

# Tutorial: A Design for Adaptation Framework for Self-Adaptive Systems

***Martina De Sanctis***

*msanctis@fbk.eu*

International Conference on Self-Adaptive  
and Self-Organizing Systems - SASO 2018

Trento, Italy

# About me

- 2012: **Master's degree** in Computer Science at the University of L'Aquila
  - Thesis: *"Performance antipatterns modeling in the Architectural Description Language Æmilia"*.
  - Supervisor: Prof. Vittorio Cortellessa
- 2013 – 2017: **Ph.D.** in Computer Science at the Doctoral School in Information and Communication Technology, University of Trento and Fondazione Bruno Kessler (FBK)
  - Thesis: *"Dynamic Adaptation of Service-Based Systems: a Design for Adaptation Framework."*
  - Supervisors: Dr. Marco Pistore & Dr. Antonio Bucchiarone

# About me

- 2017 - 2018: **PostDoc** researcher at the Distributed Adaptive Systems (DAS) unit at Fondazione Bruno Kessler
  - Research focus on the application of a design for adaptation approach to Internet of Things domains.
- 2018 – 2020: **PostDoc** researcher at the Gran Sasso Science Institute (GSSI)
  - An international PhD school and a center for advanced studies in physics, mathematics, computer science and social sciences.

# Objectives of this tutorial

- Understand features of socio-technical systems and the environment in which they operate;
- Get familiar with self-adaptive service-based systems and (part of) the state of the art approaches for engineering them;
- Get insight in the gap between the systems *design* and their *self-adaptive operation* and understand how a design for adaptation can help to reduce it;

# Outline of the tutorial

- Overview on socio-technical systems
  - A motivating scenario
- Engineering socio-technical service-based systems
- Dynamic adaptation approaches
- Design for adaptation

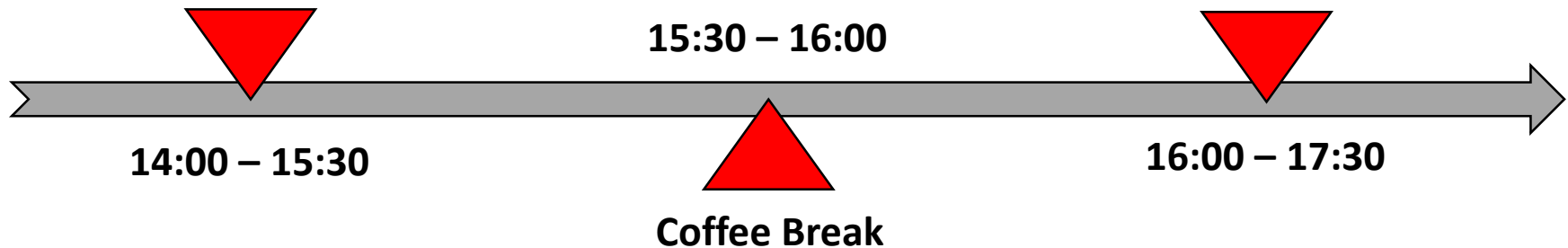
# Tutorial Timeline

## Part I:

- Context
- State of the art
- Design4adaptation

## Part II:

- Demo
- Tool support
- Scenarios



# Outline of the tutorial

- *Overview on socio-technical systems*
  - A motivating scenario
- Engineering socio-technical service-based systems
- Dynamic adaptation approaches
- Design for adaptation

# Socio-Technical Systems

“The term *socio-technical systems* was originally coined by Emery and Trist (1960) to describe systems that involve a complex interaction between *humans, machines* and the *environmental aspects* of the work system.”

- Nowadays, this interaction is true of most enterprise systems...
- ...but still these factors—*people, machines* and *context*—need to be considered when developing such systems.



# Socio-Technical Systems

Widely used to describe many complex systems, but there are *five key characteristics* of open socio-technical systems (*Badham et al., Socio-technical theory. 2000*):

- Systems should have interdependent parts.
- Systems should adapt to and pursue goals in external environments.
- Systems have an internal environment comprising separate but interdependent technical and social subsystems.
- Systems have equifinality. In other words, systems goals can be achieved by more than one means. This implies design choices to be made.
- System performance relies on the joint optimization of the technical and social subsystems.

Gordon Baxter, Ian Sommerville. *Socio-technical systems: From design methods to systems engineering*. Interacting with Computers, Volume 23, Issue 1, 2011.

# Socio-Technical Systems

- Problem: techno-centric approaches to systems design do not properly consider the complex relationships between the organisation, the people enacting business processes and the system supporting these processes.
- Solution: a pragmatic approach to the *engineering of socio-technical systems* based on the gradual introduction of socio-technical considerations into existing software development processes.
- Research goal: develop the field of *socio-technical systems engineering* (STSE), the systematic and constructive use of socio-technical principles and methods in the *procurement, specification, design, testing, evaluation, operation* and *evolution* of complex systems.

Gordon Baxter, Ian Sommerville. *Socio-technical systems: From design methods to systems engineering*. Interacting with Computers, Volume 23, Issue 1, 2011.

# Socio-Technical Systems

- Problem: techno-centric approaches to systems design do not properly consider the complex relationships between the organisation, the people enacting business processes and the system supporting these processes.

**To make an impact on practical systems engineering, social-technical factors have to be considered at all stages of the system life-cycle!**

- Research goal: develop the field of *socio-technical systems engineering* (STSE), the systematic and constructive use of socio-technical principles and methods in the *procurement, specification, design, testing, evaluation, operation* and *evolution* of complex systems.

Gordon Baxter, Ian Sommerville. *Socio-technical systems: From design methods to systems engineering*. Interacting with Computers, Volume 23, Issue 1, 2011.

# Socio-Technical Systems

- Modern societies are composed of distributed, heterogeneous and autonomous entities interacting with and within their environment.
  - Entities can be services and service providers, IoT devices, software and humans.
  - Application domains can be smart-cities, smart-construction, smart surveillance systems etc.
- The environment in which these systems operate is ***open*** and ***highly dynamic***:
  - ***System-level changes***: entities dynamically change their behavior and they can join/leave the system.
  - ***Context-level changes***: changes in the environment in which entities operate.

# Examples of Socio-Technical Systems

## CYBER-PHYSICAL SYSTEMS



## INTEGRATED URBAN MOBILITY



## CAR LOGISTIC



## CLINICAL HEALTHCARE



???

# Socio-technical Service-based Systems

- *Service-based systems (SBS)* allow the combination of numerous services into business applications implementing a business logic provided by the collaborations among the involved services.
- They are the result of the advances in *Service Oriented Computing* and the standardization of the *Web Services* technology.
- Modern service-based systems are *highly complex* (e.g., consist of diverse distributed heterogeneous components) and they must operate under *dynamic conditions*. Users are always more involved and increasingly proactive. SBS actually form *socio-technical systems*.



# Service-Oriented Computing

“*Service-Oriented Computing (SOC)* represents a distributed computing platform. As such, it encompasses many things, including its own *design paradigm* and *design principles*, *design pattern* catalogs, *pattern languages*, a distinct *architectural model*, and related concepts, technologies, and frameworks.”

*Elements* of a typical service-oriented computing platform are:

- Service-Oriented Architecture (SOA)
- Services
- Service Composition

Thomas Erl. *SOA Principles of Service Design*. The Prentice Hall Service Oriented Computing Series. Prentice Hall, 2007.

# Service-Oriented Architecture

“SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing.”

Thomas Erl. *SOA Principles of Service Design*. The Prentice Hall Service Oriented Computing Series. Prentice Hall, 2007.

“The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be *invoked*, *published* and *discovered*, and are abstracted away from the implementation.”

“*Understanding service-oriented architecture*” at <https://msdn.microsoft.com/en-us/library/aa480021.aspx> Retrieved 2018-07-19



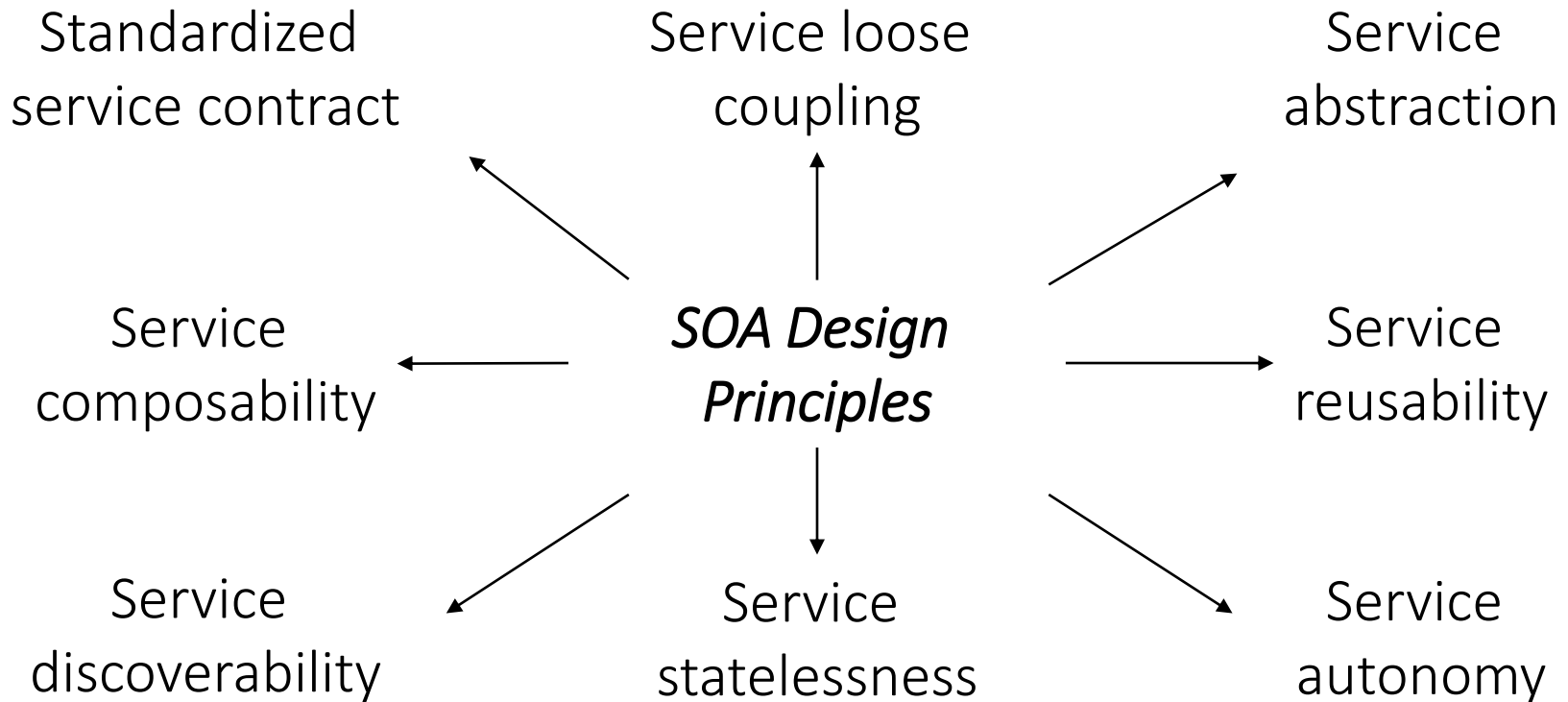
# SOA Design Principles

---

- A design principle is an accepted design guideline or practice that, when applied, results in the realization of specific design characteristics.
  - A design paradigm represents a set of complementary design principles that are collectively applied in support of common goals.
  - A design pattern identifies a common problem and provides a recommended solution.
  - A design standard is a convention internal and specific to an enterprise that may or may not be derived from a design principle or pattern.
- 

Thomas Erl. *SOA Principles of Service Design*. The Prentice Hall Service Oriented Computing Series. Prentice Hall, 2007.

# SOA Design Principles



Follow these principles means to realize *flexible* service-based applications, by significantly decreasing their development cost and further maintenance support.

# Services

“*Services* exist as physically independent software programs [...]. Each service is assigned its own distinct *functional context* and is comprised of a *set of capabilities* related to this context. Those capabilities suitable for *invocation by external consumer* programs are commonly expressed via a *published service contract* (much like a traditional API).”

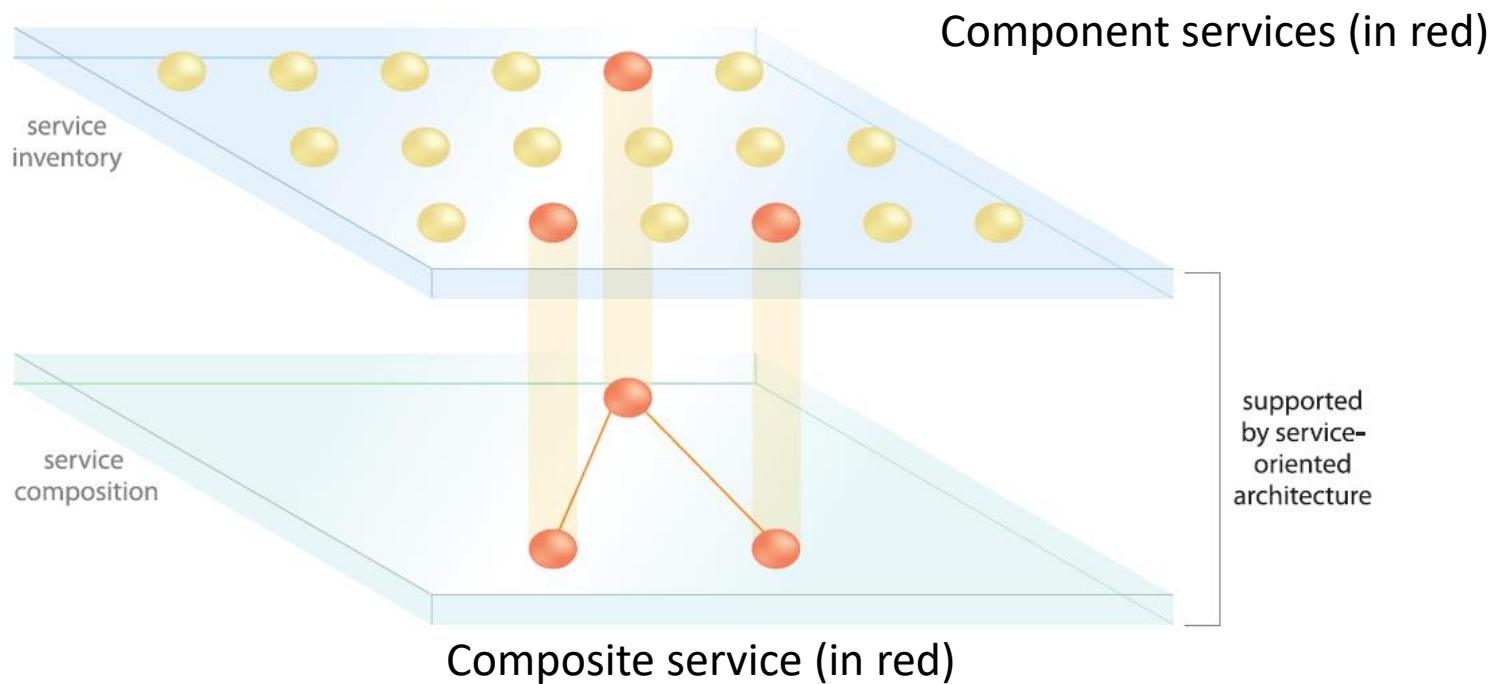


Example of entity service.

Thomas Erl. ***SOA Principles of Service Design***. The Prentice Hall Service Oriented Computing Series. Prentice Hall, 2007.

# Service Composition

“A *service composition* is a coordinated aggregate of services that have been assembled to provide the functionality required to automate a specific business task or process.”

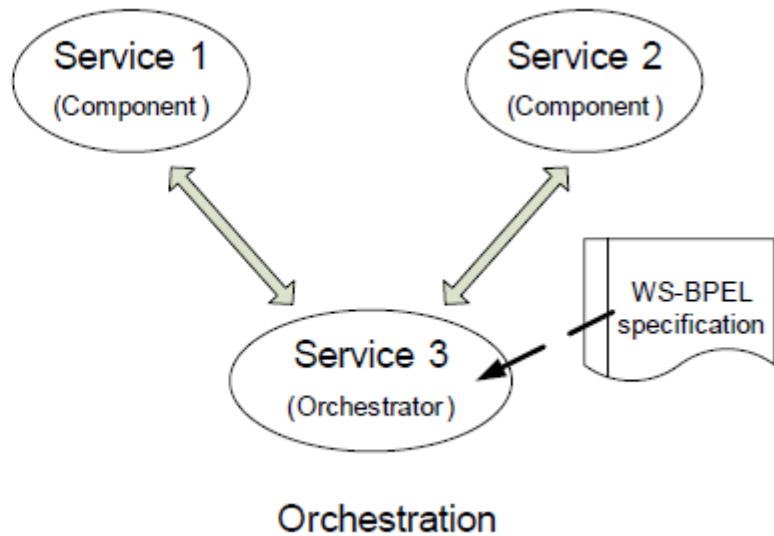
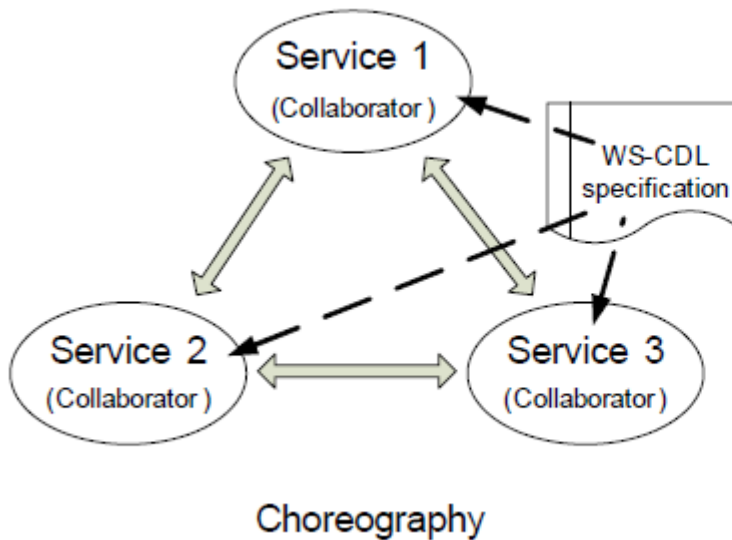


Thomas Erl. *SOA Principles of Service Design*. The Prentice Hall Service Oriented Computing Series. Prentice Hall, 2007.

# Service Composition

There exist two conventional ways to compose services:

- Orchestration
- Choreography



# Outline of the tutorial

- Overview on socio-technical systems
  - *A motivating scenario*
- Engineering socio-technical service-based systems
- Dynamic adaptation approaches
- Design for adaptation

# A Motivating Scenario

- So far we have discussed:
  - *socio-technical systems &*
  - *service-based systems.*
- Now we are going to see a scenario about a *smart mobility system* as an example of a *socio-technical service-based system*.



