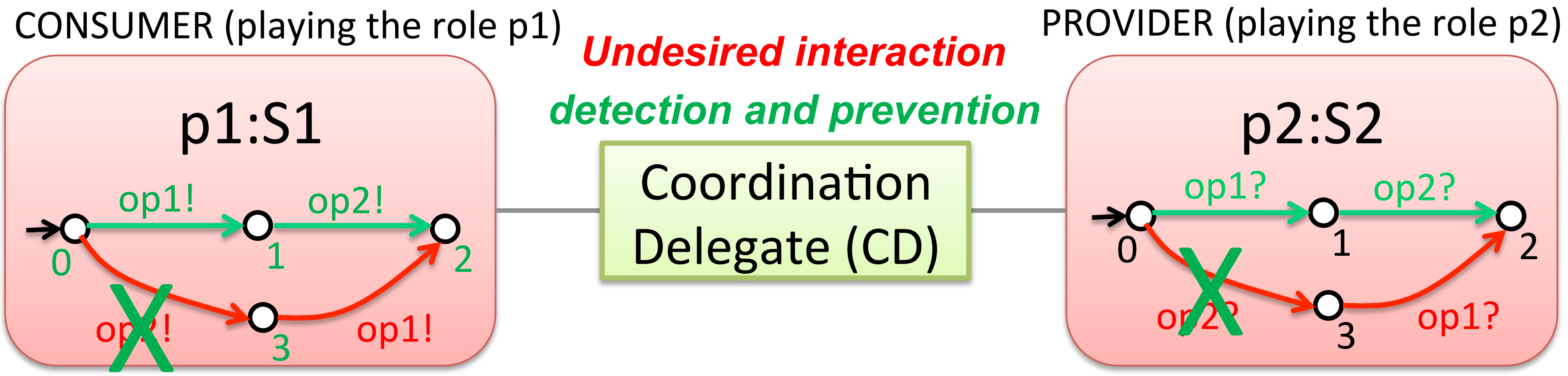


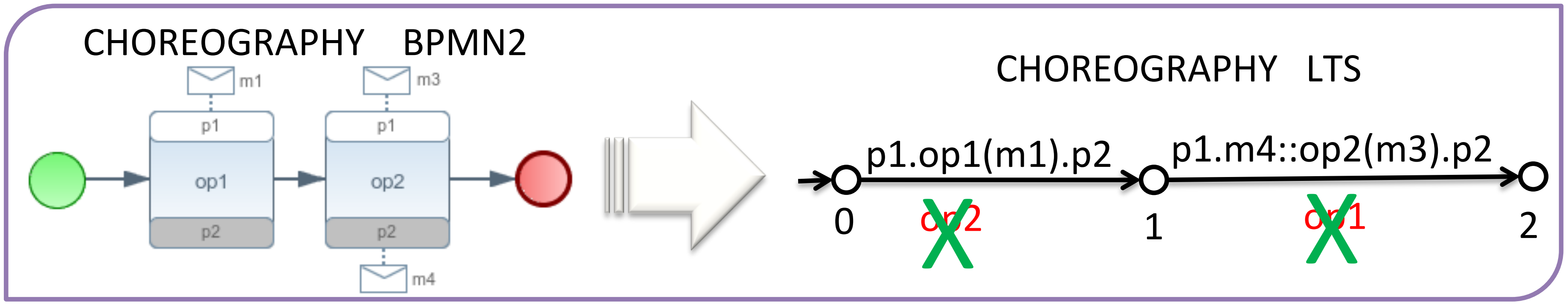
Undesired interactions



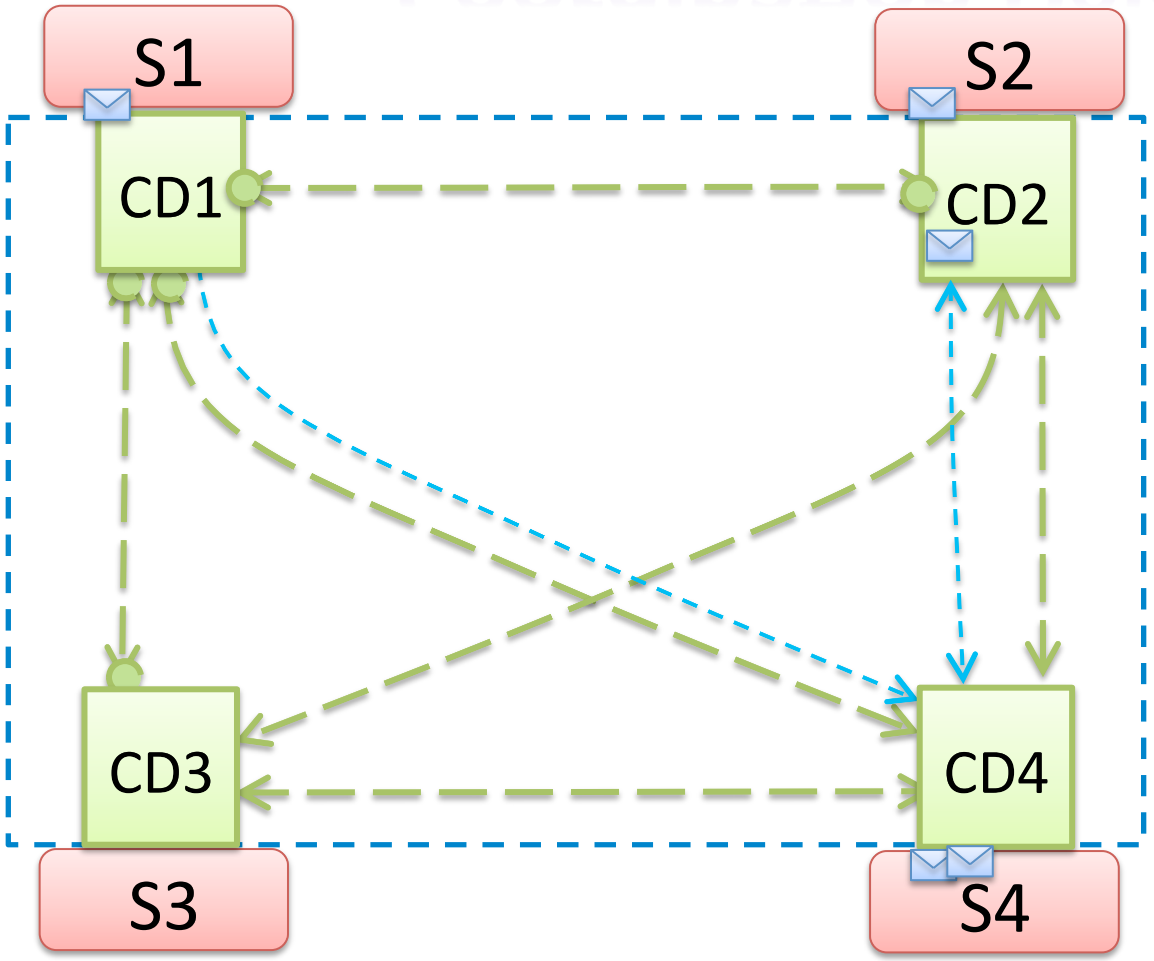


Undesired interactions





Realizability enforcement via Coordination Delegates



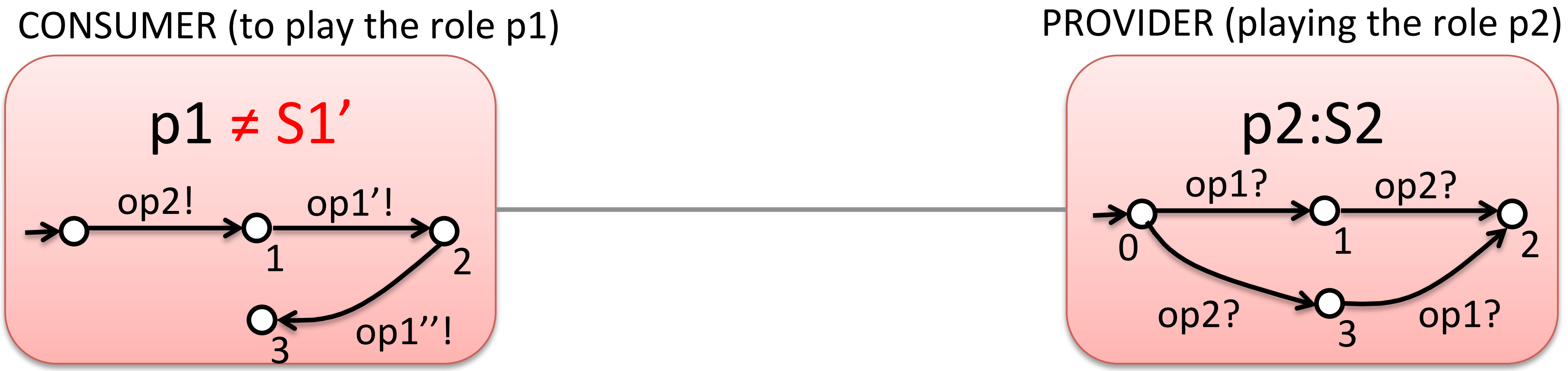
✉ standard communication (e.g., In-only/In-Out message exchange)