# Smart Mobility System Scenario



BAHN





# The Problem



JOURNEY PLANNERS



SHARING MOBILITY SERVICES



PUBLIC MOBILITY SERVICES



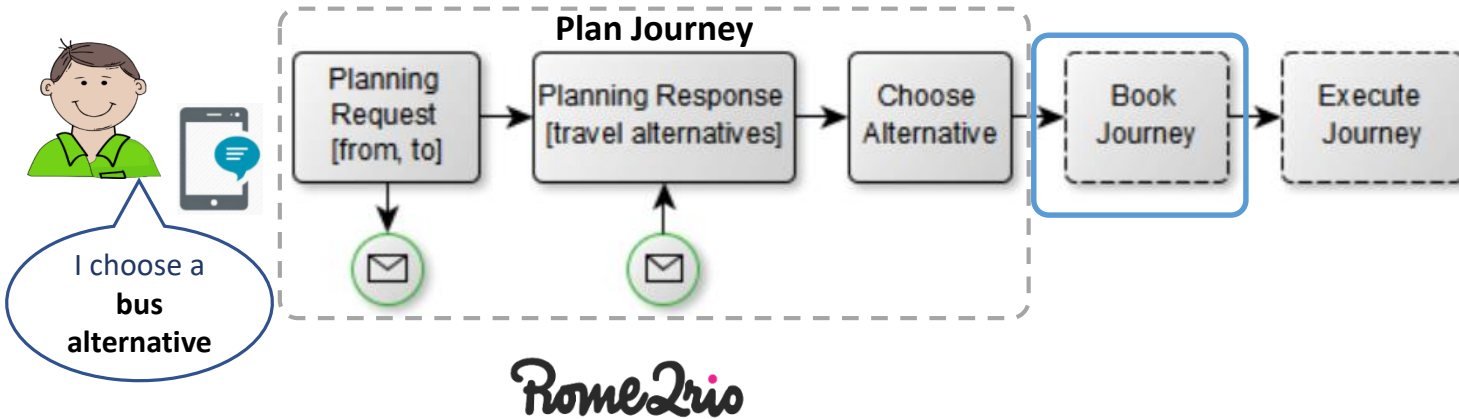
OTHER SERVICES

# Our Vision

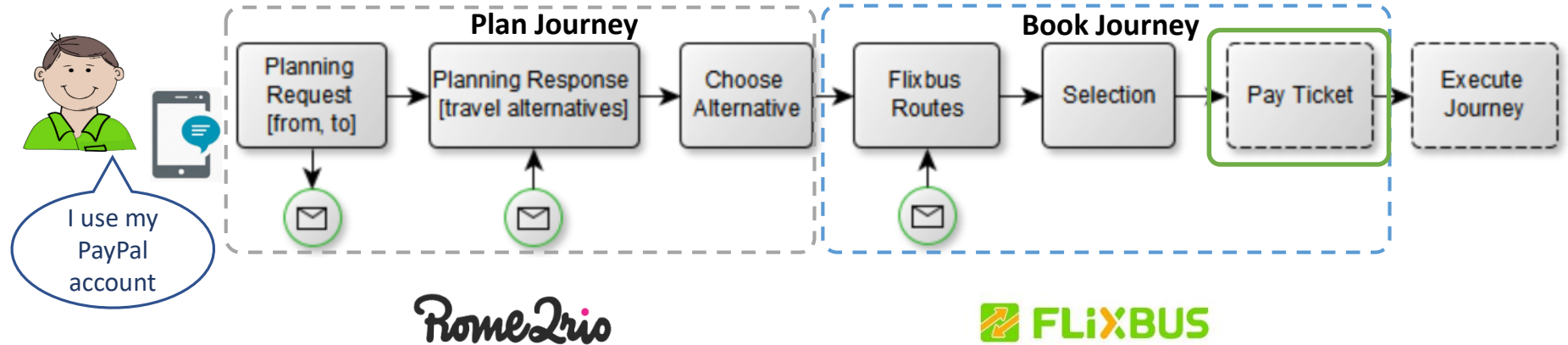
## TRAVEL ASSISTANT



# Our Vision



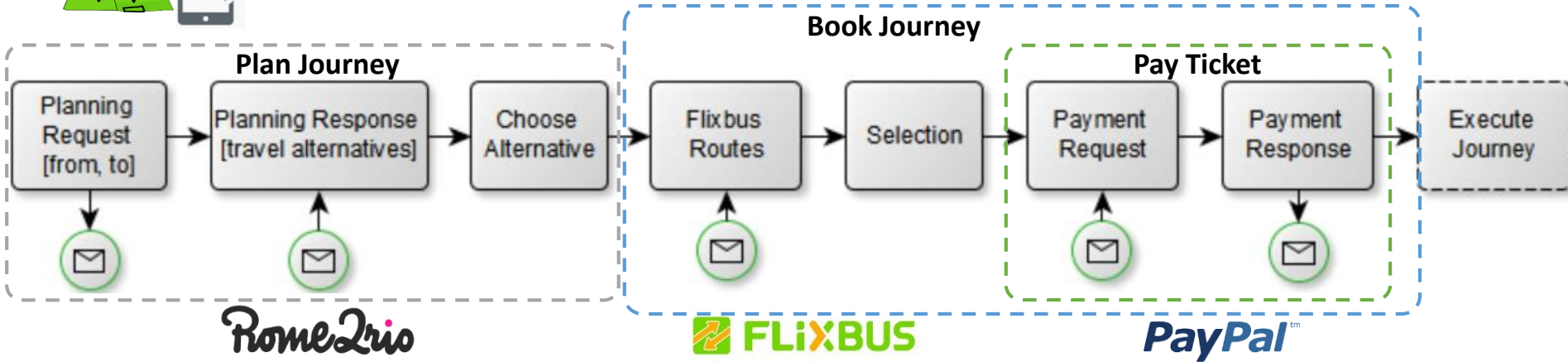
# Our Vision



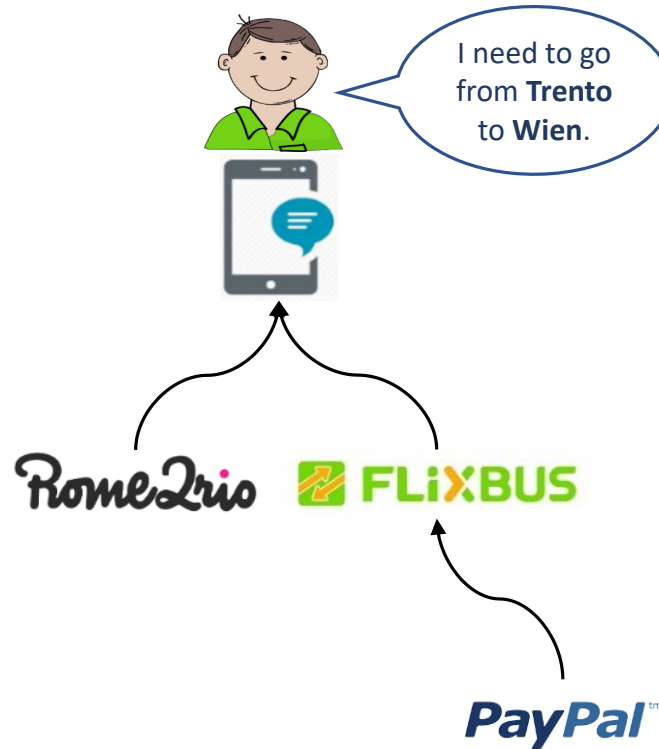
# Our Vision



I need to go from **Trento** to **Wien**.



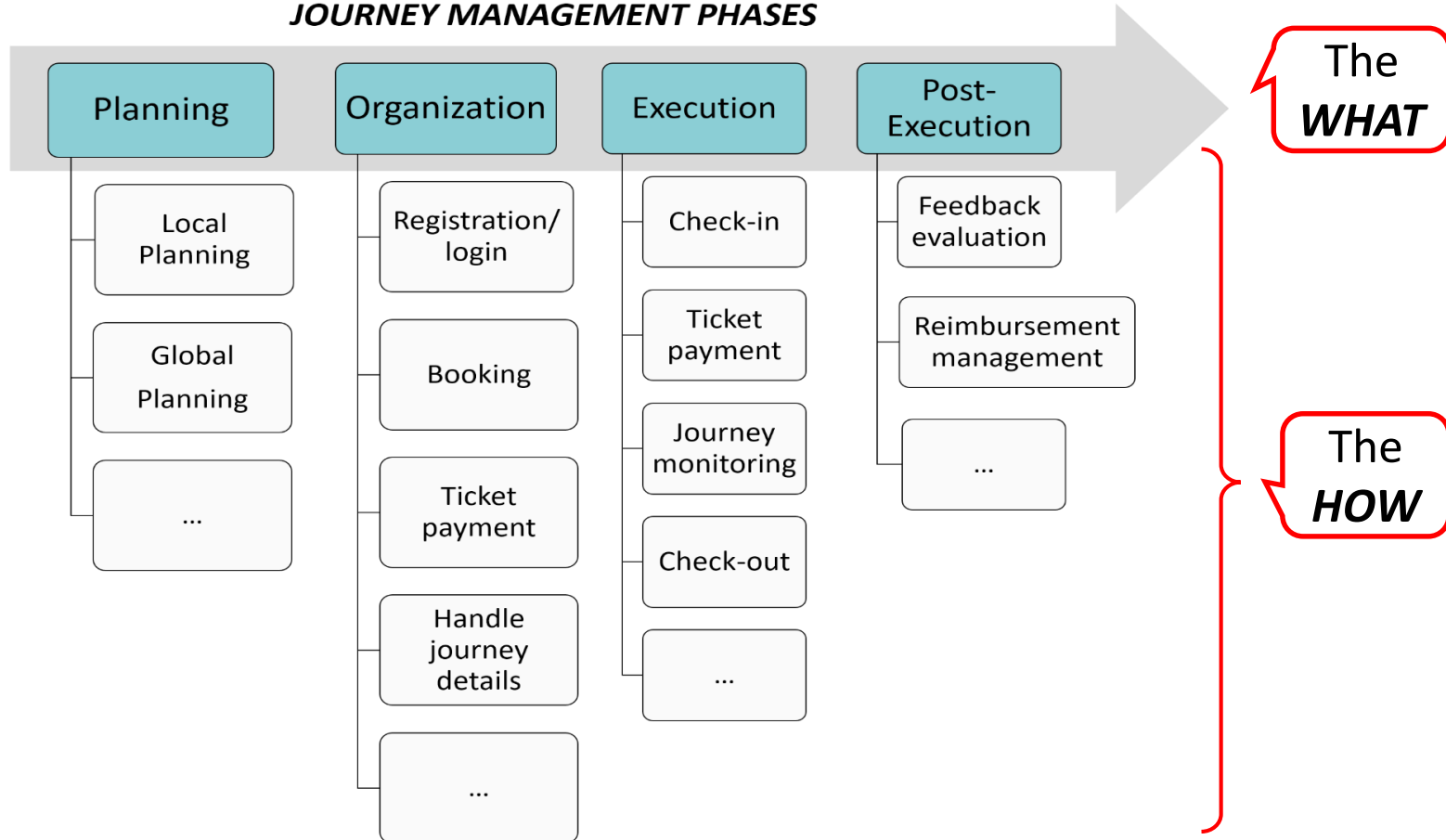
# Our Vision



# A Travel Assistant



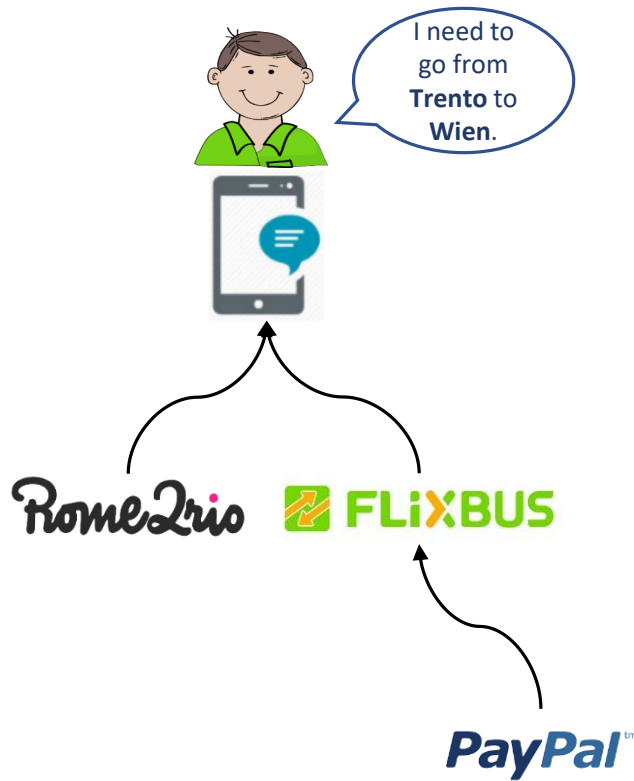
## JOURNEY MANAGEMENT PHASES



BAHN



# Requirements



- Specify the **WHAT**, at design time, while leaving the **HOW** to the runtime.
- Keep separate the *application logic* and its *context-aware configuration*.
- Provide support for the runtime and *automatic services selection* and *composition*.



# Requirements

- ***Heterogeneity & autonomy awareness***: The system must consider the heterogeneity of the services in terms of technologies or offered functionalities (e.g., REST API vs. SOAP, online booking vs. on board ticketing);
- ***Openness awareness***: The system must be capable to operate in open environments with continuously entering and leaving services, which are not known a priori (e.g., a new ride-sharing service is available in the city);
- ***Interoperability***: The system must be capable to propose complex solutions taking advantages of the variety of services (e.g., a user needs of a unique solution with her booking, payment, train journey, and taxi ride);
- ***Customizability***: The system must provide users with personalized solutions (e.g., a wheelchair user has preferences on transportation means and stops);

# Requirements

- ***Context awareness***: The system must take into account the state of the environment (e.g., strikes, bad weather, roadworks);
- ***Adaptivity***: The system must be able to *react* and *adapt* to changes in the environment that might occur and affect its operations (e.g., a strike affects the user's train journey and the system offers a new journey plan);
- ***Information accuracy***: The system must provide up to date and reliable information and solutions (e.g., temporary changes on a bus route);
- ***Portability***: The system must be deployable in different environments without an ad-hoc reconfiguration from the developers (e.g., the travel assistant must be usable in Trento as well as in Paris).

# Outline of the tutorial

- Overview on socio-technical systems
  - A motivating scenario
- *Engineering socio-technical service-based systems*
- Dynamic adaptation approaches
- Design for adaptation

# Engineering socio-technical service-based systems

- We review approaches whose aim is, among the others, to increase and support the flexibility and dynamicity of the modeled systems, both supporting adaptation.
  - Software Product Lines
  - Service Mashups
  - Microservices
  - Cloud Services
  - Open Services

# Software Product Lines

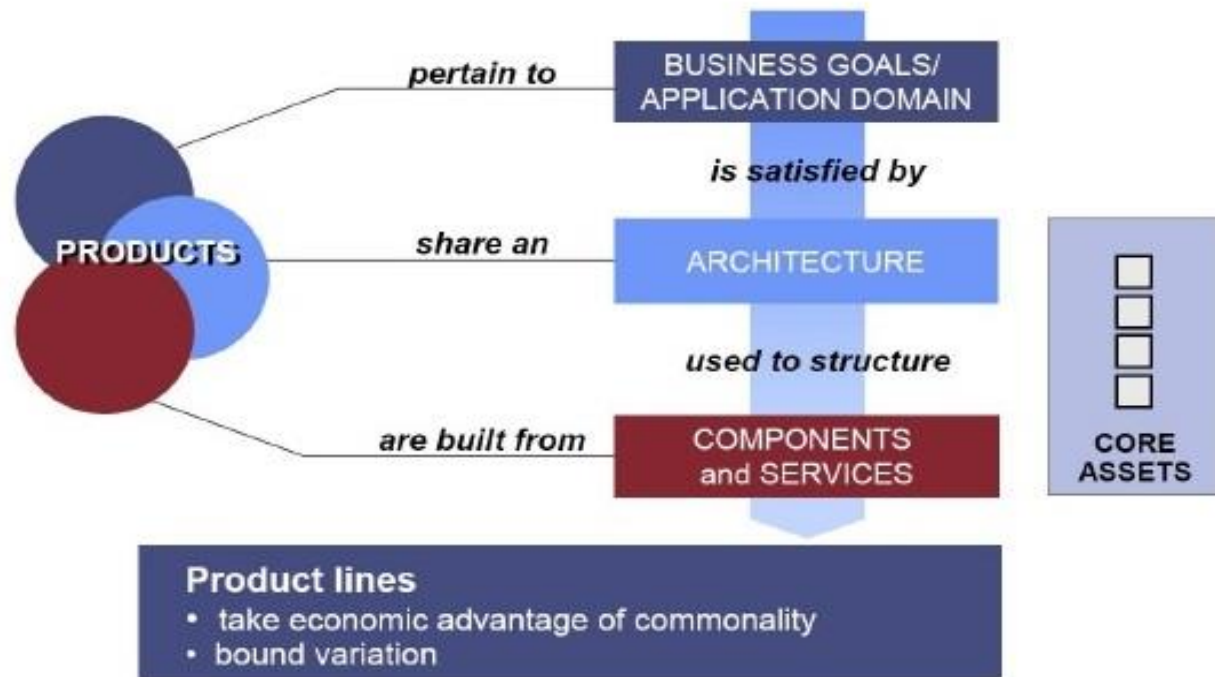
“A *Software Product Line (SPL)* is a set of software-intensive systems that share a *common, managed set of features* satisfying the specific needs of a particular market segment or mission and that are developed from a *common set of core assets* in a prescribed way.”

Software Product Lines *Carnegie Mellon Software Engineering Institute* Web Site.

- The designer analyzes a *software family* as a whole and establishes the common and *reusable assets* that form its basic platform, as well as the possible *application-specific customizations*.

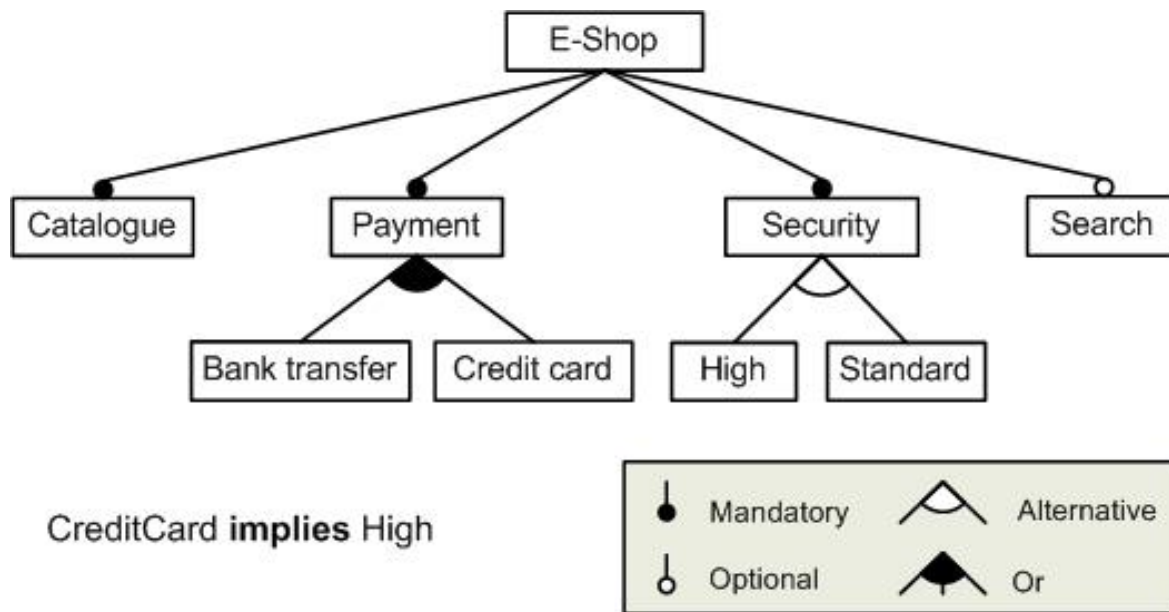
# Software Product Lines

- *Software product line development* represents a software engineering approach, with its *methods, tools* and *techniques*, for engineering software systems.
- It supports a *strategic reuse*, by exploring *commonality* and *variability*.



# Feature Models

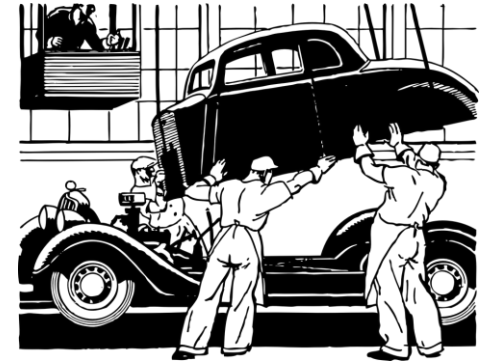
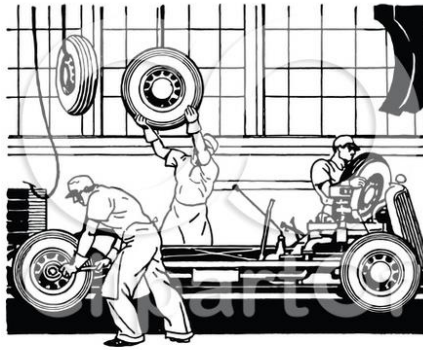
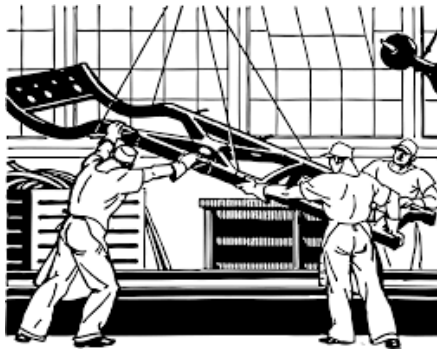
- A *feature model* species the alternative variations and constraints that can exist for each feature.



A feature diagram representing a configurable e-shop system.

# Software Product Lines

- SPL can be applied in different domains.
- automotive manufacturers can create unique variations of one car model using a single pool of carefully designed parts and a factory specifically designed to configure and assemble those parts.



- Other examples are: smart spaces, robotic applications, etc



# Mashups

“From the Human–Computer Interaction perspective, *mashup* refers to a composition of contents and/or features from several sources that determine new client-side interactive applications (generally hosted online). In general, Web mashups can combine data, presentations and functionalities from different Web sites into a single, novel, Web application.”

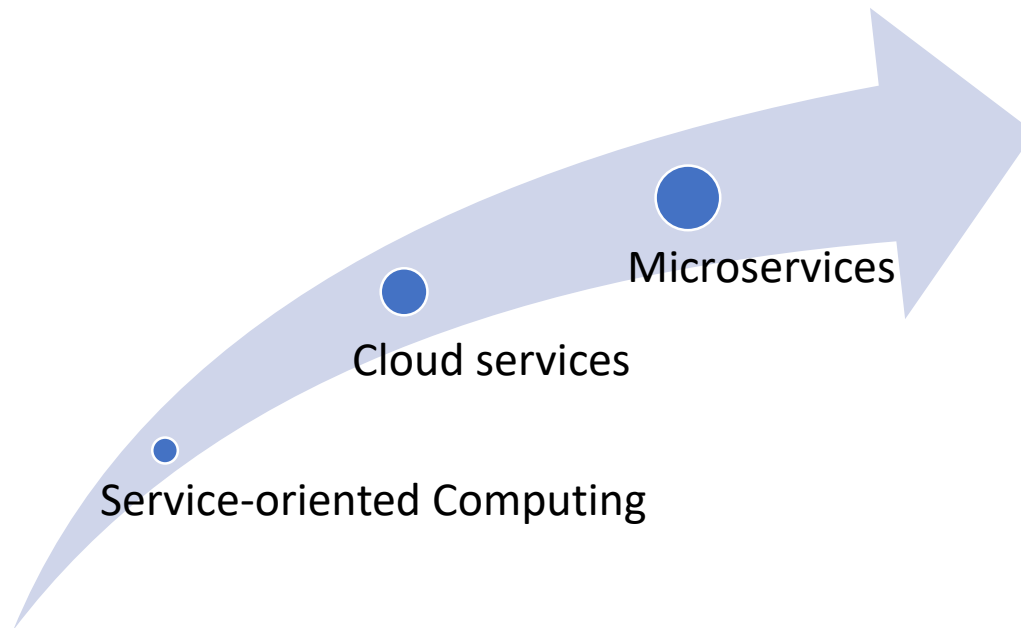
Giuseppe Ghiani, Fabio Paternò, Lucio Davide Spano, Giuliano Pintori. *An environment for End-User Development of Web mashups*. In Int. J. Human-Computer Studies – Elsevier, 2016.

# Service Mashups

- A *service mashup* is, simply, a value-added service build from existing services.
  - Christoph Dorn, Daniel Schall, and Schahram Dustdar. *Context-aware adaptive service mashups*. In 4th IEEE APSCC 2009.
  - Mahdi Bashari, Ebrahim Bagheri, and Weichang Du. *Automated composition of service mashups through software product line engineering*. In ICSR 2016.
  - Andrei Ciortea, Olivier Boissier, Antoine Zimmermann, and Adina Magda Florea. *Responsive decentralized composition of service mashups for the internet of things*. In IOT 2016.
- Approaches enabling end-users to create by themselves mashups (e.g., IoT mashups), by using *event-conditions-actions* (ECA) rules.
  - IFTTT – *If This, Then That* service (<https://ifttt.com/>)

# Microservices

- Triggered by the need for moving from *monolith applications* to *distributed applications*.
- An *evolution* or an *implementation* (?) of SOA.



# Microservices

“A *microservice* is a minimal independent process interacting via messages. A microservice architecture is a distributed application where all its modules are microservices.”

“...the *microservice architectural style* is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”

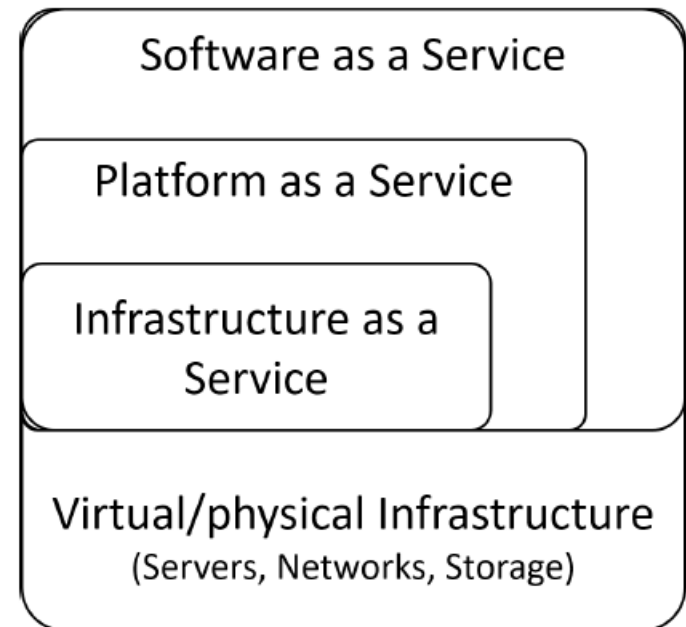
- Martin Fowler

# Cloud Computing

“Any computing environment in which computing, networking, and storage resources can be provisioned and released elastically in an on-demand, self-service manner.”

From “Migrating to Cloud Native Applications Architectures” by Matt Stine

- Strongly support the engineering and delivery of service-based systems.
- It plays a central role in the development of the future Internet of Services.
- It emphasizes the role of services, by enabling the *everything-as-a-service paradigm*.



# Open Services

- Easy to understand and to access services that can be exploited to develop applications or can be improved to provide new value-added services on top of them.
- Mainly specified through Web APIs.
- Organizations are building on these trends by providing specific *API management platform*.
  - *Programmable Web* has now more than 10,000 API in its directory (<http://www.programmableweb.com>).

# Outline of the tutorial

- Overview on socio-technical systems
  - A motivating scenario
- Engineering socio-technical service-based systems
- *Dynamic adaptation approaches*
- Design for adaptation

# Towards Self-adaptation

Modern service-based systems are *highly complex* and they must operate under *dynamic conditions*.

- Consist of diverse *distributed heterogeneous components*
  - e.g., existing IT systems, smart objects & devices, end-user applications.
- Components are *autonomous* but need to collaborate
  - each component has its own *implementation logic, objectives, requirements* and *local (partial) knowledge*.
  - each component exposes functionalities to the outside world to enable collaboration.
- The systems operate in *highly dynamic environment*
  - *System-level changes*: components dynamically change their behavior (including offered functionalities) and join/leave the system
  - *Context-level changes*: changes in the environment in which components operate



# Challenges

- *Context-awareness* (entities act according to their surrounding) in *volatile context* (unpredictable dynamic changes in the context).
- *Dynamic execution environment* (the set of potential collaborators is dynamically changing).
- *Customizable procedures* (similar entities, e.g. public bus operators of different company, generally follow the same procedure, that may differ in details).

# Self-adaptation

- *Self-adaptation* is a promising approach for managing the *uncertainty* given by the dynamic factors affecting modern systems.
- It is based on the idea of allowing systems to get knowledge about their surrounding and their context at runtime and then reasoning about them and adapt accordingly always by pursuing their goals.

# Self-adaptation

Self-adaptation is one of the main concerns in the context of the *Internet of Services*. It has been studied in different areas:

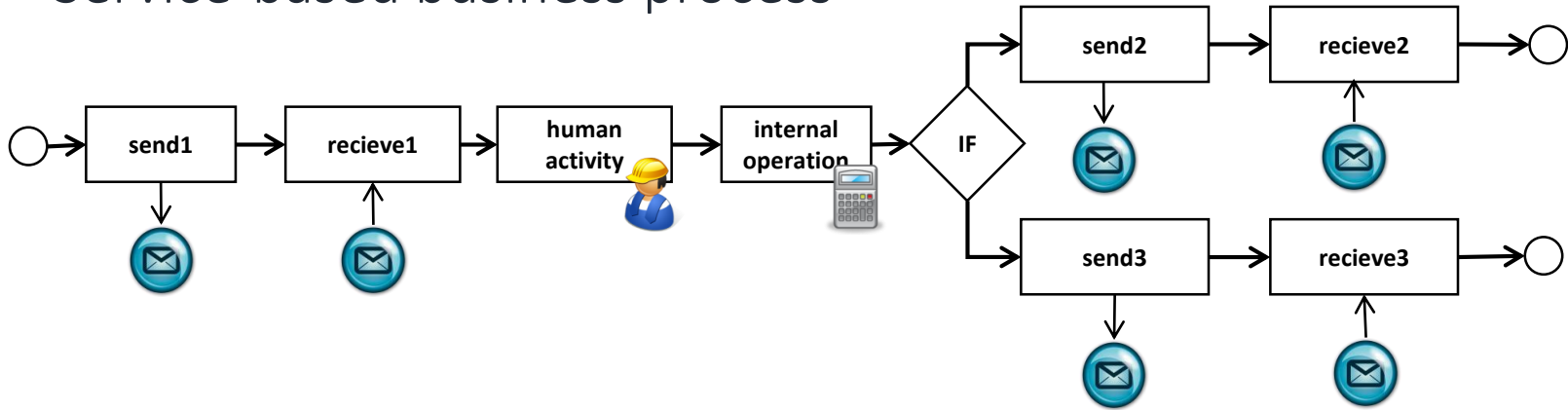
- Self-Managed Systems,
- Multi-Agent Systems,
- Ubiquitous Computing,
- Service-Oriented Computing.

“To be self-managed, a system should reconfigure itself to satisfy the changed specification and/or environment.”

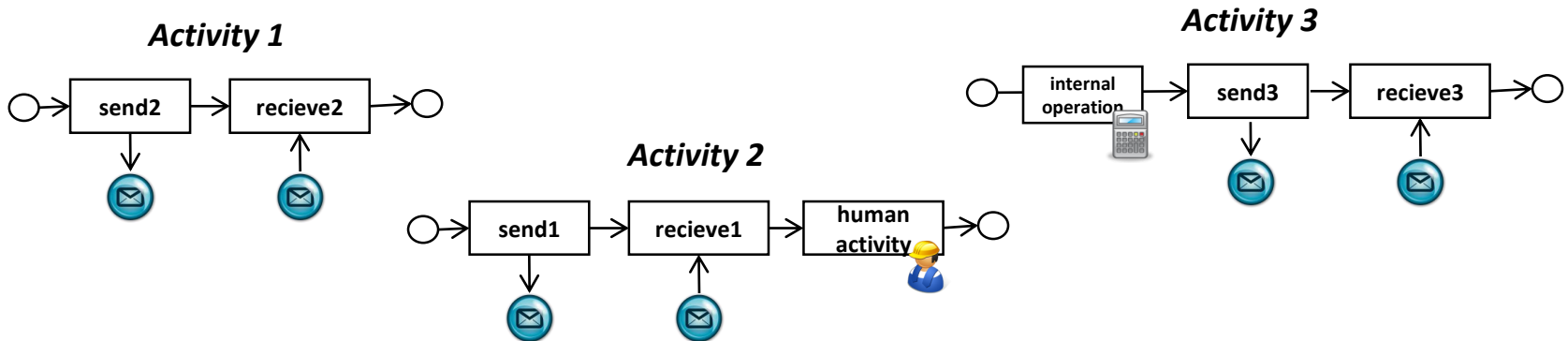
Jeff Kramer and Jeff Magee. *Self-managed systems: an architectural challenge*. In International Conference on Software Engineering, ICSE , 2007.

# Dynamic SOA

## ➤ Service-based business process



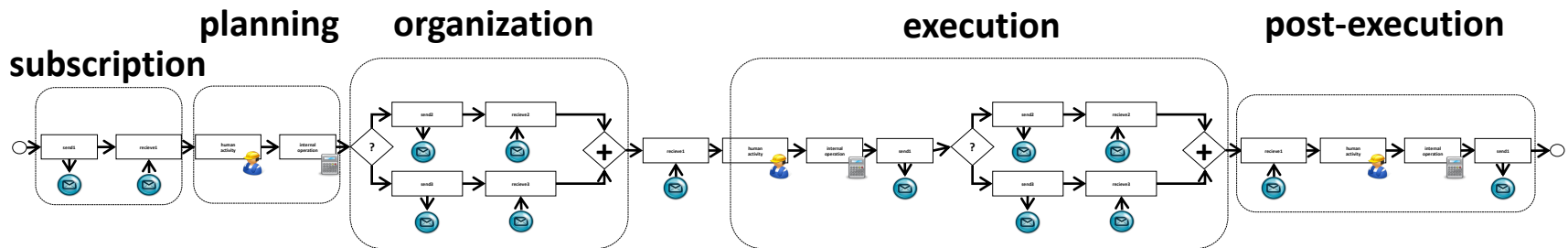
## ➤ Process fragments – building blocks for processes



# Dynamic Factors

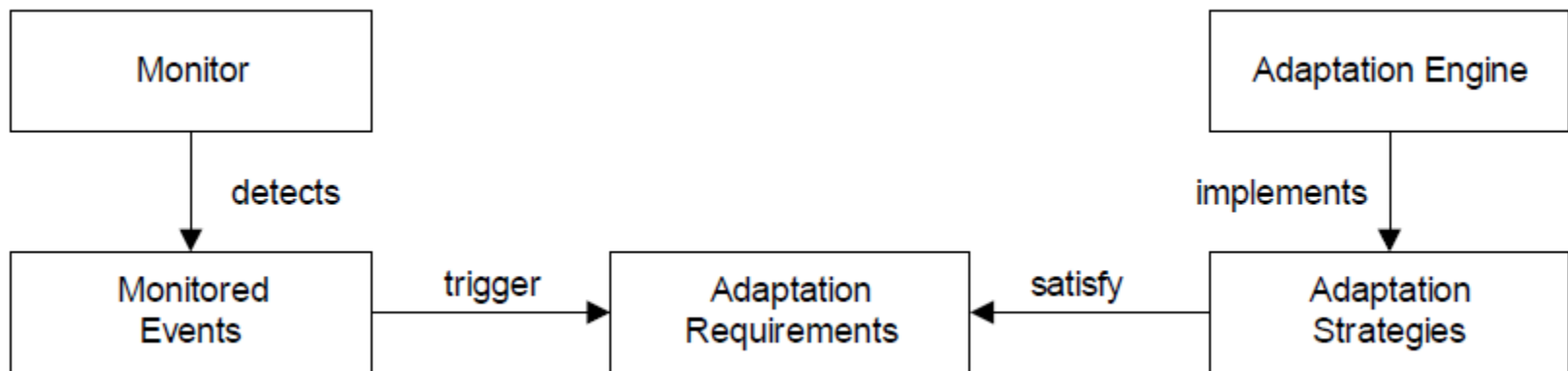
1. **volatile context** (may affect certain procedures or require extra procedures);
2. **customizable procedures** (each bus company may perform different stages differently);
3. **dynamic fragments availability** (fragments may disappear and reappear, new fragments may emerge);

## Travel Assistant



# Self-adaptation

- Adaptation of service-based business processes.
- Business processes allow for specifying complex and structured business activities (and applications) in SOA. That is why their ability to flexibly adapt to various changes in the execution environment is of high importance for enabling *adaptable service-based systems*.