● additional communication: coordination information for coordination purposes (exchanged among the Coordination Delegates to enforce the realization of the specified choreography, in a distributed way)

choreography realization *(composition of CHOReOS connectors plus CDs)*

Mismatching interactions

Mismatching interactions are those interactions that differ in the semantics and granularity of the operations, and in the control structure of the protocols

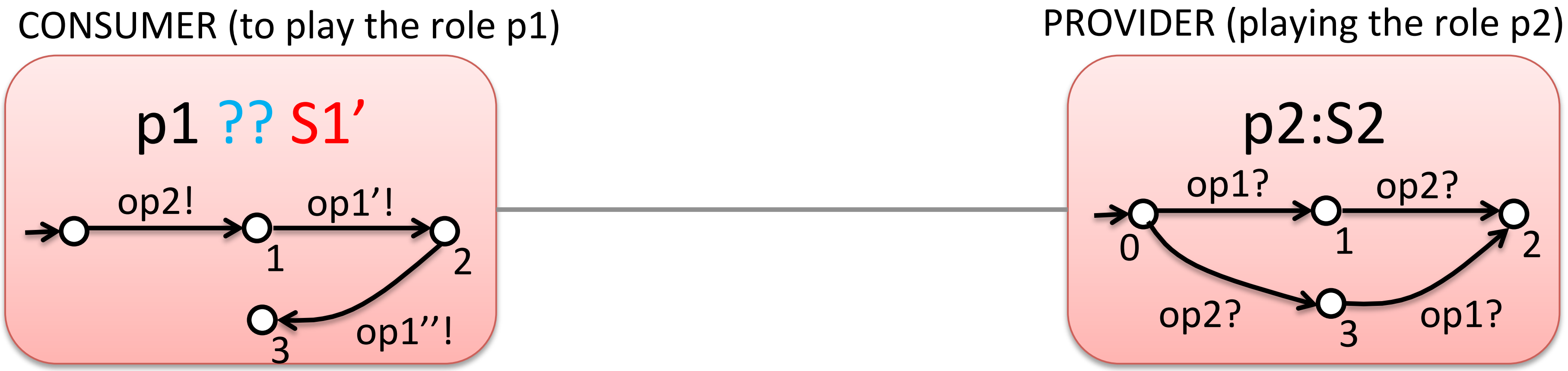


CONSUMER (to play the role p1)
PROVIDER (playing the role p2)
p1 ≠ S1'
op2!
op1'!
op1''!
1
2
3
p2:S2
op1?
op2?
op2?
op1?
0
1
2
3

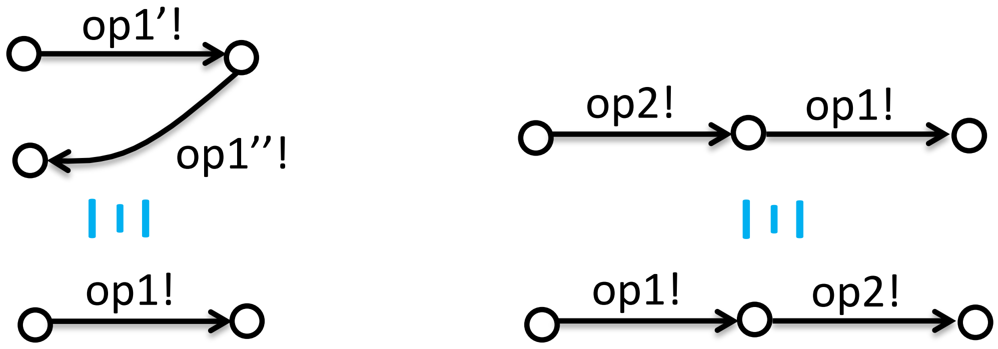
Let us suppose that, instead of discovering S1, another service, say S1', would have been discovered

Mismatching interactions

Mismatching interactions are those interactions that differ in the semantics and granularity of the operations, and in the control structure of the protocols

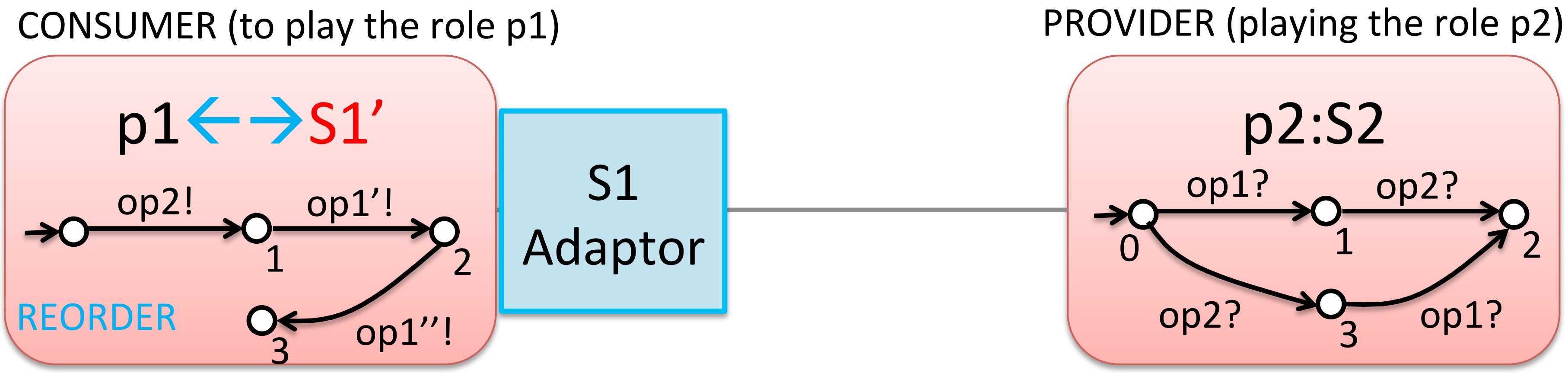

CONSUMER (to play the role p1)
PROVIDER (playing the role p2)
p1 ?? S1'
op2!
op1'!
op1''!
1
2
3
p2:S2
op1?
op2?
op2?
op1?
0
1
2
3