# Self-adaptation

Different approaches exist to realize the adaptation of service-based business processes. Based on the adaptation requirements criterion we can distinguish:

- *Built-in* adaptation
- *Rule-based* adaptation
- *Goal-based* adaptation

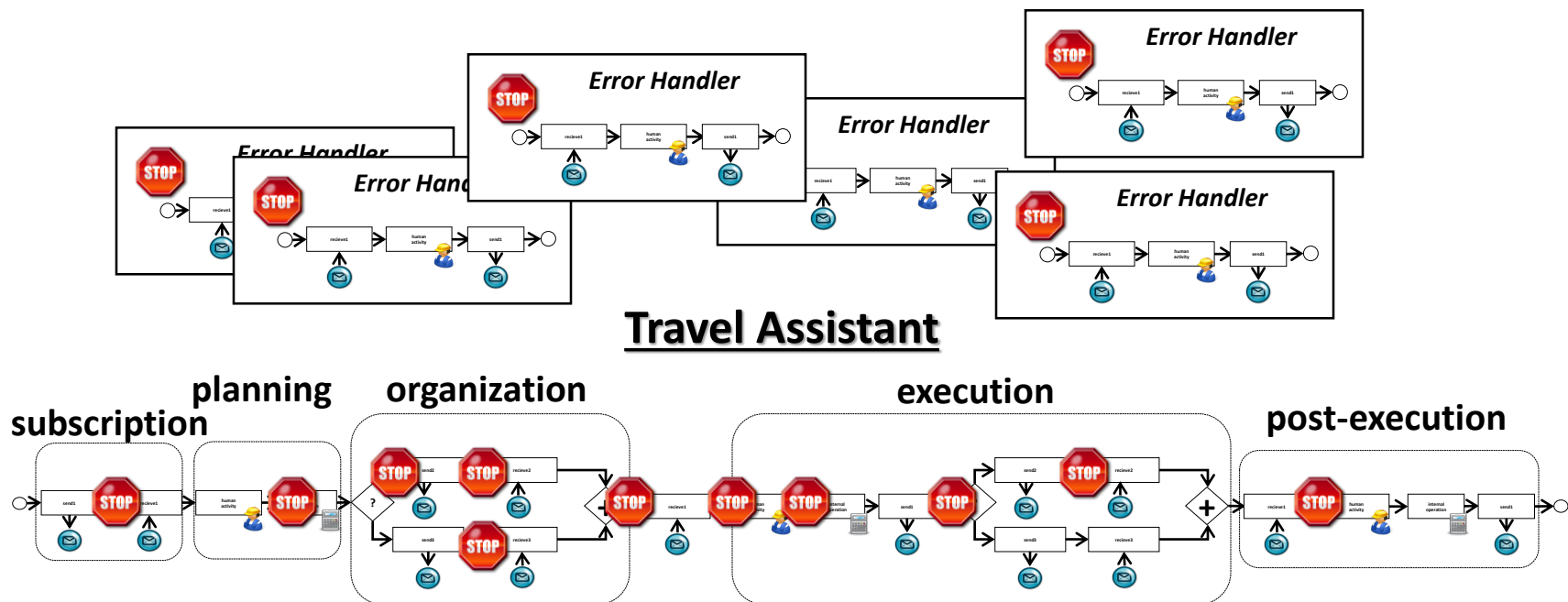
Heorhi Raik. *Service Composition in Dynamic Environments: From Theory to Practice*. PhD Thesis, 2012.

# Built-in adaptation

## Existing solutions

### ➤ Error Handling

1. volatile context;
2. dynamic fragments availability.



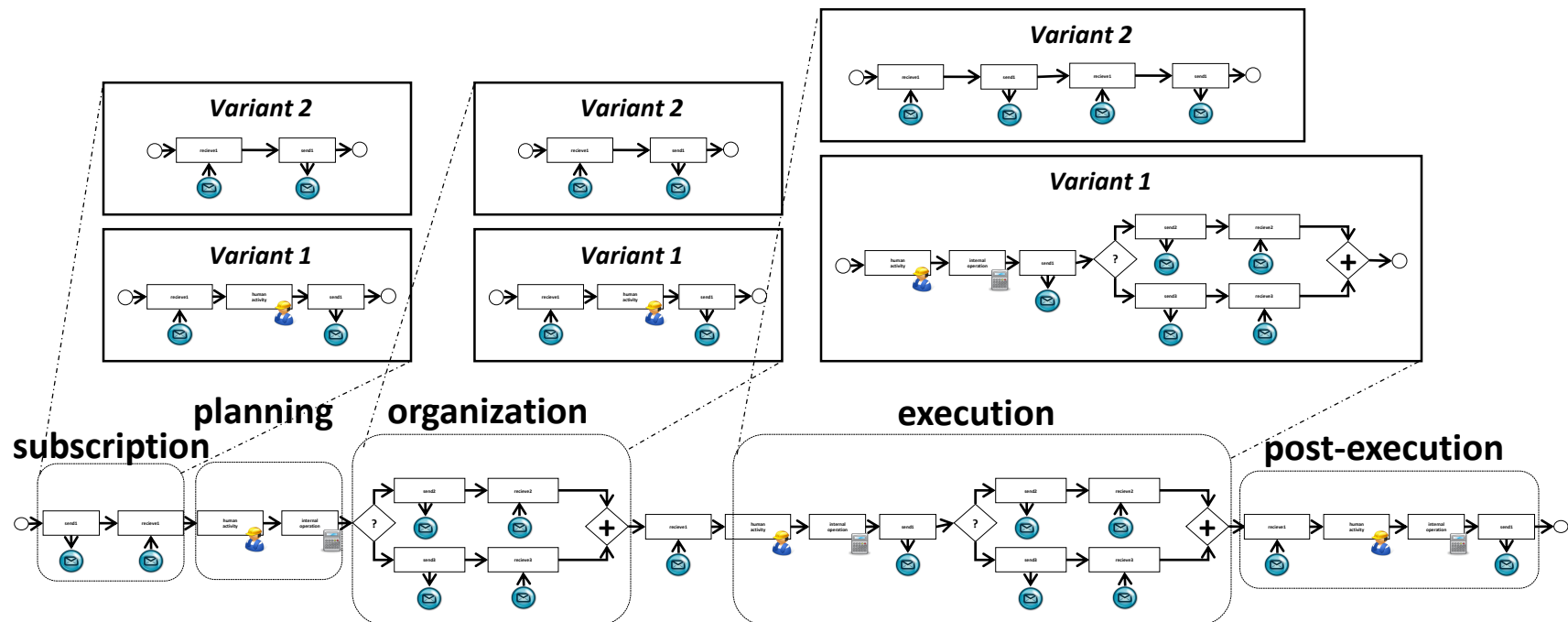


# Rule-based adaptation

Existing solutions

## ➤ Process Variant

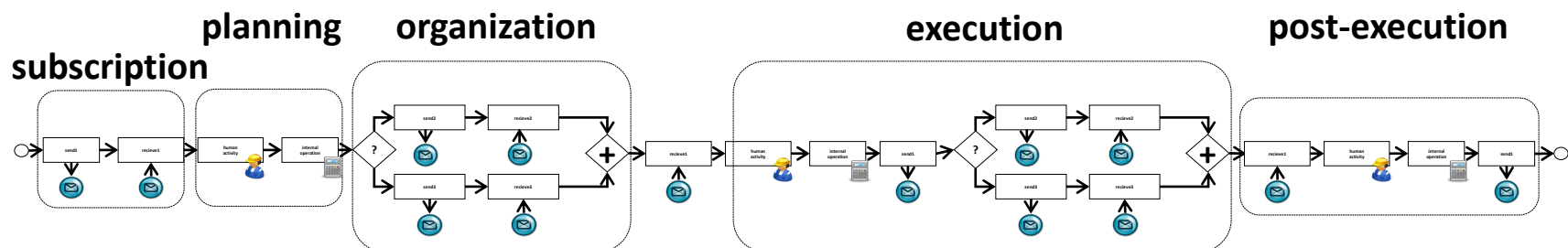
1. customizable procedures;
2. dynamic fragments availability;



# Goal-based adaptation

- Goal-based approaches express their adaptation needs in form of *abstract goals* to be achieved.
- Based on *abstract workflows*, defining the initial process, *dynamically bounded to services* for each task that needs to be performed.
  1. Volatile context;
  2. customizable procedures;
  3. dynamic fragments availability;

## Travel Assistant



# Other Approaches for self-adaptive SBS

- *Adaptive service-based systems* allow for combining numerous services into business applications implementing a business logic, also complex, that is provided by the collaborations among the involved services.
- Dynamic Software Product Lines (DSPL)
- Configurable Process Modeling Approaches (CPM)
- Models@Runtime-based approaches
- ...

# In Summary

- Overview on socio-technical systems
  - A motivating scenario
- Engineering socio-technical service-based systems
- Dynamic adaptation approaches
- Design for adaptation

# Systems Requirements

- **Heterogeneity & autonomy awareness:** The system must consider the heterogeneity of the services in terms of technologies or offered functionalities (e.g., REST API vs. SOAP, online booking vs. on board ticketing);
- **Openness awareness:** The system must be capable to operate in open environments with continuously entering and leaving services, which are not known a priori (e.g., a new ride-sharing service is available in the city);
- **Interoperability:** The system must be capable to propose complex solutions taking advantages of the variety of services (e.g., a user needs of a unique solution with her booking, payment, train journey, and taxi ride);
- **Customizability:** The system must provide users with personalized solutions (e.g., a wheelchair user has preferences on transportation means and stops);
- **Context awareness:** The system must take into account the state of the environment (e.g., strikes, bad weather, roadworks);
- **Adaptivity:** The system must be able to *react* and *adapt* to changes in the environment that might occur and affect its operations (e.g., a strike affects the user's train journey and the system offers a new journey plan);
- **Information accuracy:** The system must provide up to date and reliable information and solutions (e.g., temporary changes on a bus route);
- **Portability:** The system must be deployable in different environments without an ad-hoc reconfiguration from the developers (e.g., the travel assistant must be usable in Trento as well as in Paris).

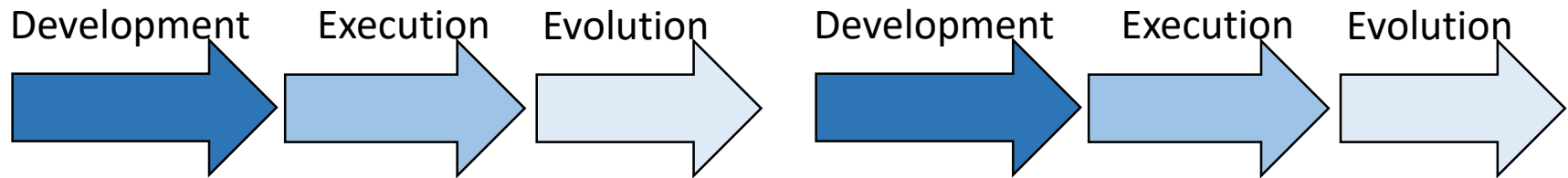
# Main limitations of current approaches

	DSPL	Mashups	CPM	M@R	Microservices
Heterogeneity & autonomy aw.	X	X			
Openness awareness	X	X	X	X	
Interoperability		X		X	
Customizability					
Context awareness					
Adaptivity		X	X		X * short-term adaptation
Information accuracy					
Portability	X	X	X	X	

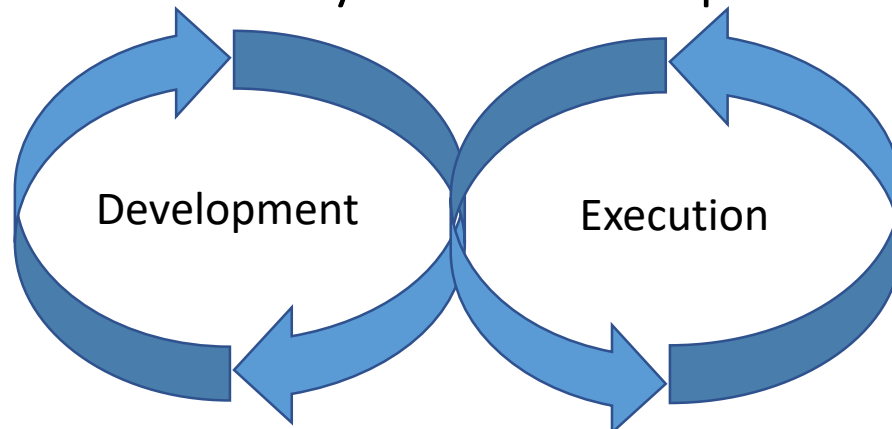
# Gap between Design & Execution

- Low applicability in open environments, typical of modern systems.
- There exists a gap between the *design* and the *operation* of systems.

## Traditional systems evolution process



## New trend in systems evolution process

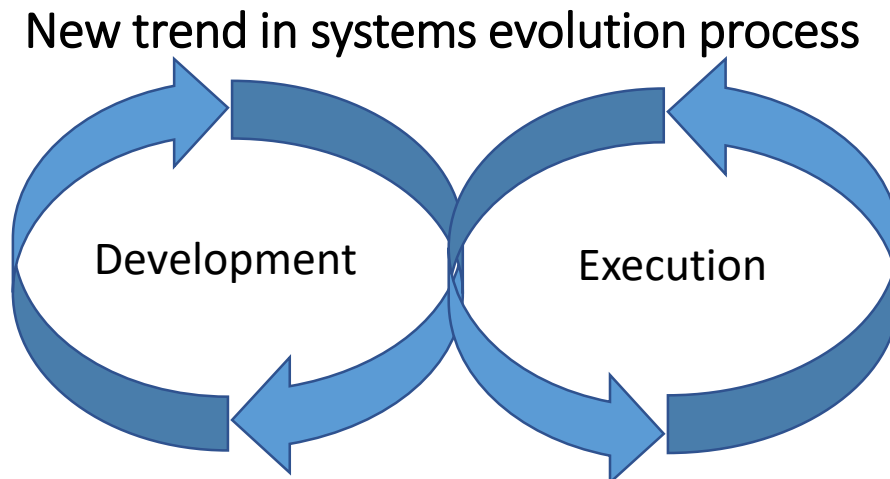


# Gap between Design & Execution

- Low applicability in open environments, typical of modern systems.
- There exists a gap between the *design* and the *operation* of systems.

*Adaptation cannot be considered an exception!!!  
Service-based systems need to be*

**ADAPTIVE BY DESIGN**





# Outline of the tutorial

- Overview on socio-technical systems
  - A motivating scenario
- Engineering socio-technical service-based systems
- Dynamic adaptation approaches
- *Design for adaptation*

# Problem Statement

We should define models, techniques and tools for the continuous *development* and *deployment* of service-based systems, to support their complete life-cycle by facilitating:

- the *continuous integration* of new services
- the *systems operation under dynamic circumstances*,

to face the openness and dynamicity of the environment.

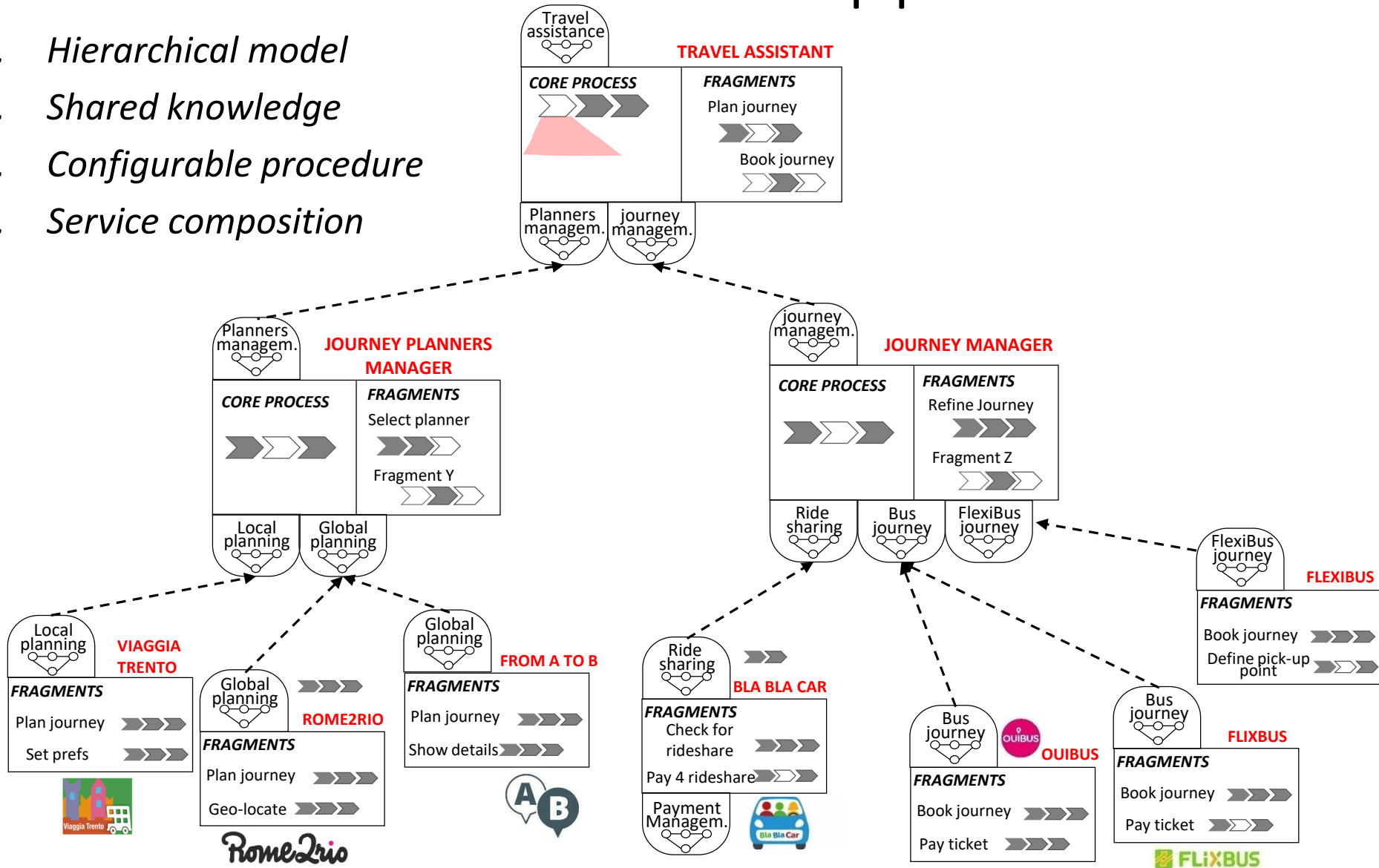
We do so by introducing a *design for adaptation framework*:

- supports the design, development and operation of systems operating in highly dynamic environments;
- considers the adaptivity as an intrinsic characteristic of systems rather than an exception to be handled.