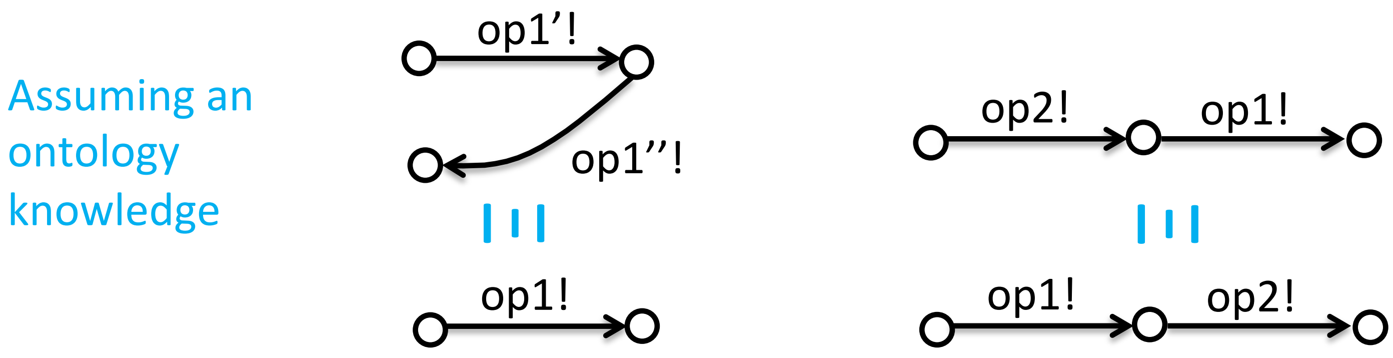
Assuming an ontology knowledge



Mismatching interactions

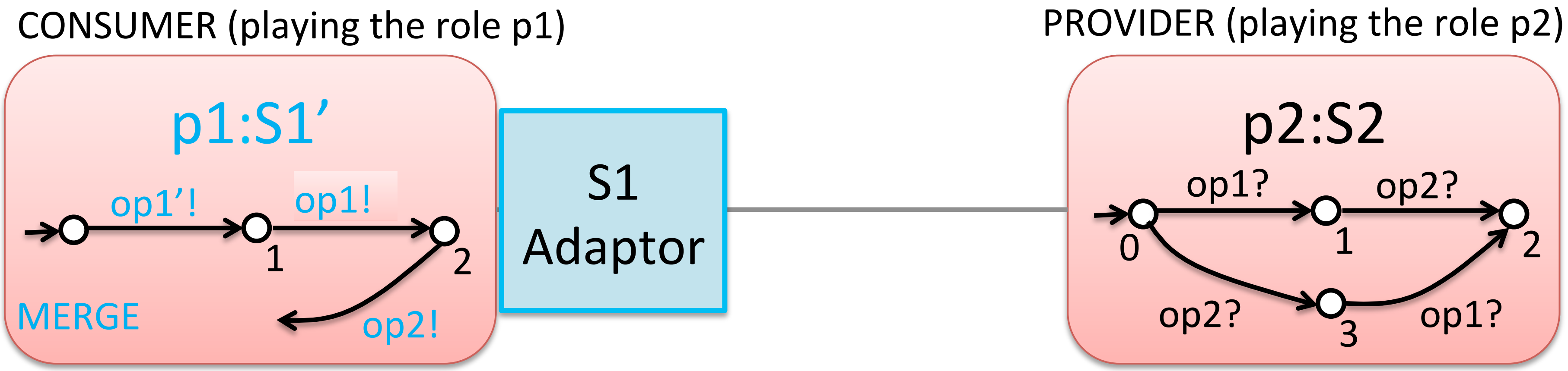
Mismatching interactions are those interactions that differ in the semantics and granularity of the operations, and in the control structure of the protocols

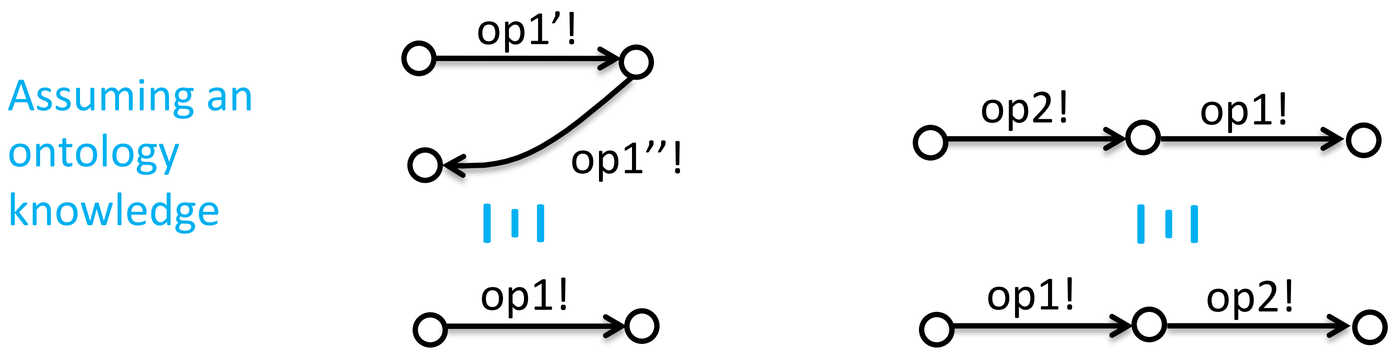




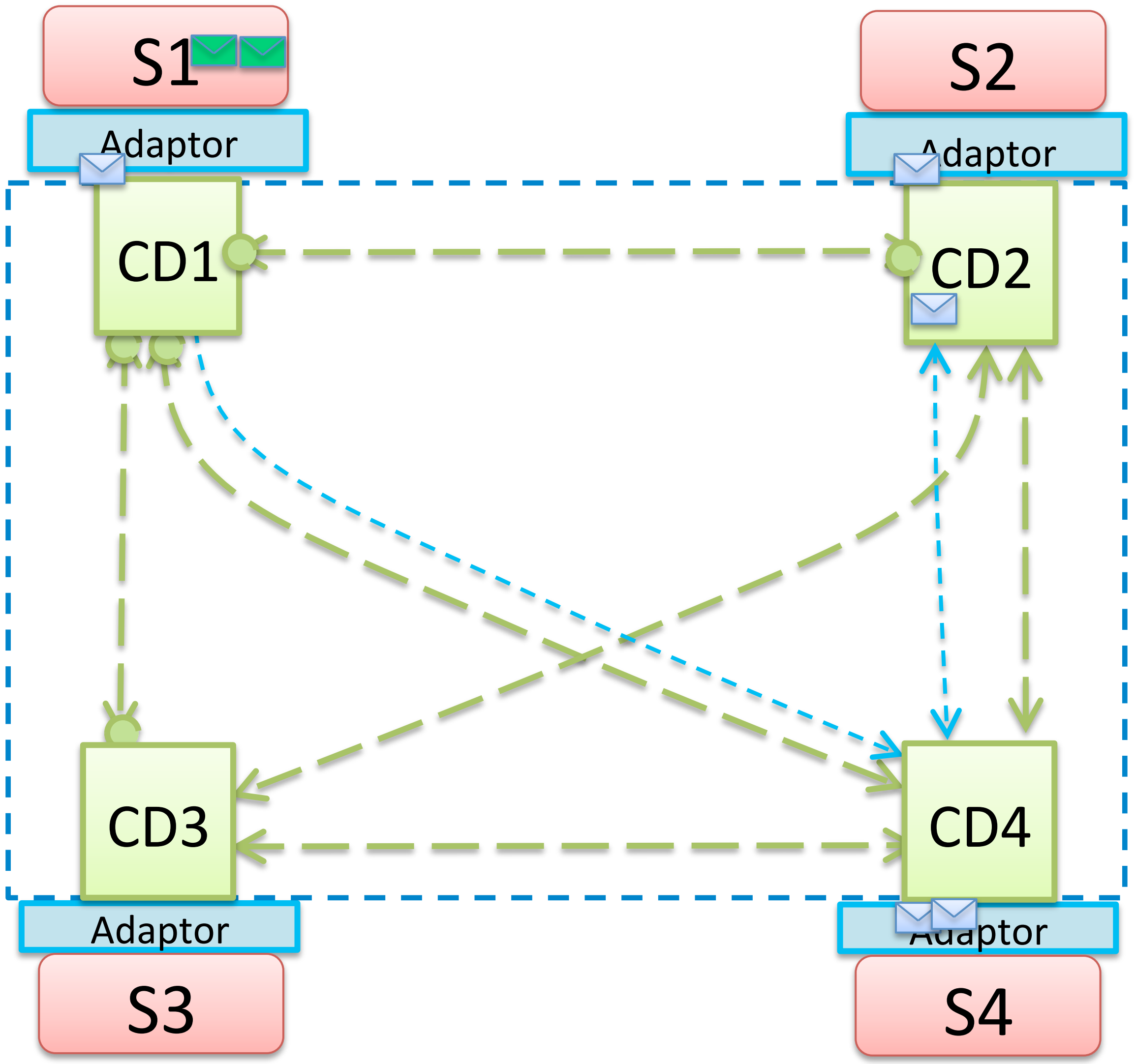
Mismatching interactions

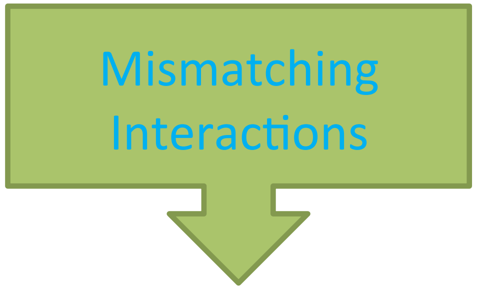
Mismatching interactions are those interactions that differ in the semantics and granularity of the operations, and in the control structure of the protocols



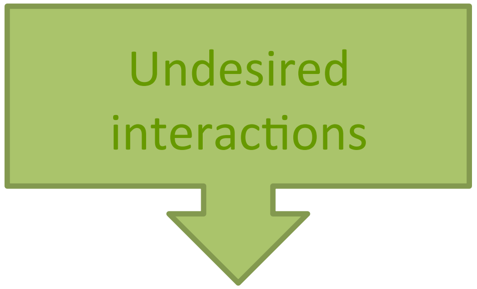


Overall architectural style



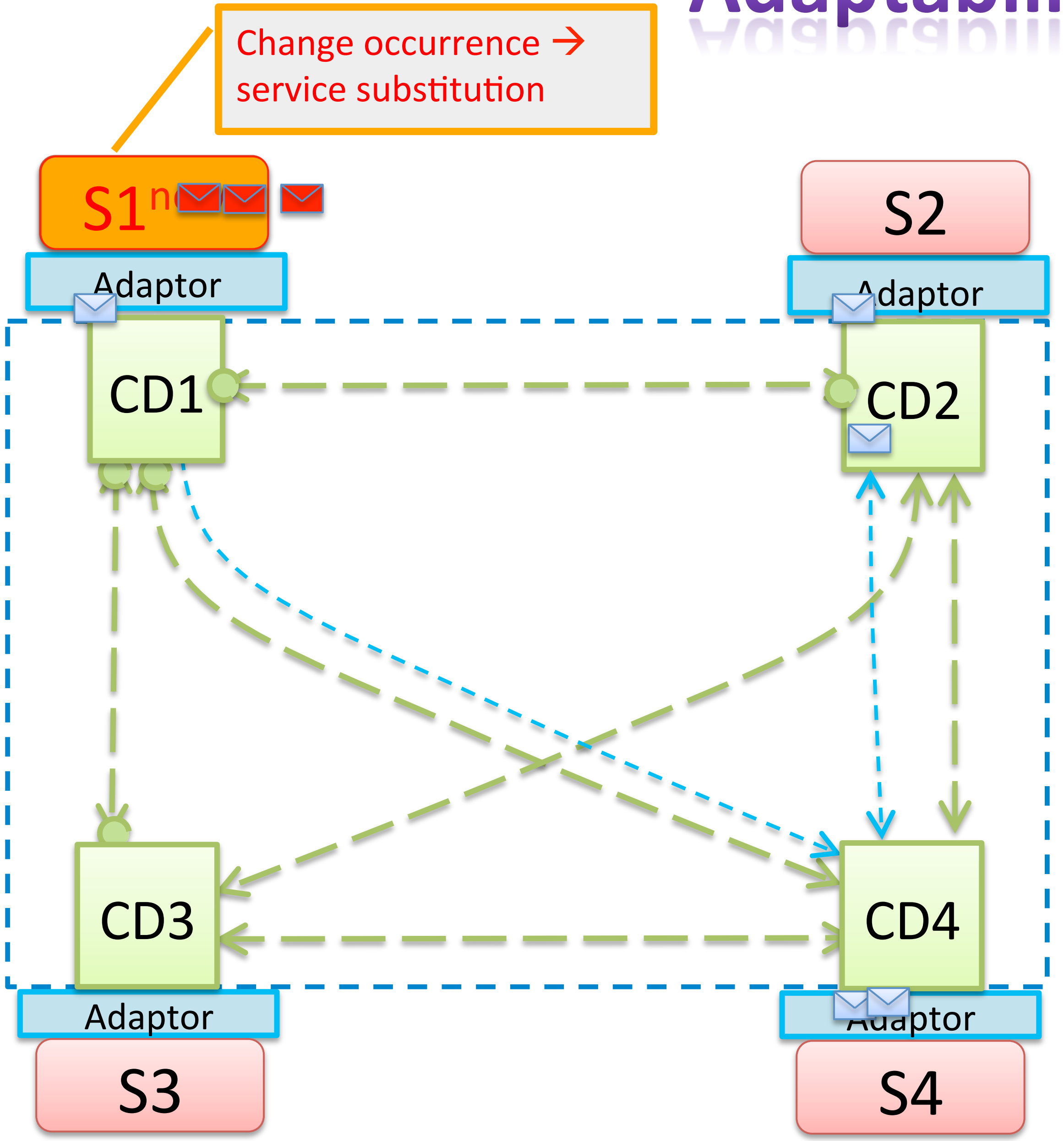


ADAPTORS



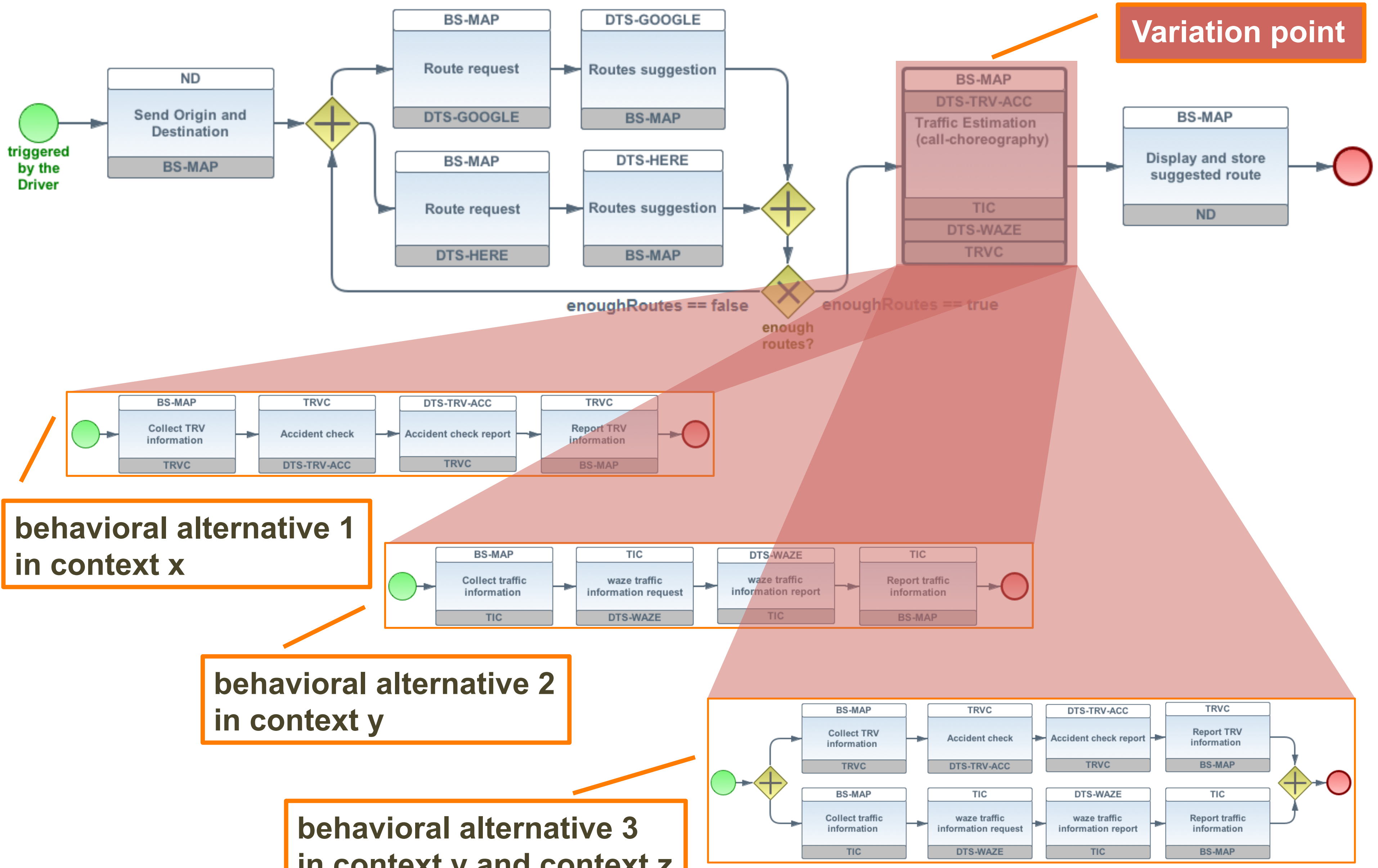
CDs

Adaptability



Choreography evolution through adaptation to possible changes in the discovered services, while still keeping the prescribed coordination.