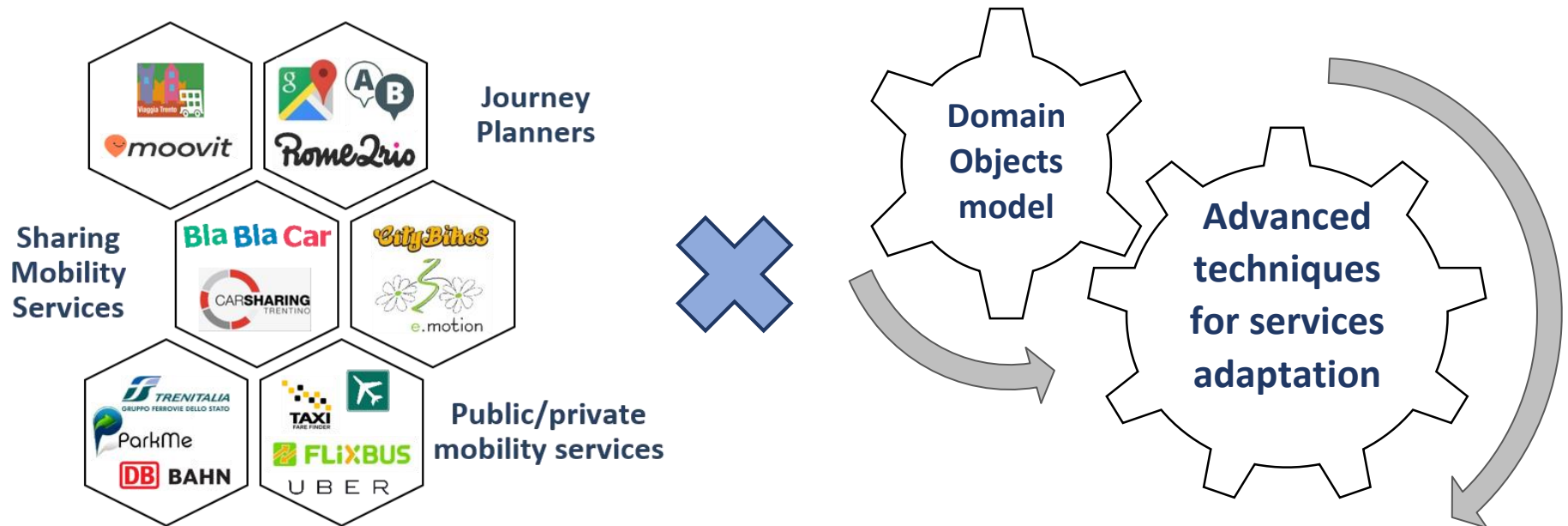
A. Bucchiarone, C. Cappiello, E. Di Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore. ***Design for adaptation of service-based applications: Main issues and requirements.*** In ServiceWave at ICSOC 2009.

# Overview of the Approach

1. Hierarchical model
2. Shared knowledge
3. Configurable procedure
4. Service composition



# Overview of the Approach



# The Design for Adaptation Approach

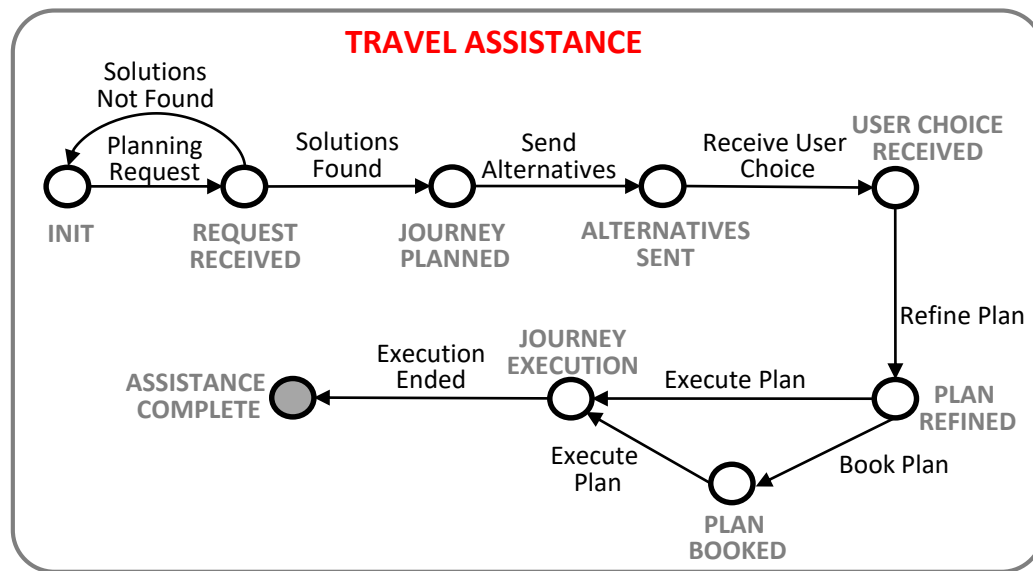
- It is based on two distinct but correlated models – the *Domain model* and the *Domain Object model*.
- It allows for the separation of concerns – *application* (the *what*) vs. *adaptation* (the *how*) logic (*design-time* vs. *runtime*).
- It provides support for the *runtime and automatic services selection, composition* and *re-configuration*.

A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, P. Traverso: *Design for Adaptation of Distributed Service-Based Systems*. ICSOC 2015

A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, P. Traverso: *Incremental Composition for Adaptive By-Design Service Based Systems*. ICWS 2016

# The Domain Model

- The **Domain** is modeled as a set of **domain properties**, i.e., relevant concepts (e.g., bus journey, ticket booking, journey monitoring, journey planning, etc.) in the referred domain (e.g., mobility). A domain property is a **State Transition System**.

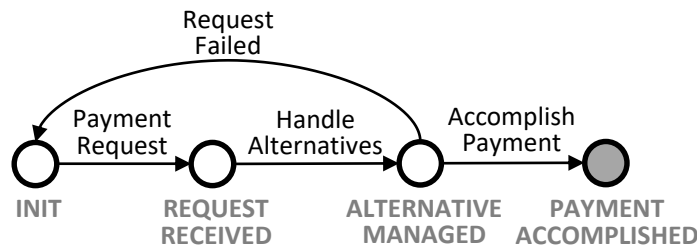


**Definition 1 (Domain Property).** A domain property is a state transition system  $dp = \langle L, l^0, E, T \rangle$ , where:  $L$  is a set of states and  $l^0 \in L$  is the initial state;  $E$  is a set of events; and  $T \subseteq L \times E \times L$  is a transition relation.

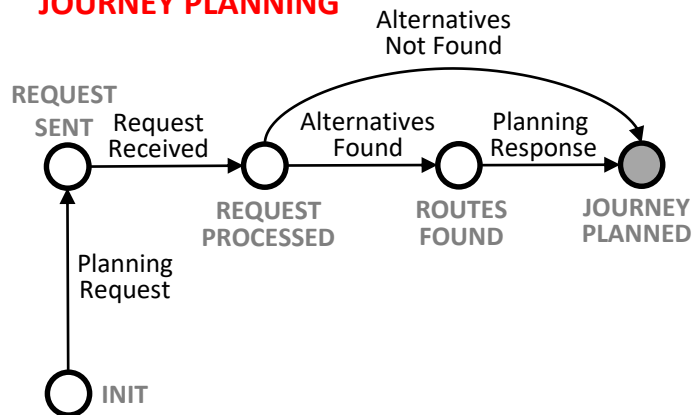
# The Domain Model

**Definition 2** (Domain model). A domain model is a set of domain properties  $C = \{dp_1, dp_2, \dots, dp_n\}$  with  $dp_i = \langle L_i, l_i^0, E_i, T_i \rangle$  for every  $1 \leq i \leq n$ , and such that for every pair  $1 \leq i, j \leq n$ , if  $i \neq j$ , then  $E_i \cap E_j = \emptyset$ .

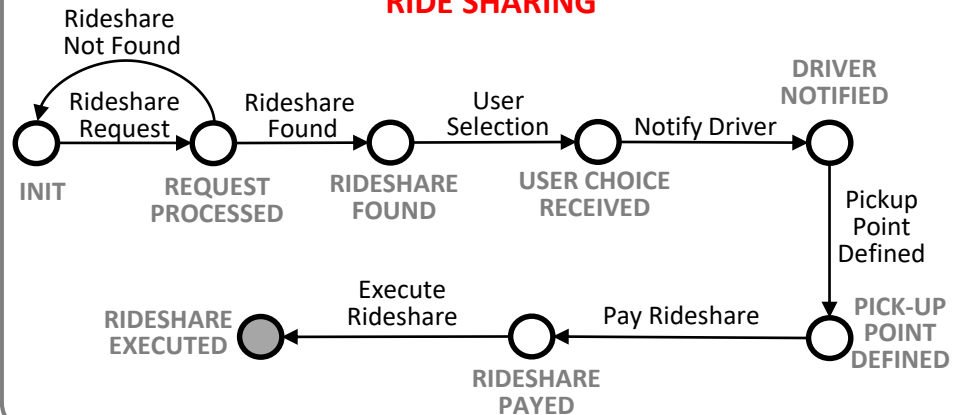
## PAYMENT MANAGEMENT



## JOURNEY PLANNING



## RIDE SHARING



# The Domain Object Model

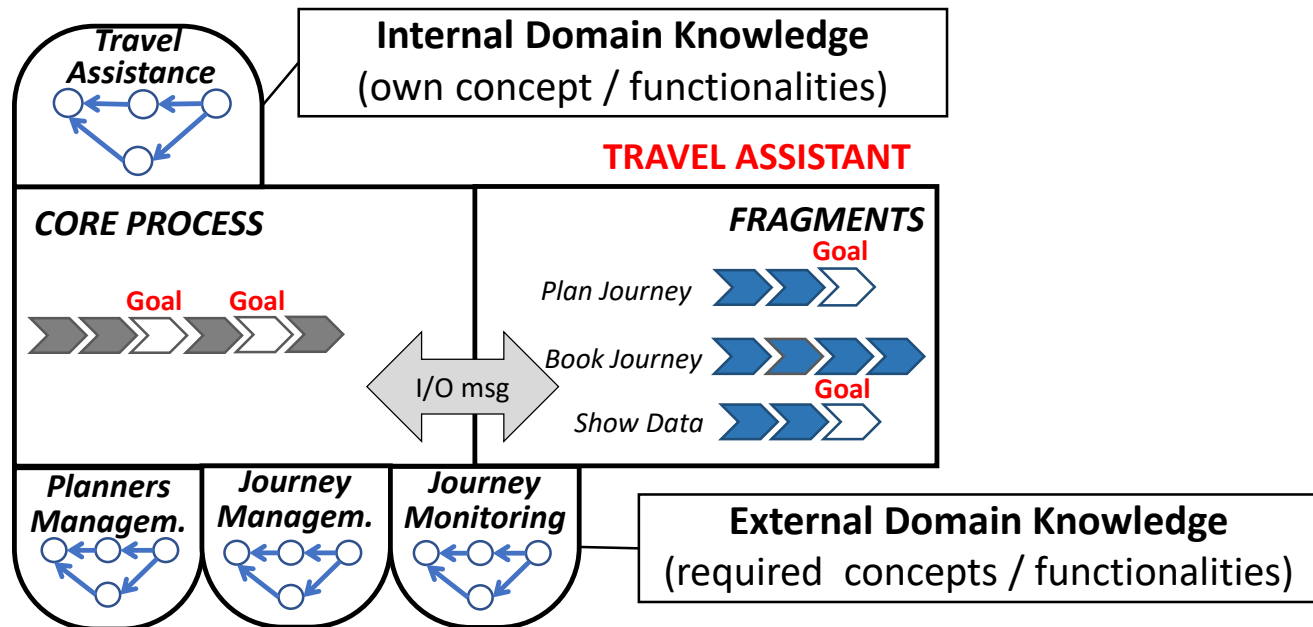
- A *domain object* represents a uniform way to model autonomous and heterogeneous *services* that facilitates the easy interconnection among them, via dynamic *relations*.
- Services in the environment must be wrapped as *Domain Objects*.
- Each Domain Object enters in the system by giving a *concrete implementation* of a domain property in the domain model.
- *Value-added domain objects* can be also specified.



# The Domain Object Model

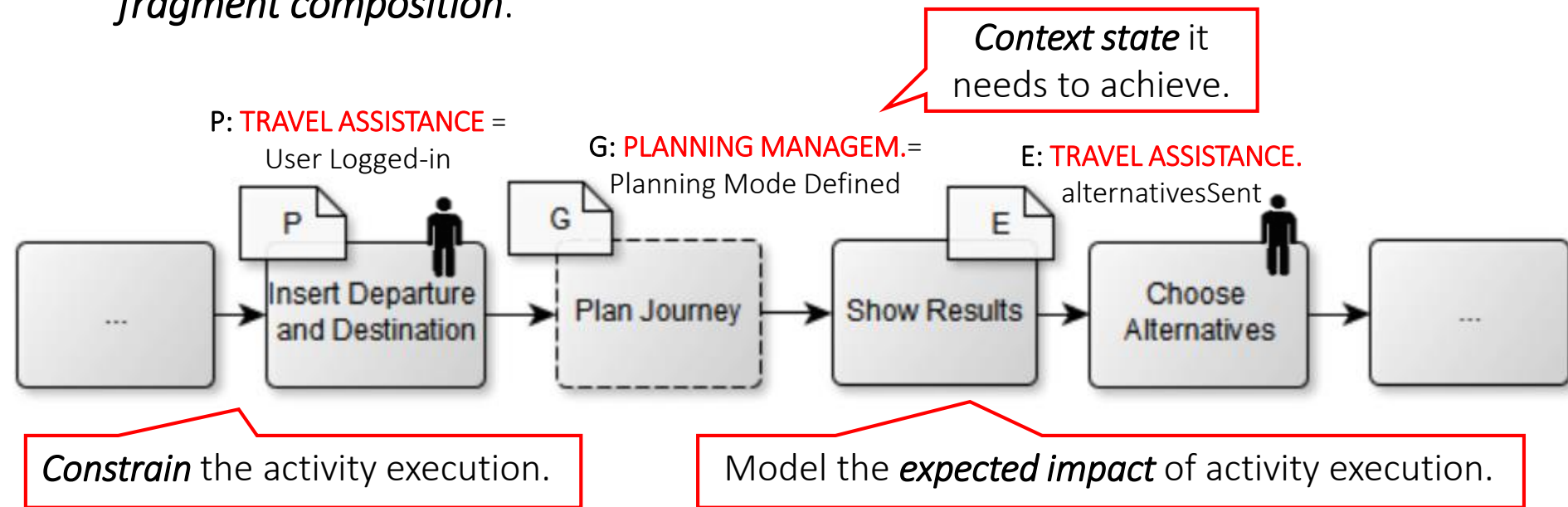
**Definition 6** (Domain Object). A domain object is a tuple  $o = \langle \mathbb{DK}_I, \mathbb{DK}_E, p, \mathbb{F} \rangle$ , where:

- $\mathbb{DK}_I$  is an internal domain knowledge,
- $\mathbb{DK}_E$  is an external domain knowledge,
- $p$  is a process, called core process, defined on  $\mathbb{DK}_I$  and  $\mathbb{DK}_E$ ,
- $\mathbb{F} = \{f_1, \dots, f_n\}$  is a set of processes, called fragments, defined on  $\mathbb{DK}_I$  and  $\mathbb{DK}_E$ , where for each  $f_i \in \mathbb{F}$ ,  $a \in A_{in}(f_i)$  implies  $a \in A_{out}(p)$  and  $a \in A_{out}(f_i)$  implies  $a \in A_{in}(p)$ .



# Core Processes and Fragments

- *The core process* implements the internal behavior of the domain object.
- *Fragments* model offered services that can be *dynamically discovered* and *used* by other components.
  - Supported specification languages: **APFL** (BPEL ++)
- Annotations are used to understand *adaptation needs* and perform *automated fragment composition*.

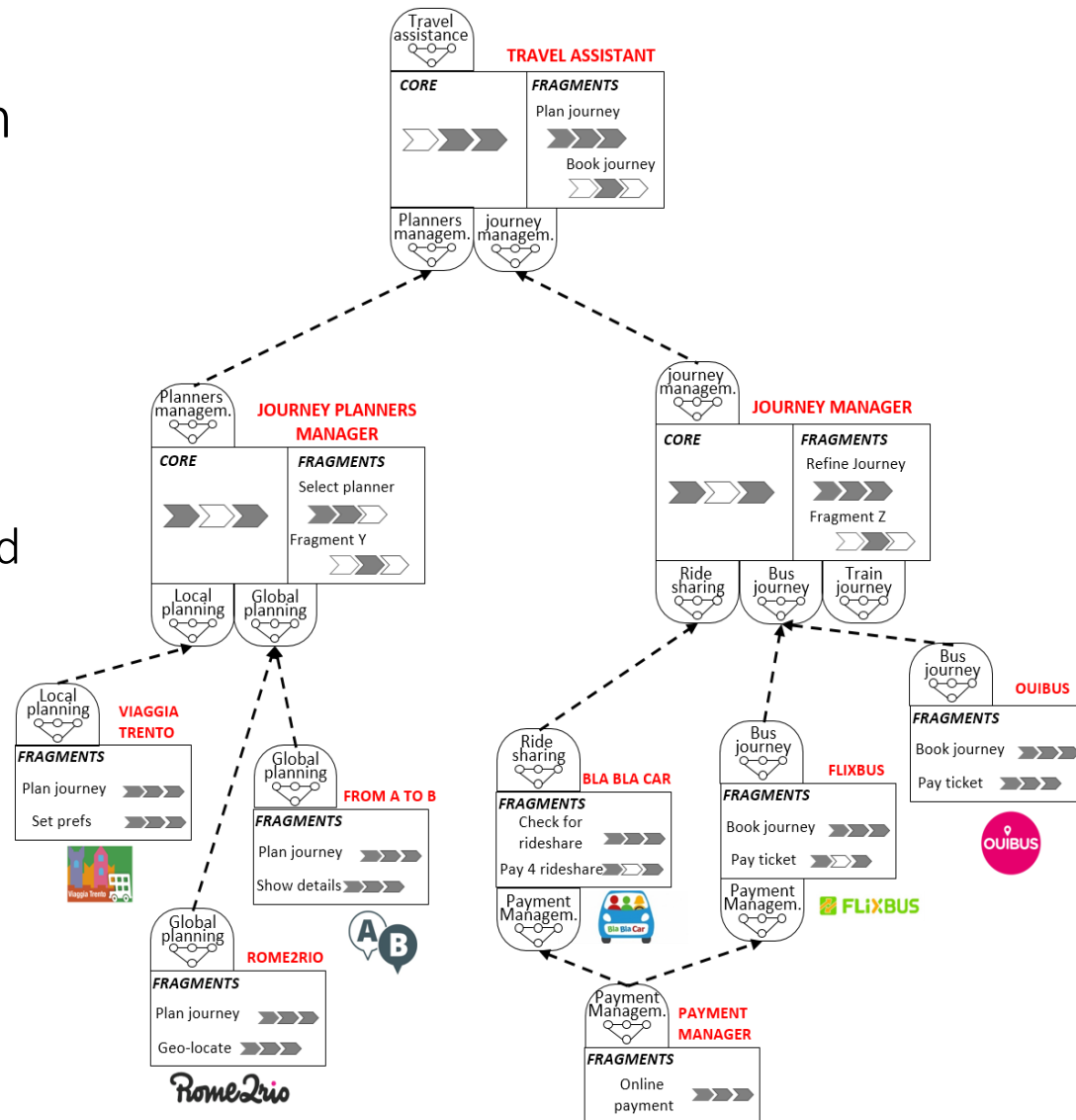


A. Bucchiarone, A. Lluch-Lafuente, A. Marconi, M. Pistore: *A Formalisation of Adaptable Pervasive Flows*. WS-FM 2009: 61-75

# Domain Objects Dependencies

The resulting adaptive system is a *dynamic hierarchical network* of domain objects.