Choreography evolution



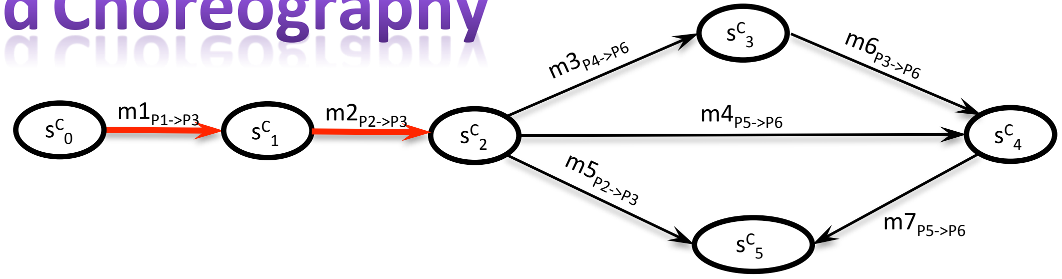
Variation point

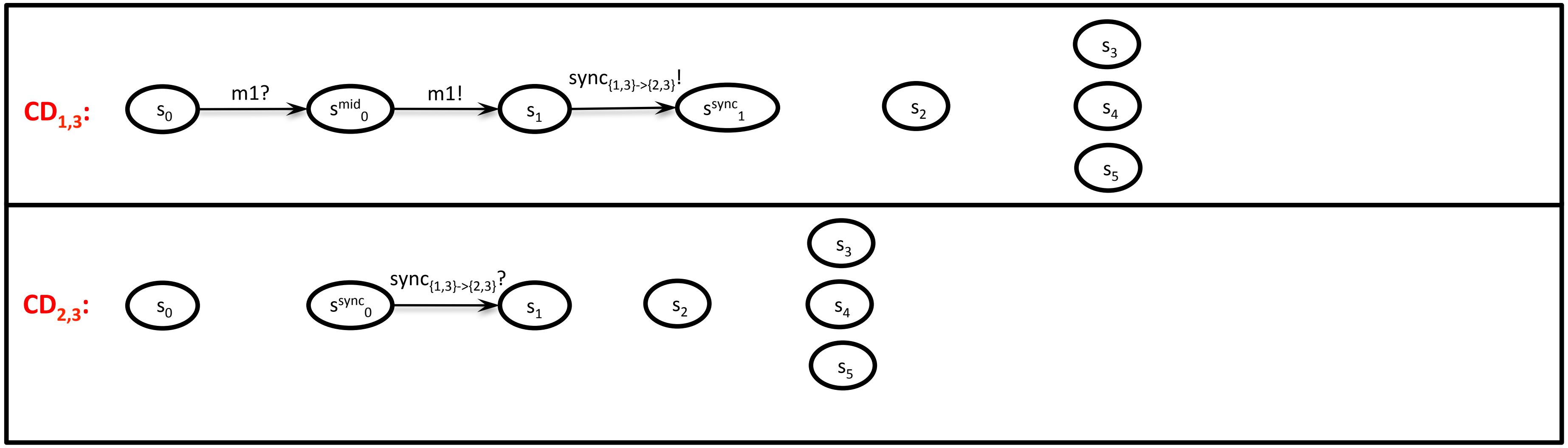
behavioral alternative 1 in context x

behavioral alternative 2 in context y

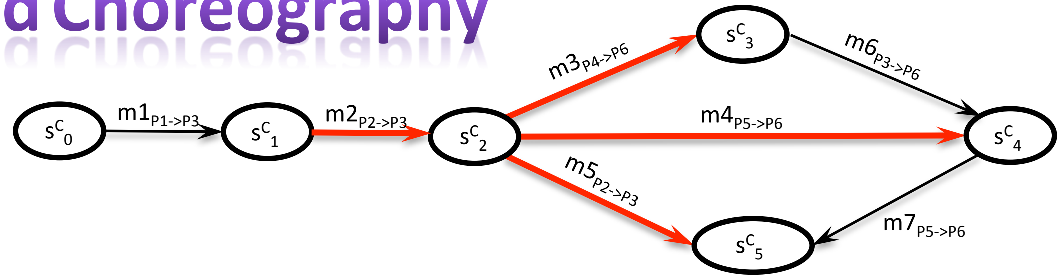
behavioral alternative 3 in context y and context z

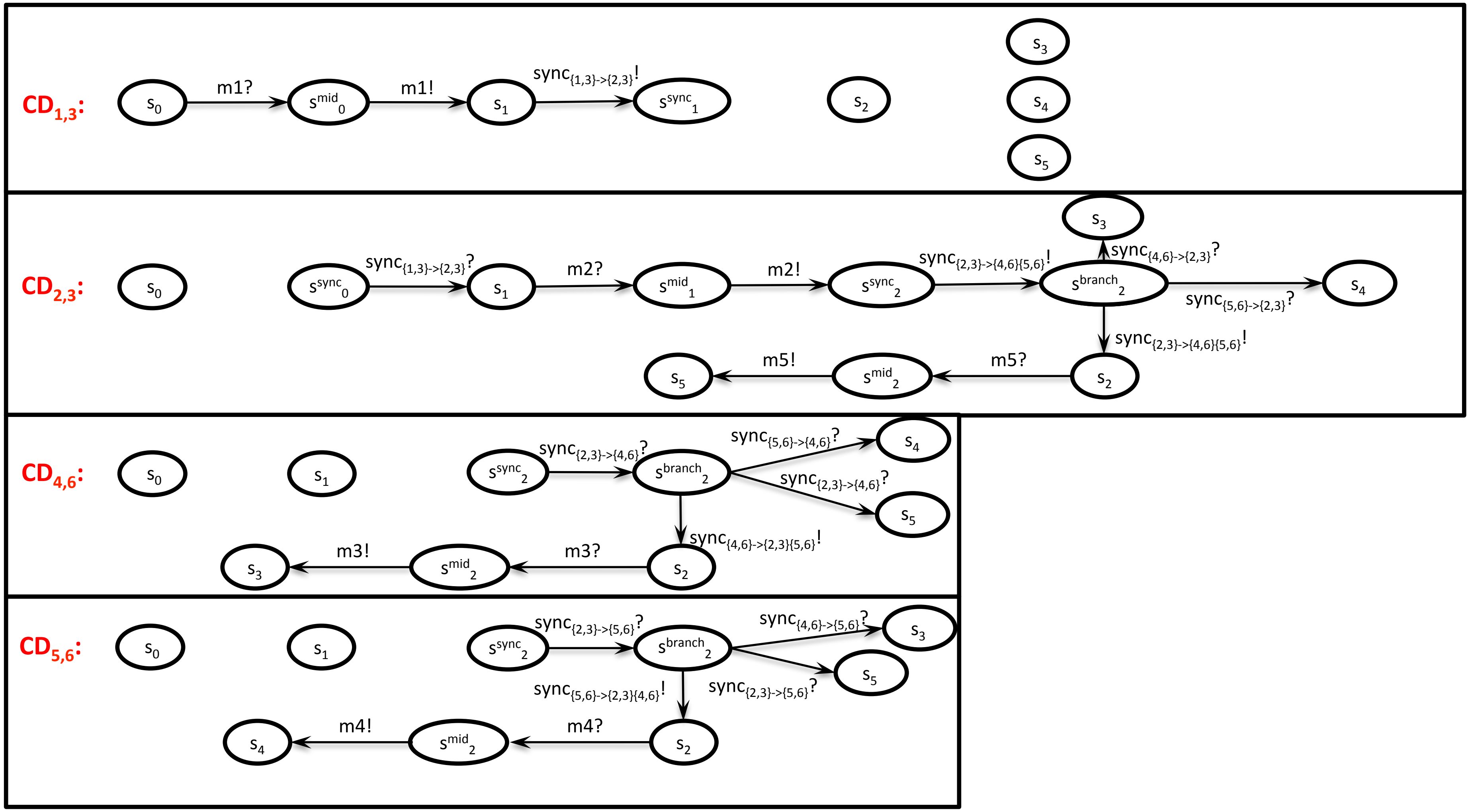
Automata-based Choreography Specification



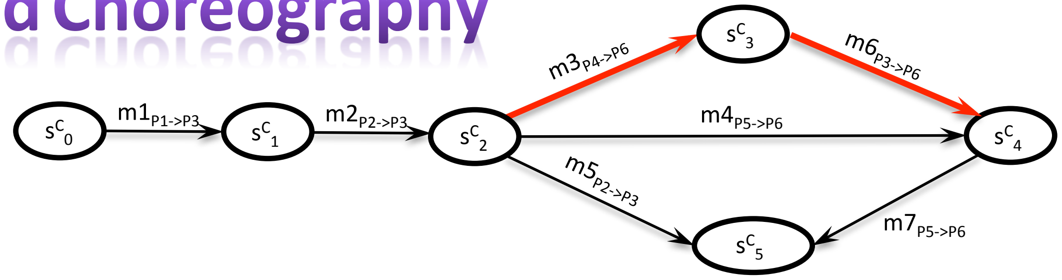


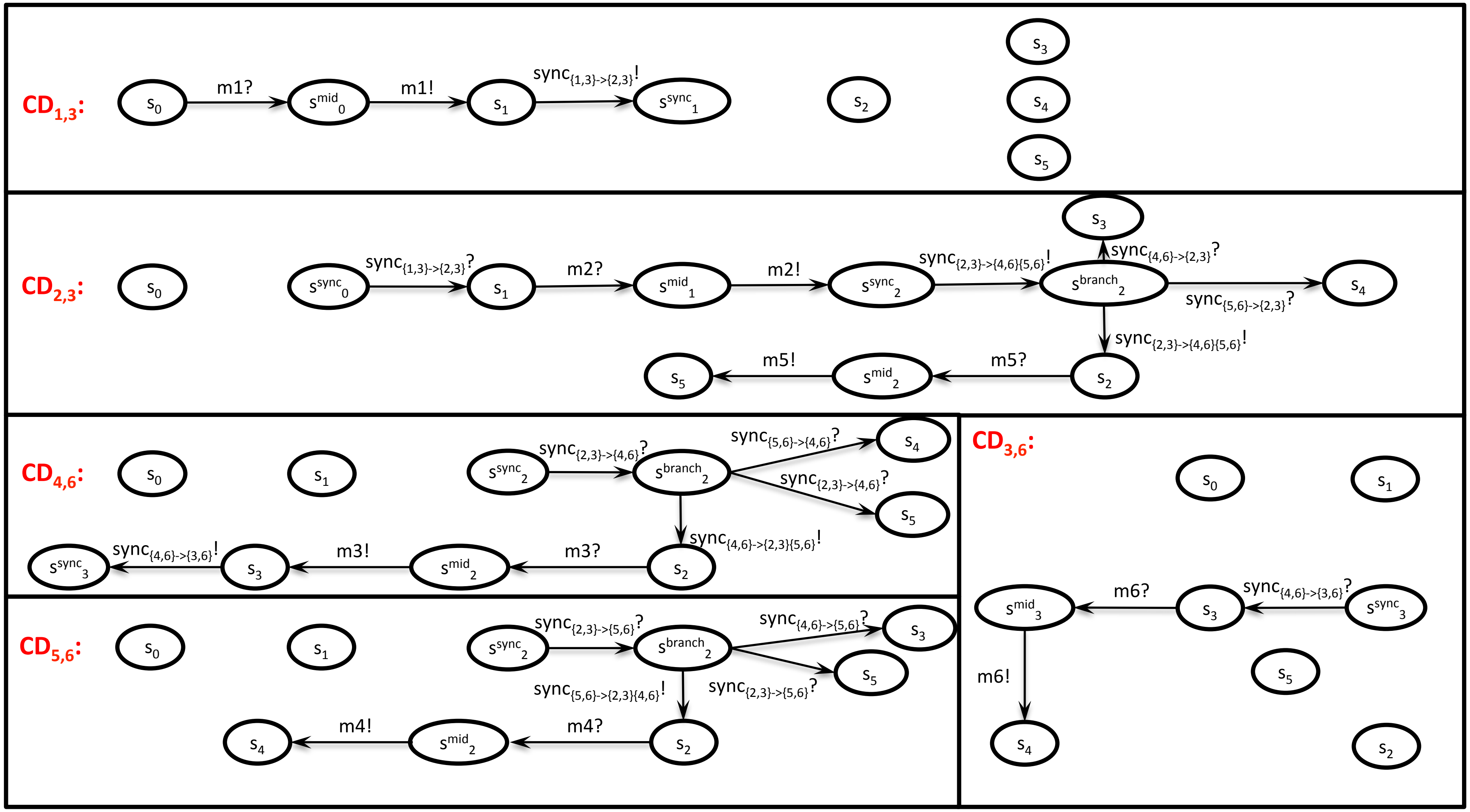
Automata-based Choreography Specification



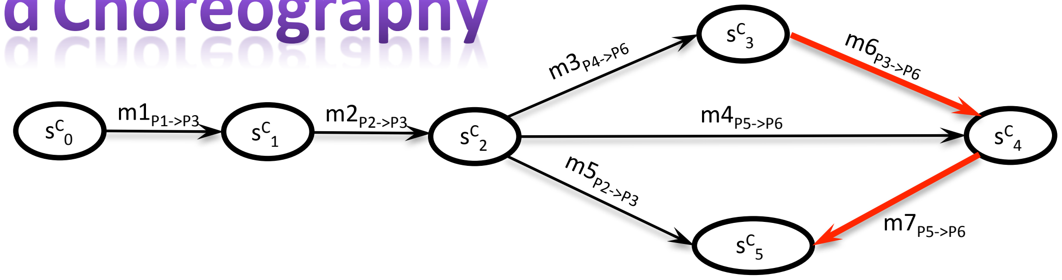


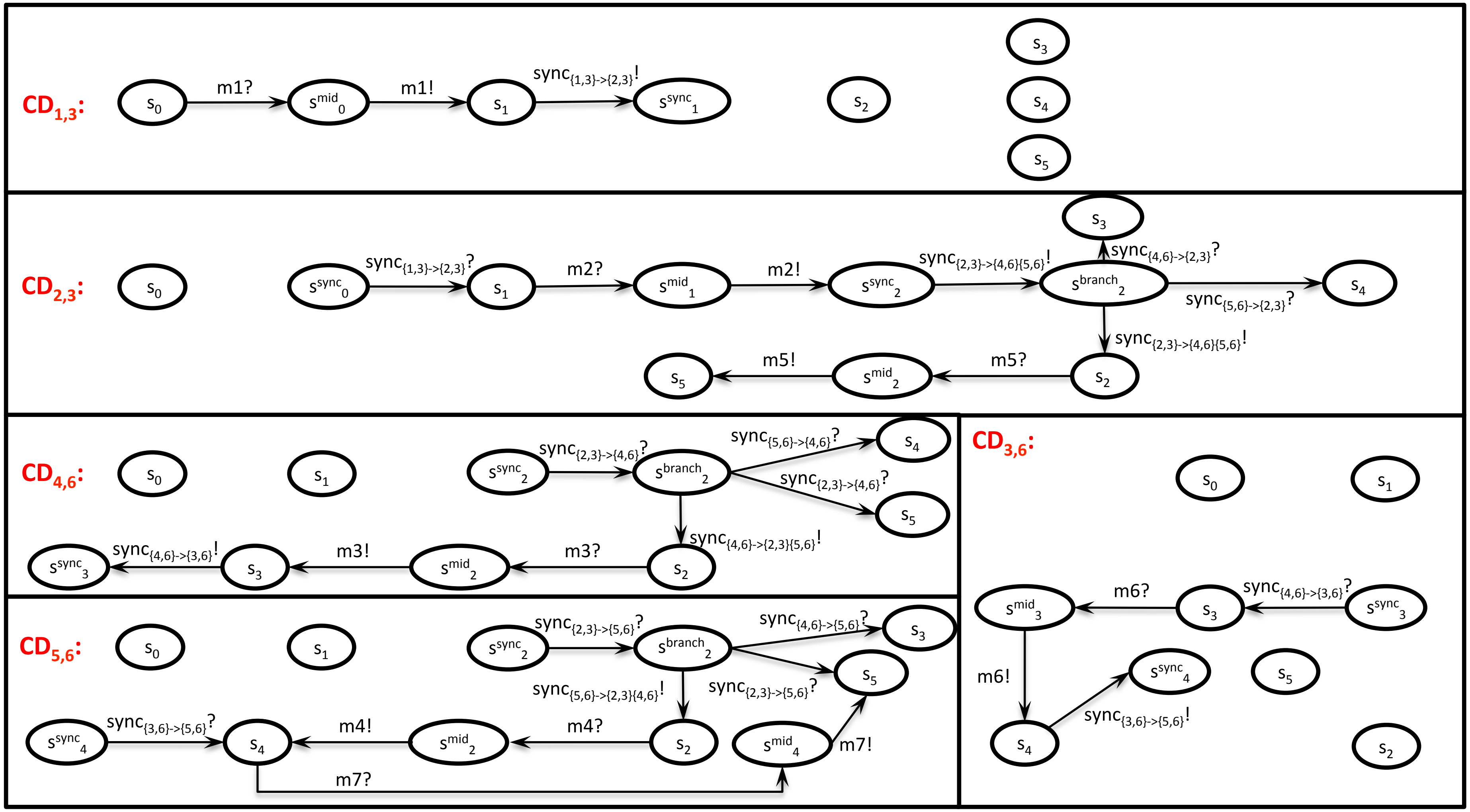
Automata-based Choreography Specification