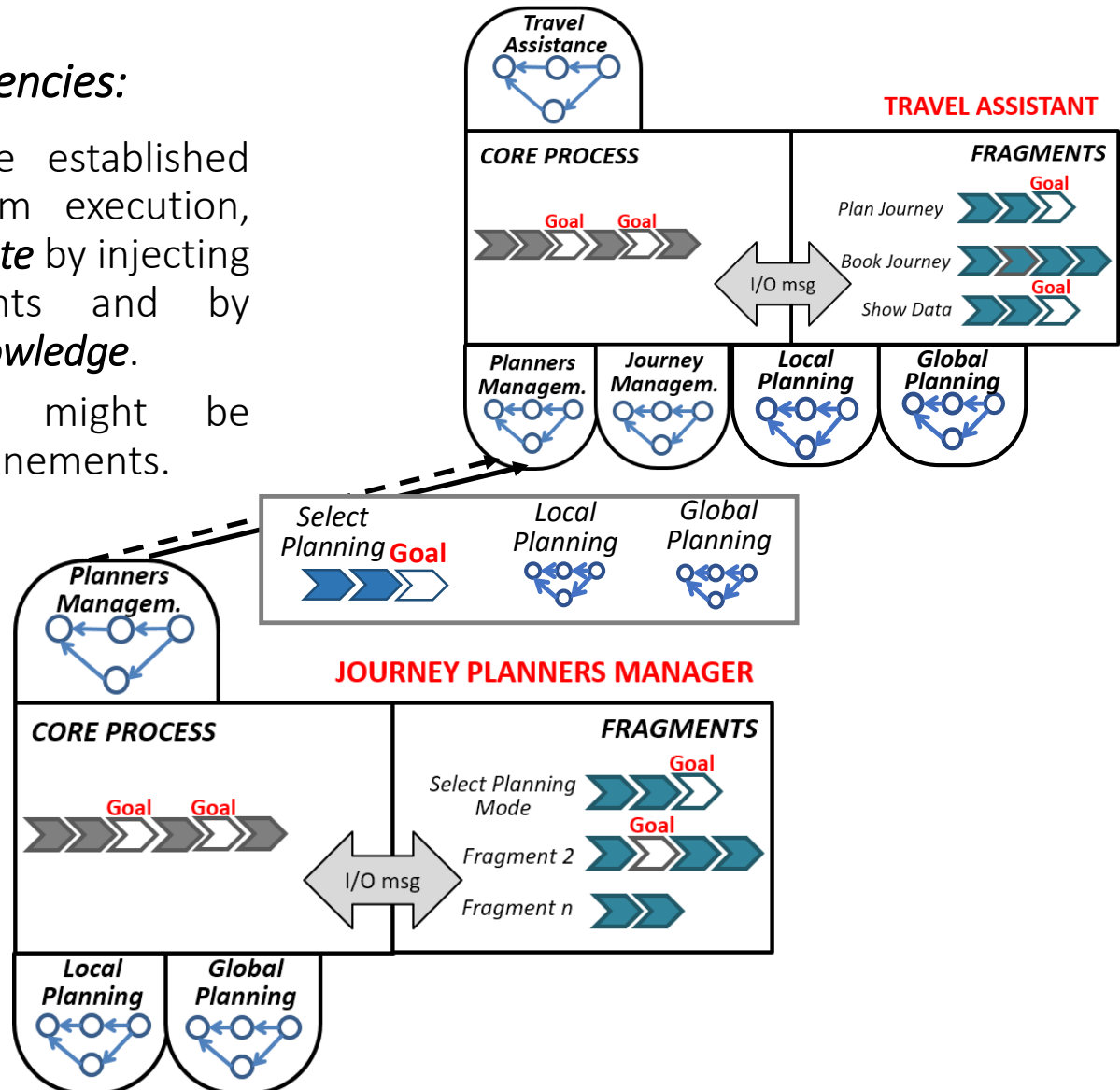
- **Soft Dependencies:** are established through matches between provided functionality (*internal domain knowledge*) and required behaviors (*external domain knowledge*).



# Domain Object's Knowledge Extension

From *soft* to *strong dependencies*:

- **Strong Dependencies:** are established when, during the system execution, *domain objects inter-operate* by injecting and executing fragments and by *spanning their external knowledge*.
  - new dependencies might be established due to refinements.



# Dynamic Service Composition

The approach exploits advanced techniques for *dynamic* and *incremental service composition* and *re-configuration* that are:

- **Context-aware**: the specialization of the application logic is based on the context.
- **Incremental**: each abstract activity is refined in the *local scope* of the domain object that is executing the activity.
- **Automatic**: techniques handle the *selection*, *composition* and *injection* of *process fragments*, without any human intervention.

H. Raik, A. Bucchiarone, N. Khurshid, A. Marconi, and M. Pistore. *ASTRO-CaptEvo: Dynamic context-aware adaptation for service-based systems*. In SERVICES 2012. [SERVICE CUP winner](#)

A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. *Dynamic adaptation of fragment-based and context-aware business processes*. In ICWS 2012. [Best Paper Award](#)

# Adaptation Mechanisms

- The *Adaptation Engine* we exploit is based on AI planning and it implements three types of *adaptation mechanisms* to resolve different runtime problems (e.g., abstract activity not refined, precondition violated, service unavailable).

P. Bertoli, M. Pistore, and P. Traverso. *Automated composition of web services via planning in asynchronous domains*. Artif. Intell., 2010.

# Adaptation Mechanisms

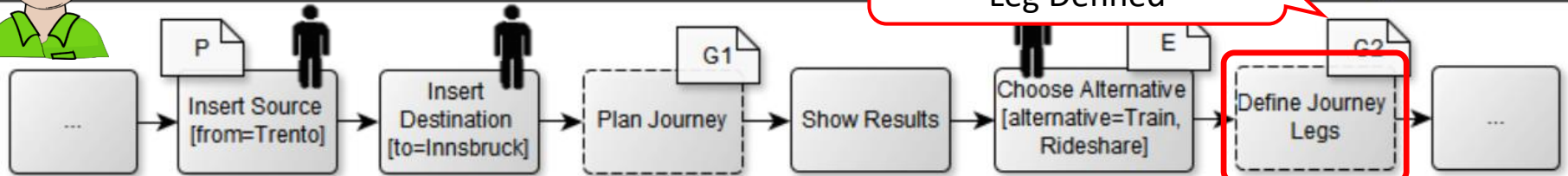
- Adaptation needs
  - Need for **refining an abstract activity** within a process instance
  - **Violation of a precondition** of an activity that is going to be executed
- Adaptation mechanisms
  - **Refinement**: dynamic refinement of abstract activity by context-aware composition of available fragments
  - **Local adaptation**: identify a fragment composition that allows to re-start a faulted process from a specific activity
  - **Compensation**: dynamically compute a compensation process for a specific activity
- Adaptation strategies
  - **Combine adaptation mechanisms** to solve complex adaptation problems
    - E.g., Re-refinement, Backward adaptation
  - **Search for alternative solutions**
    - E.g., Local on current activity -> Backward on current refinement -> Re-refinement -> ...
  - **One-shot vs incremental** adaptation

# Domain Objects Runtime Execution



**JOURNEY MANAGEM. =**  
Leg Defined

process@TravelAssistant

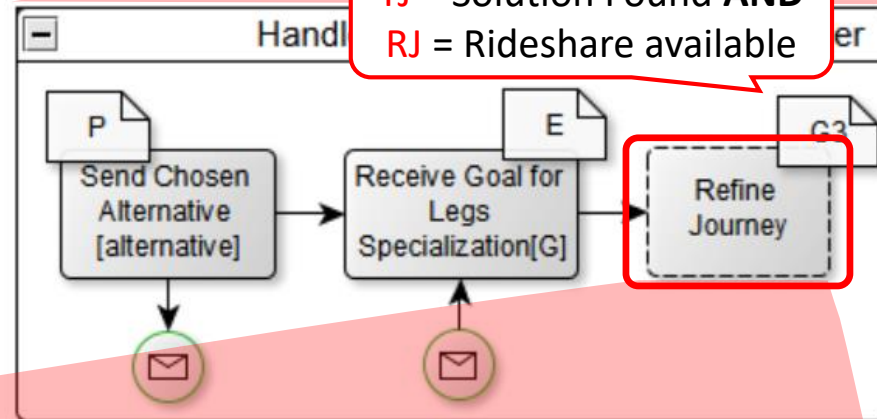


The  
**WHAT**

*Incremental, on-the-fly  
refinement. Dynamic service  
composition.*

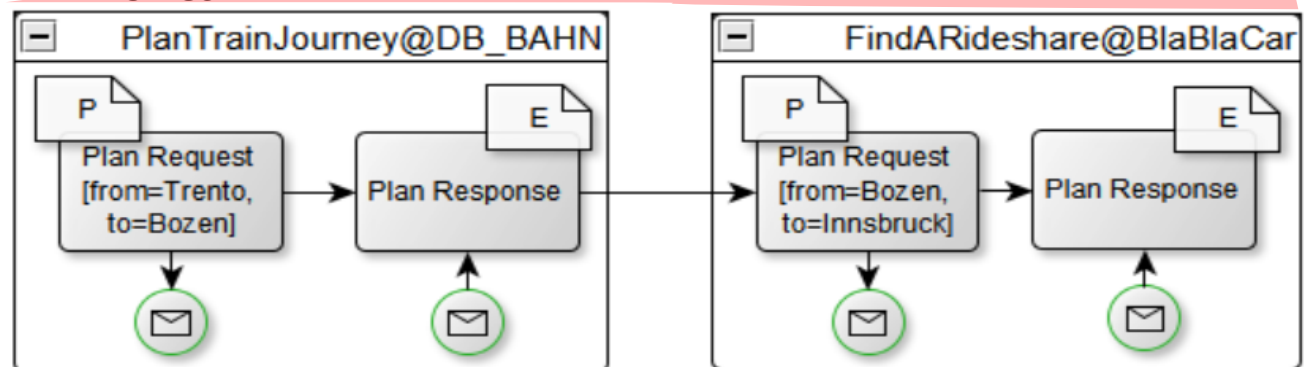
**PLAN FOR G2**

**TJ = Solution Found AND**  
**RJ = Rideshare available**



The  
**HOW**

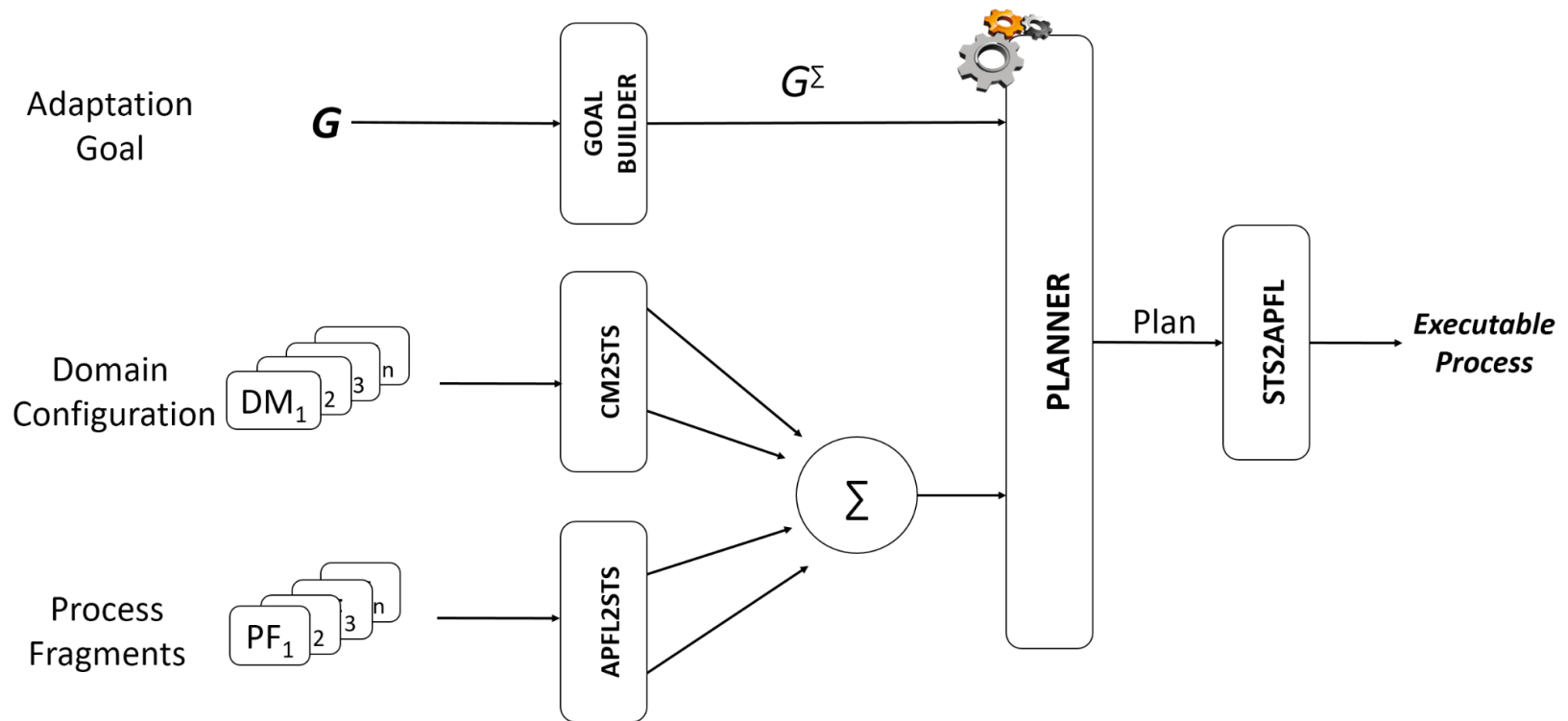
**PLAN FOR G3**





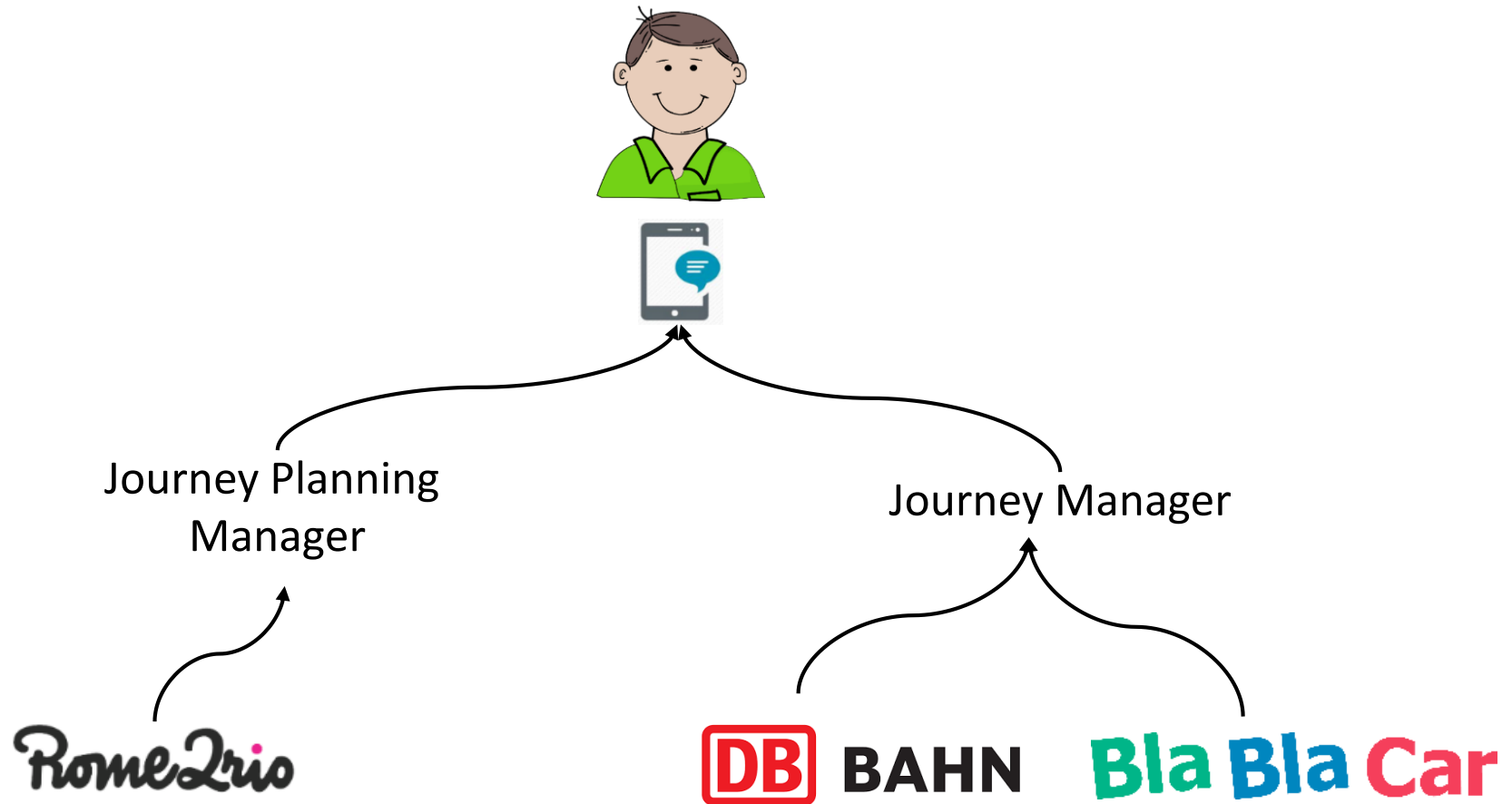
# Service Composition as AI Planning

- The *composition problem* is transformed into a *planning problem*.
- The approach is formally proved to be **correct** and **complete**.



P. Bertoli, M. Pistore, and P. Traverso. *Automated composition of web services via planning in asynchronous domains*. Artif. Intell., 2010.

# Dynamic Domain Objects Hierarchy



# A Comprehensive Framework

PLATFORM PROVIDER

SERVICE PROVIDER

END-USERS

INTERACTION

ADAPTATION

SYSTEM MODELS

DOMAIN

# A Comprehensive Framework

PLATFORM PROVIDER

SERVICE PROVIDER

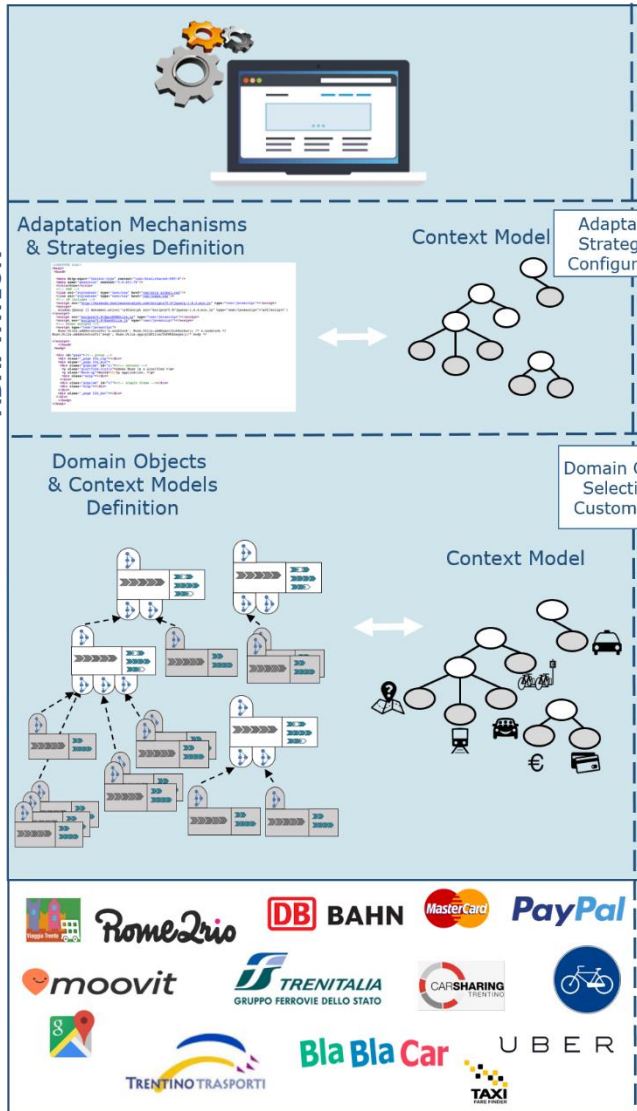
END-USERS

INTERACTION

ADAPTATION

SYSTEM MODELS

DOMAIN



# A Comprehensive Framework

PLATFORM PROVIDER

SERVICE PROVIDER

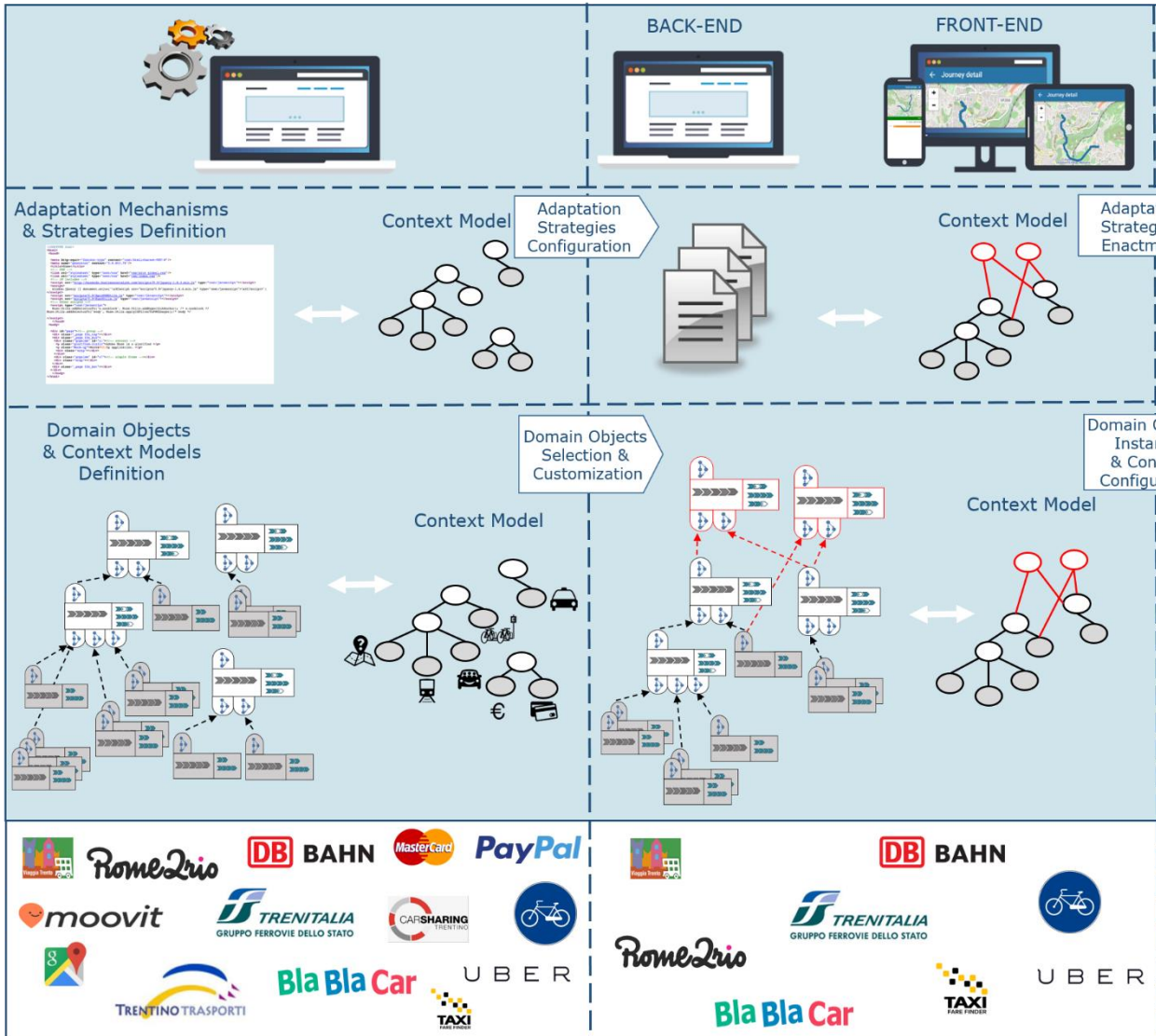
END-USERS

INTERACTION

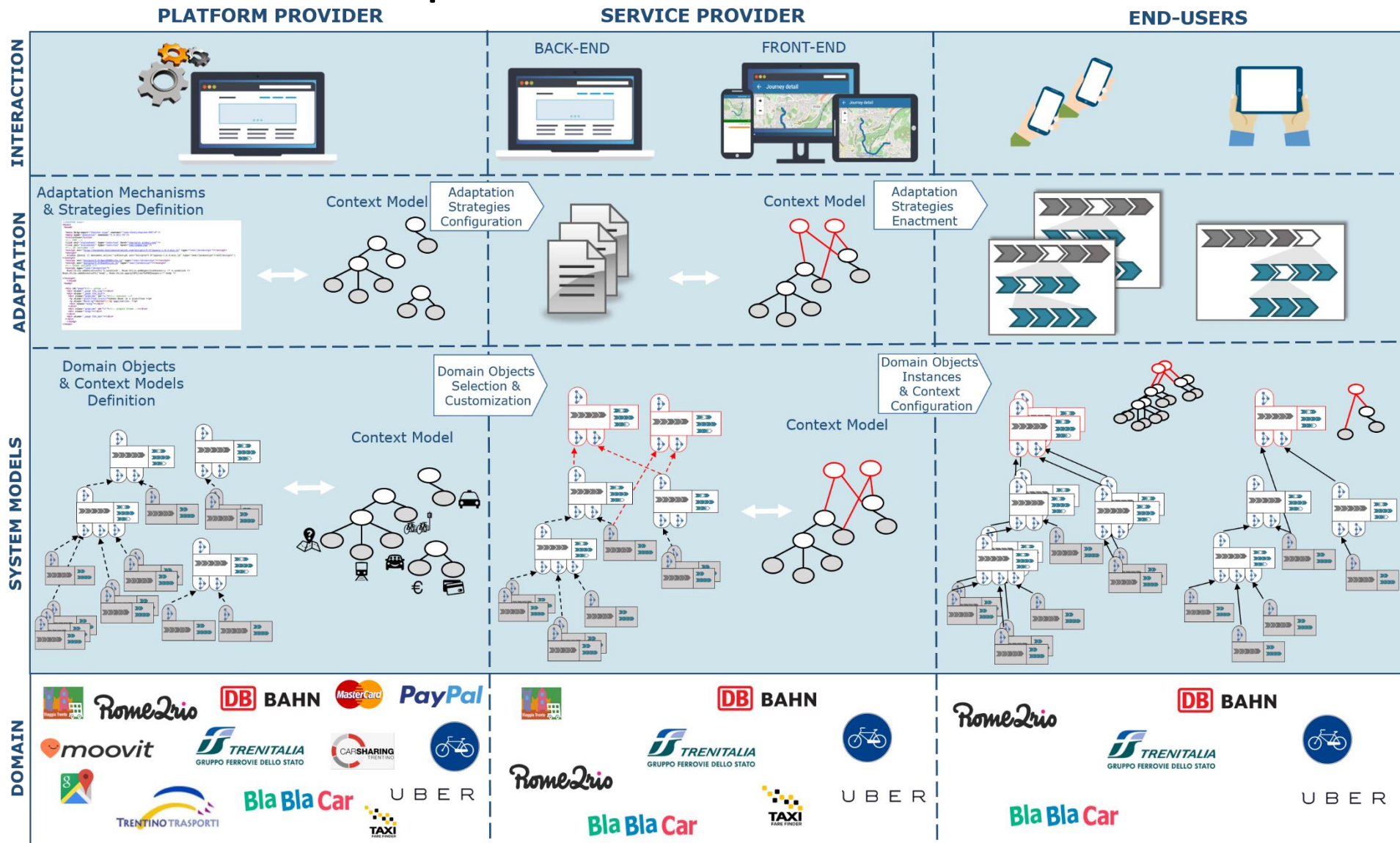
ADAPTATION

SYSTEM MODELS

DOMAIN



# A Comprehensive Framework



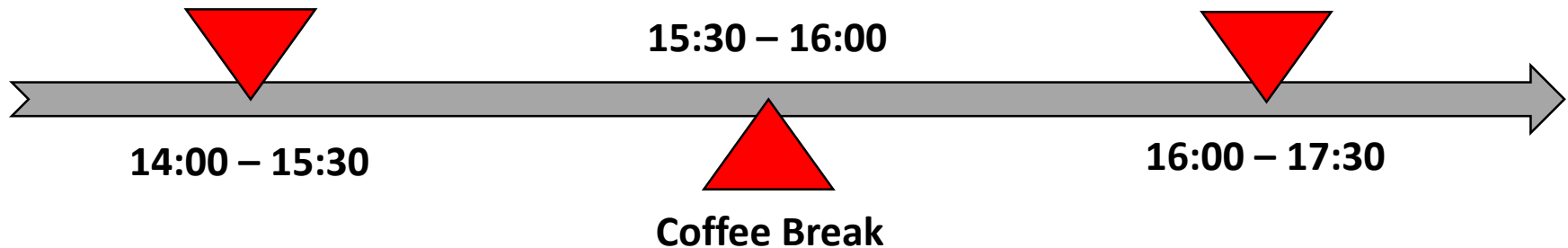
# Tutorial Timeline

## Part I:

- Context
- State of the art
- Design4adaptation

## Part II:

- Demo
- Tool support
- Scenarios

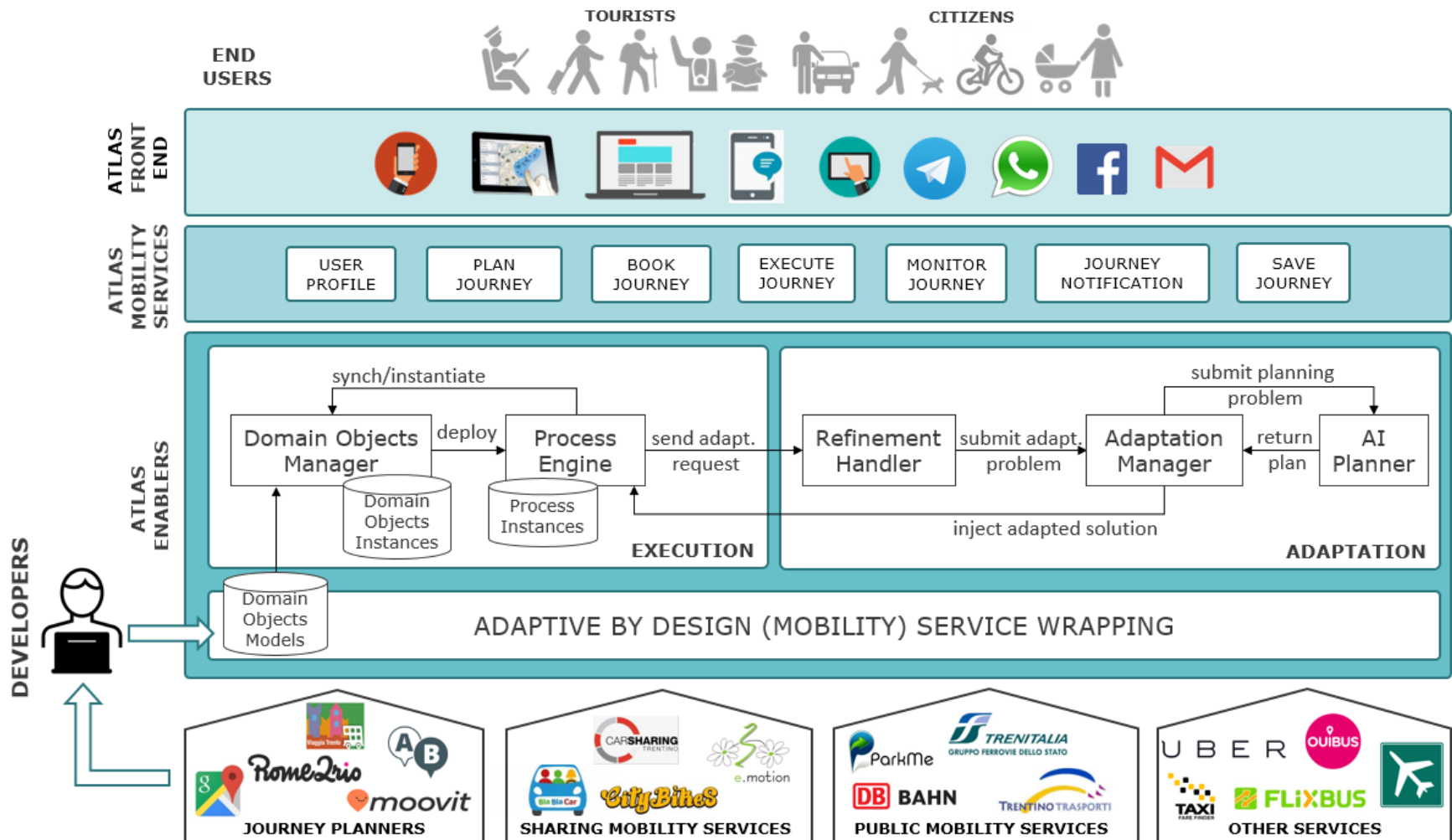




A World-wide Travel Assistant  
exploiting Service-based Adaptive Technologies

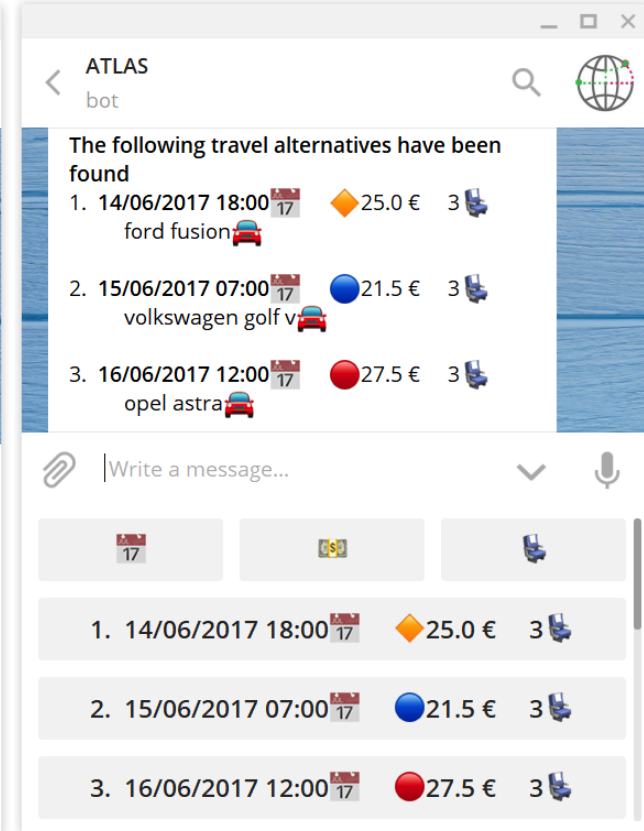
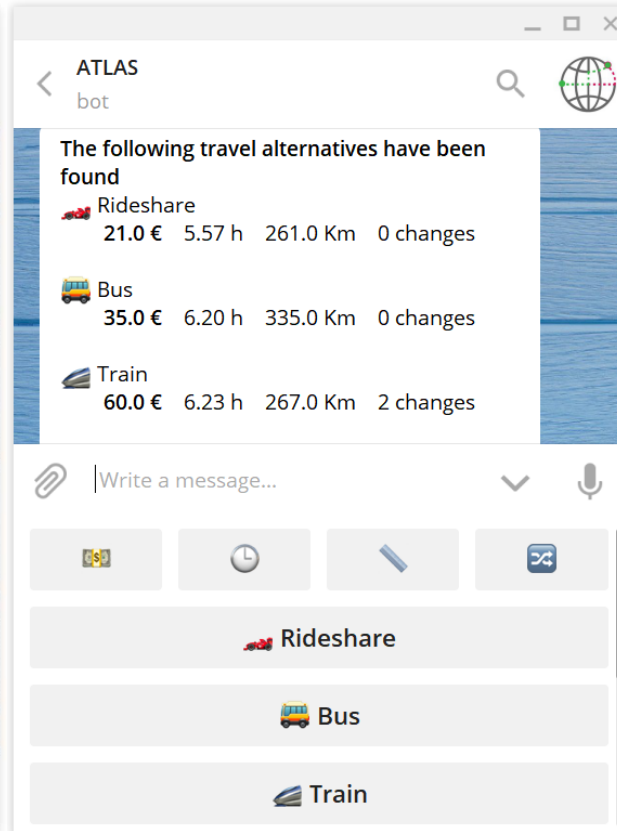


# ATLAS Travel Assistant



A. Bucchiarone, M. De Sanctis, and A. Marconi. *ATLAS: A world-wide travel assistant exploiting service-based adaptive technologies*. ICSC 2017