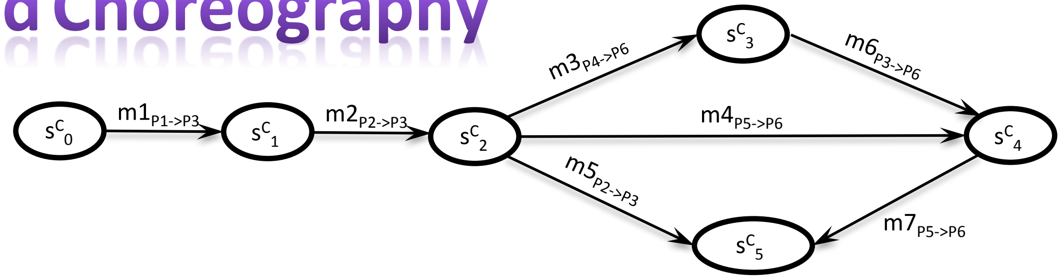


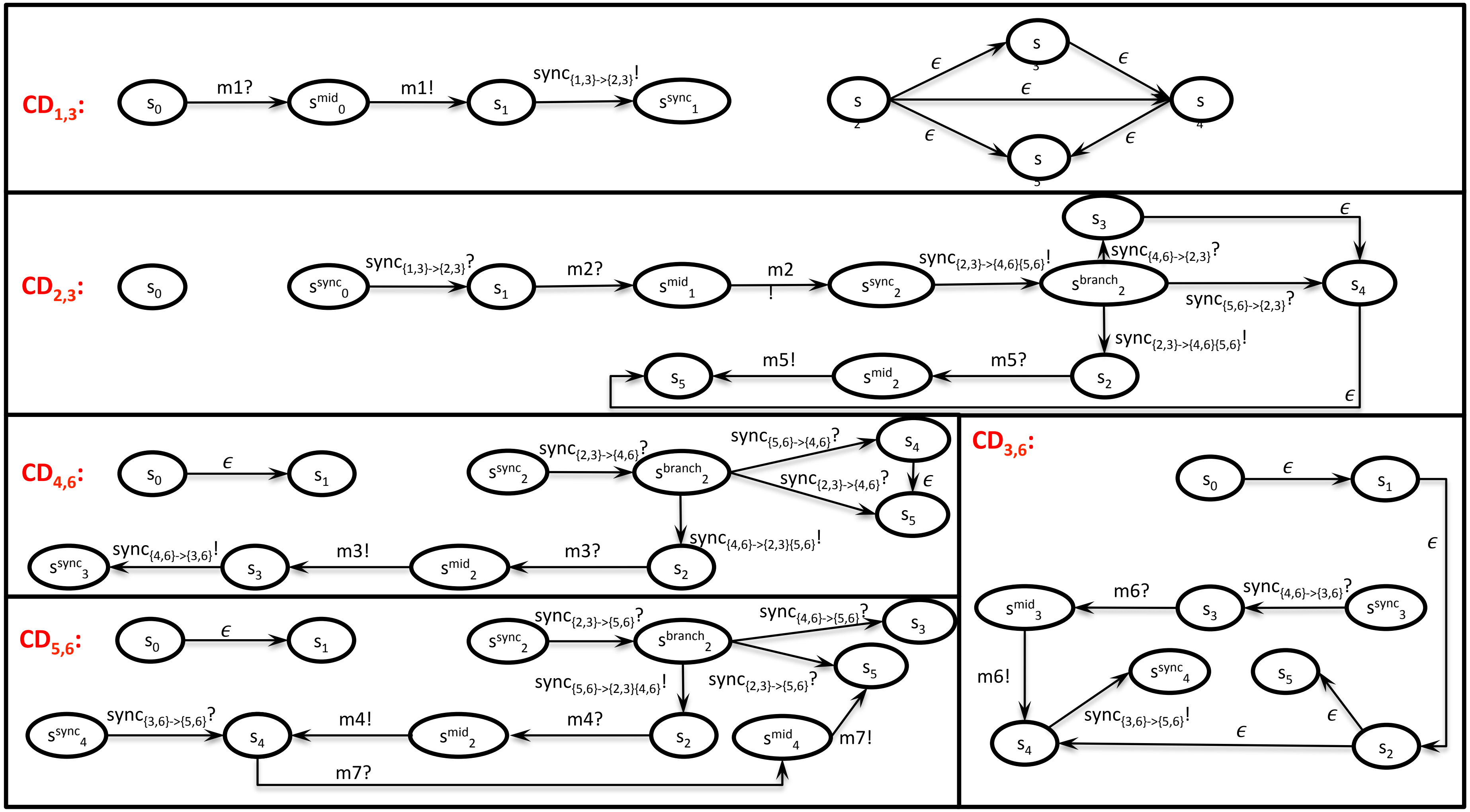
Automata-based Choreography Specification



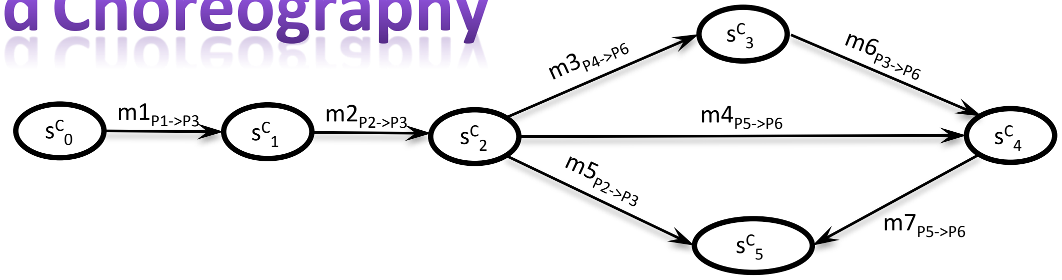


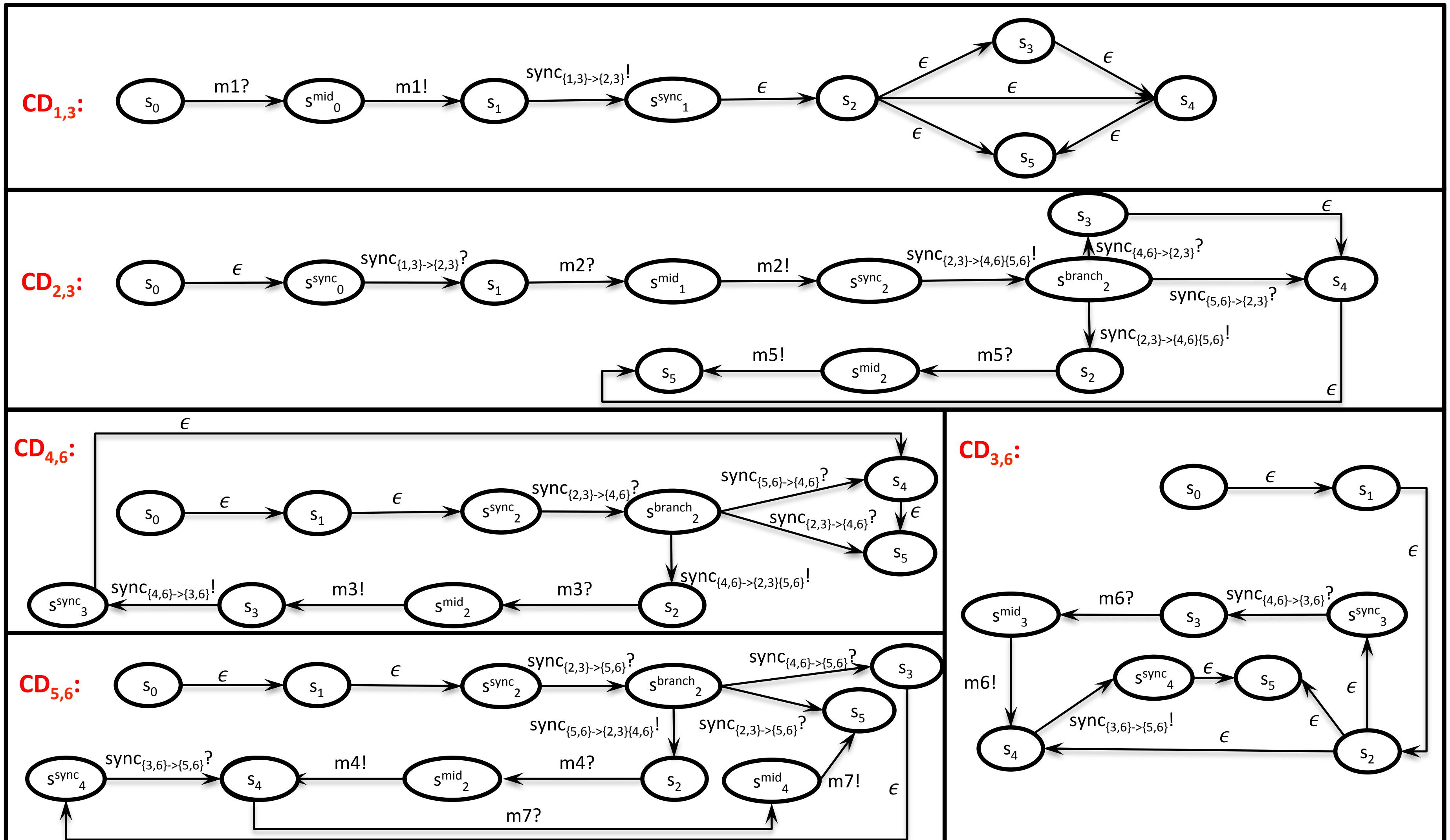
Automata-based Choreography Specification





Automata-based Choreography Specification





Conclusion 1

- (bit of) SA increasingly important to produce “glue” code to develop “correct by construction” composed systems out of heterogeneous third party components. Not only coordination … we can add logic …

- Architectural patterns as a way to give “structure” to the environment, i.e. to constrain the environment, they provide assumptions that need to be guaranteed by components’ behavior and facilitate model mining

- Synthesis as a viable tool based on very realistic assumptions, synthesis can go beyond interactions

- Applied successfully in the choreography domain

Conclusion 2

- A lot of theoretical work to exploit
- From theory to practice
- Synthesis is difficult but …
- … we have demonstrated that it can be practical for software production thanks to
  - Composability
  - Software Architecture
  - applied Formal Verification

Travelling in the digital space with …

- Marco Autili
- Antonia Bertolino
- Massimo Tivoli
- Patrizio Pelliccione
- Romina Spalazzese
- Davide Di Ruscio
- …

http://it.123rf.com/photo_3450020_cute-straniero-in-astronave--colore-illustrazione.html

Some bibliography

- M. Tivoli and P. Inverardi, ***Failure-free coordinators synthesis for component-based architectures*** (2008), in: Science of Computer Programming, 71:3(181-212)
- M. Autili, L. Mostarda, A. Navarra and M. Tivoli, ***Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems*** (2008), in: Journal of Systems and Software, 81:12(2210-2236)
- P. Inverardi and M. Tivoli, **Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns**, in: ICSE 2013.
- A. Di Marco, P. Inverardi, and R. Spalazzese*.* ***Synthesizing Self-Adaptive Connectors meeting Functional and Performance Concerns***. In SEAMS 2013, pp. 133-142, IEEE Press, Piscataway, NJ, USA.
- P. Inverardi, R. Spalazzese and M. Tivoli. ***Application-Layer Connector Synthesis***. Formal Methods for Eternal Networked Software Systems (SFM'11), pages 148-190, Springer-Verlag Berlin Heidelberg, LNCS, volume 6659, 2011.
- Spalazzese R., Inverardi P., Issarny V..***Towards a Formalization of Mediating Connectors for on the Fly Interoperability*****.** In Proceedings of WICSA/ECSA 09. pages 345-348, 2009
- Nicola Nostro, Romina Spalazzese, Felicita Di Giandomenico, Paola Inverardi: Achieving functional and non functional interoperability through synthesized connectors. Journal of Systems and Software 111: 185-199 (2016)
- Marco Autili, Paola Inverardi, Filippo Mignosi, Romina Spalazzese, Massimo Tivoli: **Automated Synthesis of Application-Layer Connectors from Automata-Based Specifications.** LATA 2015: 3-24
- Vittorio Cortellessa, Antinisca Di Marco, Paola Inverardi: **Model-Based Software Performance Analysis.** Springer 2011, ISBN 978-3-642-13620-7, pp. I-XII, 1-190
- Antonia Bertolino, Paola Inverardi, Henry Muccini: **Software architecture-based analysis and testing: a look into achievements and future challenges.** Computing 95(8): 633-648 (2013)

Derivation of Partial Models from Running Systems

- ***Strawberry*** by Bertolino, Inverardi, Pelliccione, Tivoli – ESEC/FSE 2009

  Black-box ***Techniques used*: Syntactic analysis, testing, and synthesis**
- ***GK-Tail*** by Lorenzoli, Mariani, Pezzè – ICSE 2008

  Grey-box ***Techniques used*: Static analysis of execution traces**
- ***SPY*** by Ghezzi, Mocci, Monga – ICSE 2009

  Black-box ***Techniques used*: Dynamic analysis + graph transformation**
- ***Jadet*** by Wasylkowski, Zeller, Lindig – ESEC/FSE 2007 and
- ***Tikanga*** by Wasylkowski, Zeller – ASE journal 2011

  White-box ***Techniques used*: Static program analysis + model checking + concept analysis**
- ***TAUTOKO*** by Dallmeier et al. – TSE 2012

  Black-box ***Techniques used*: Test case generation + dynamic specification mining**
- ***LearnLib*** by Hungar et al.– Test Conference, 2003

  Black-box ***Techniques used*: Invariant detection**
- ***Daikon*** by Ernst et al.– 1999

  Grey-box ***Techniques used*: Active automata learning and experimentation**
- K. Krogmann, M. Kuperberg, and R. Reussner – TSE 2010

  Grey-box ***Techniques used*: Static and dynamic analysis + genetic programming**

Automatic Connector Synthesis to Support Integration

- Automated Synthesis of Service **Choreographies** [Autili, Di Salle, Inverardi, Tivoli, 2009-2015]
- **distributed choreography-based coordination**
- Automated **Connector/Coordinator/Adaptor/Mediator synthesis** [Autili, Inverardi, Spalazzese, Tivoli, 2007-2009 & 2011-2013]
- **centralized vs distributed coordinators, modular connectors, heterogeneous protocols mediation,**
- **full automation**
- Formal approaches to ***protocol conversion*** [Calvert, Lam 1990 & Lam 1998]
- Specification of ***protocol adapters*** [Yellin, Strom 1997]

seminal work not focused on sophisticated mediation logics, e.g., message reordering or different granularity of protocol languages

- Automatic ***mediation of business processes*** [Vaculin et al. 2007 & 2008]

focus on the semantic web service domain, no formal characterization

- Connector wrappers as ***protocol transformations*** [Garlan et al. 2003]
- Algebra of ***stateless connectors*** [Bruni, Lanese, Montanari 2006]

support for modularity, the focus is on connector design and specification => no automation

- ***Converter synthesis*** [Passerone et al. 2002]

they assume an "inconsistency-free" specification of the converter

- Generation of ***component adapters*** [Canal, Poizat, Salaün 2008]

it requires to know many implementation details about the adaptation