# ATLAS Travel Assistant



Rome2rio

Bla Bla Car

# ATLAS Travel Assistant

ATLAS  
File Edit Help

Play Step

Domain Objects Models Runtime Execution

Domain Object Instances  
TelegramTravelAssistant\_2

Process Model

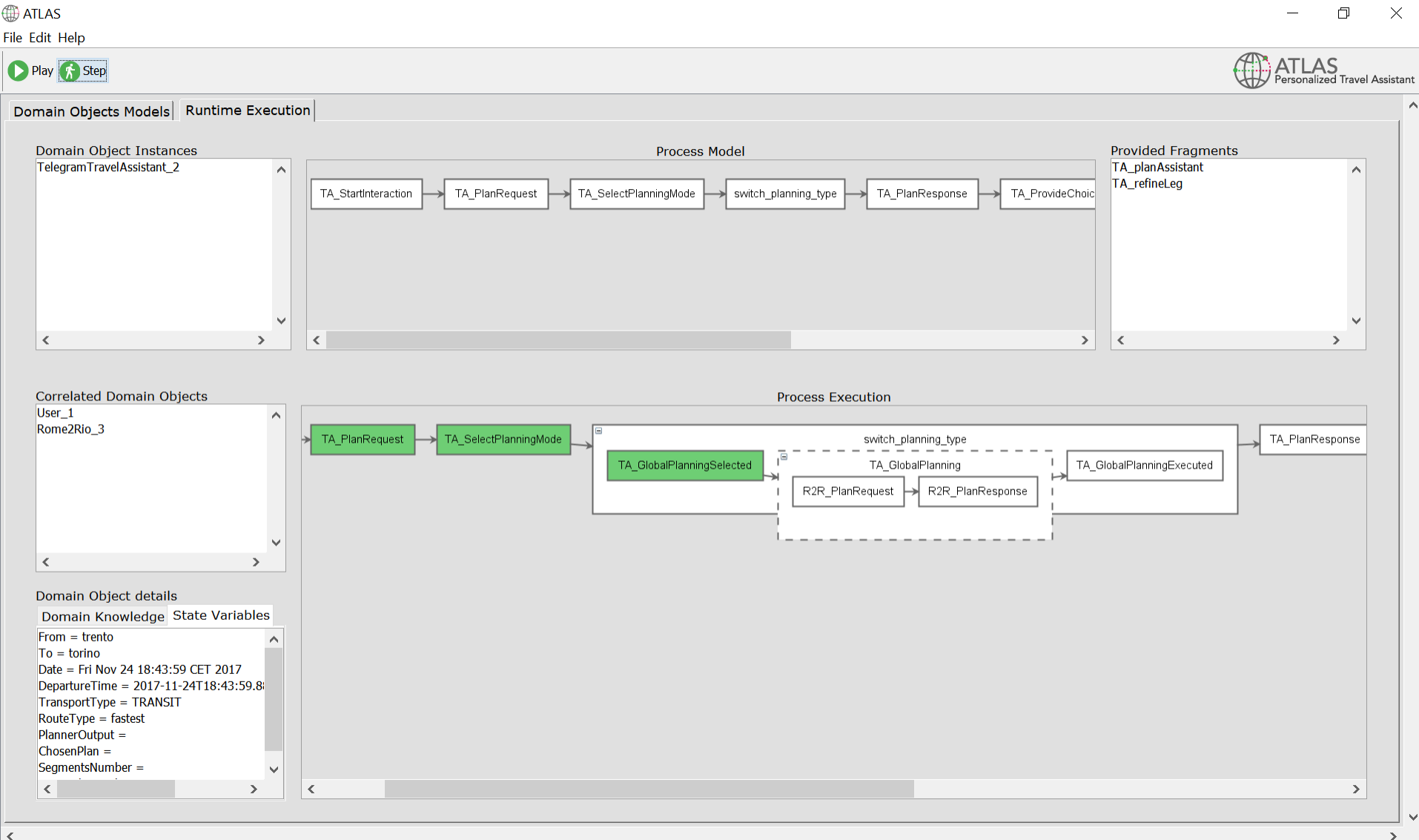
Provided Fragments  
TA\_planAssistant  
TA\_refineLeg

Correlated Domain Objects  
User\_1  
Rome2Rio\_3

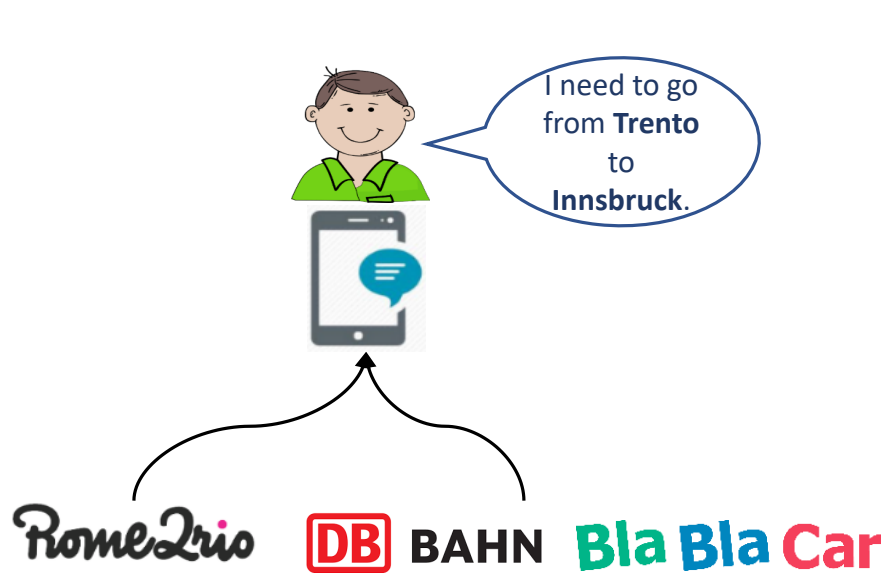
Process Execution

Domain Object details  
Domain Knowledge State Variables

From = trento  
To = torino  
Date = Fri Nov 24 18:43:59 CET 2017  
DepartureTime = 2017-11-24T18:43:59.8  
TransportType = TRANSIT  
RouteType = fastest  
PlannerOutput =  
ChosenPlan =  
SegmentsNumber =



# ATLAS Travel Assistant



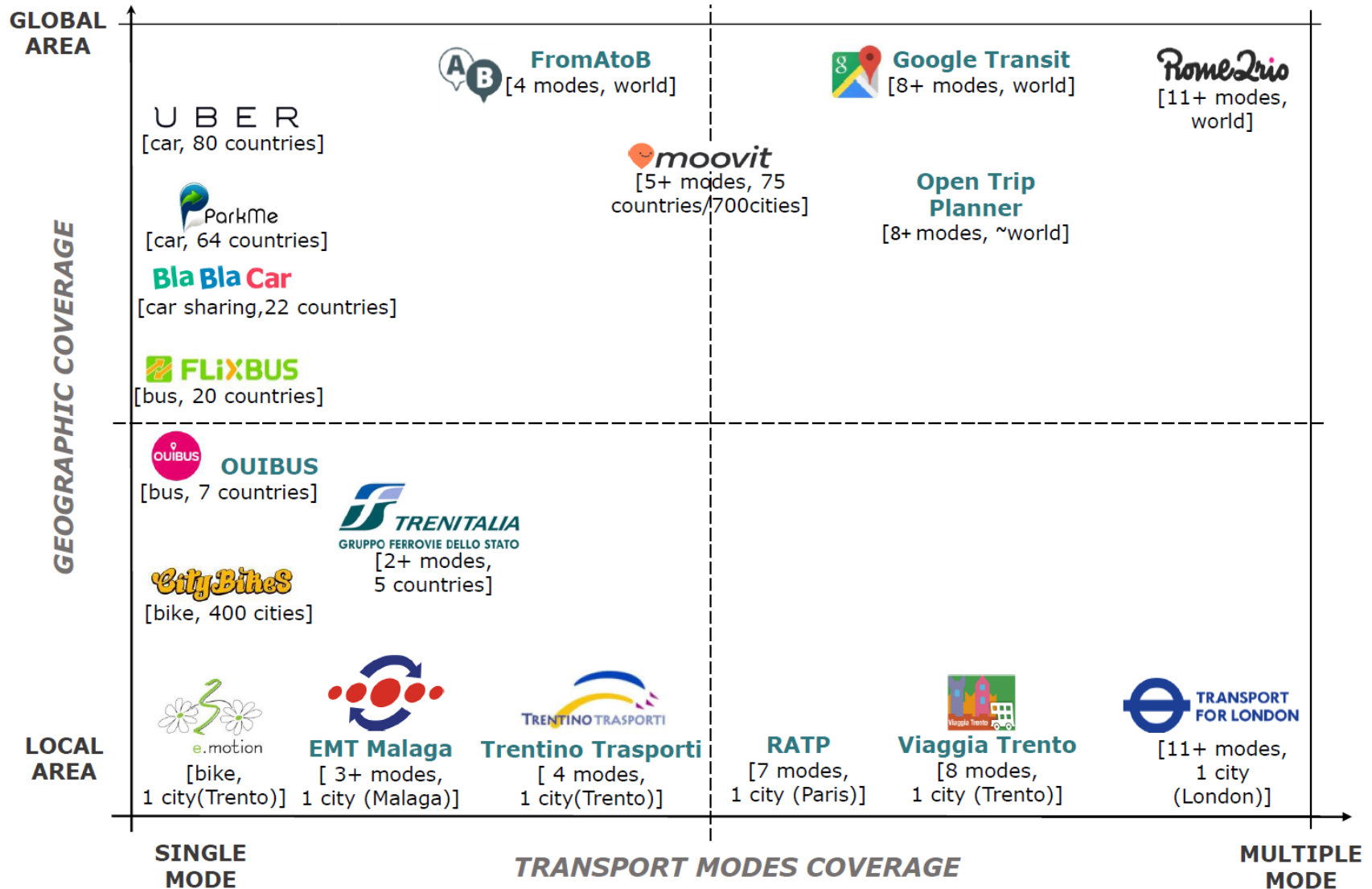
From *global* journey planning...



...to *local* journey planning.



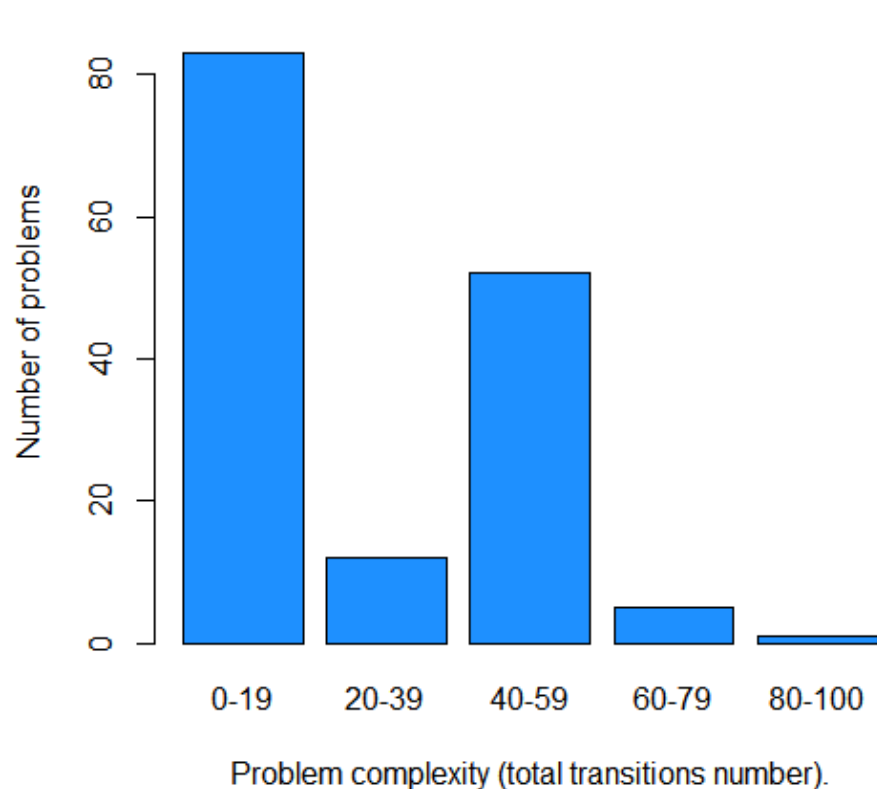
# ATLAS Travel Assistant



# ATLAS Demo

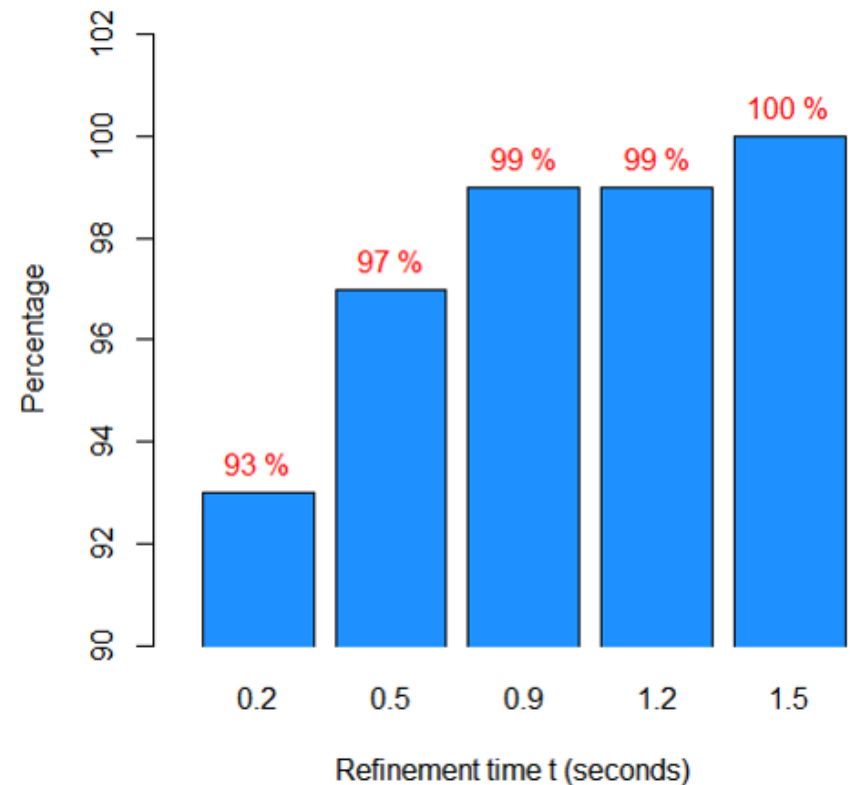
- ATLAS code available at: <https://github.com/das-fbk/ATLAS-Personalized-Travel-Assistant>.
- A video on the execution of ATLAS is available at: <https://vimeo.com/231387418>

# Analysis of the Adaptation



Distribution of Problems Complexity.

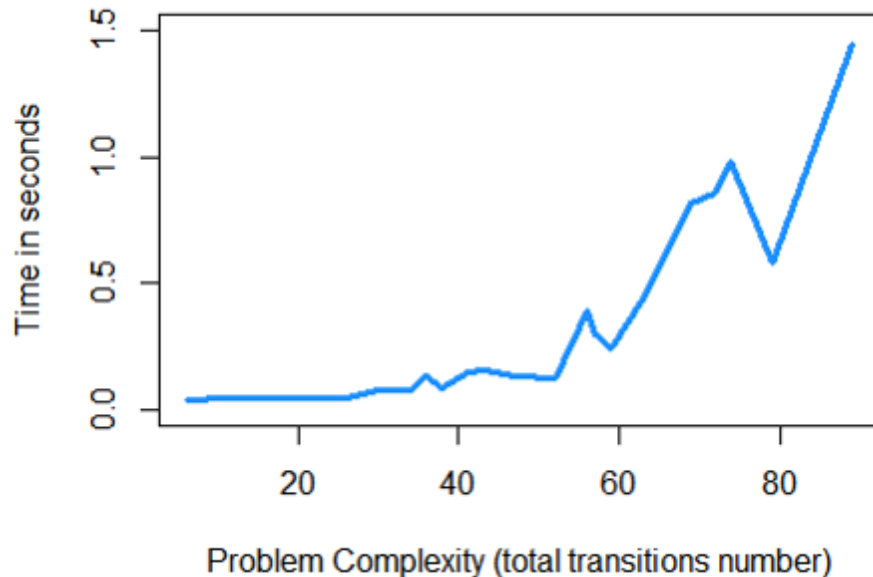
- the occurrence of complex problems is relatively rare.



% of problems solved within time  $t$ .

- Only 3% of the problems require more than 0.5 seconds to be solved, and the worst case is below 1.5 seconds.

# Analysis of the Execution Time



Trend of the Adaptation Time.

- The adaptation time ranges from **0.04** to **1.45** secs.

Service Name	(Avg) Response Time
Bla Bla Car	0.78 secs
City Bikes	0.23 secs
Google Transit	0.65 secs
Rome2Rio	1.20 secs
Travel4London	3.20 secs
Viaggia Trento	0.77 secs

Services Execution Time.

- The services response time ranges from **0.23** to **3.20** secs.



# IoT - FED

Forming and enacting Emergent  
configurations through Domain objects in the IoT

# Domain Objects for IoT

- We believe that the *design for adaptation approach* is suitable for the application in *IoT domains*, where:
  - *Smart objects* (sensors, actuators, etc) act as service providers
  - *Services* (applications) need to dynamically exchange functionalities among them and with smart objects
  - *Data* can be stored in the devices, in the cloud ...
  - *Application logic* can run on the devices, on the cloud ...

# Adjust Light Scenario

## Adjust Light (AL) application:

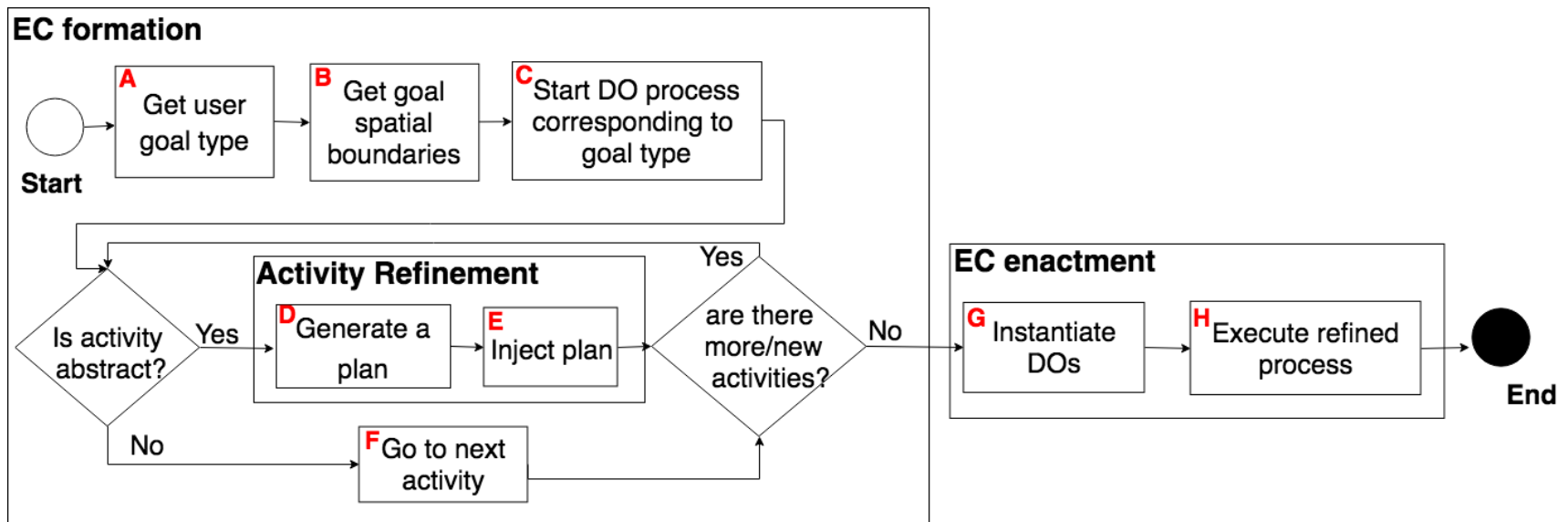
- Each room (e.g., hotel, office rooms) is equipped with several IoT things (e.g., light sensors, connected curtains and lights);
- Users requests as goal to increase/decrease the light level by using an application running on their smartphone;
- An ***Emergent Configuration*** of things (e.g., user's smartphone, a light sensor, connected curtain and light) is *formed* and *enacted* to achieve the user goal.

F. Alkhabbas, R. Spalazzese, and P. Davidsson. ***Architecting Emergent Configurations in the Internet of Things***. In Software Architecture (ICSA), 2017 IEEE International Conference on, pages 221–224. IEEE, 2017.

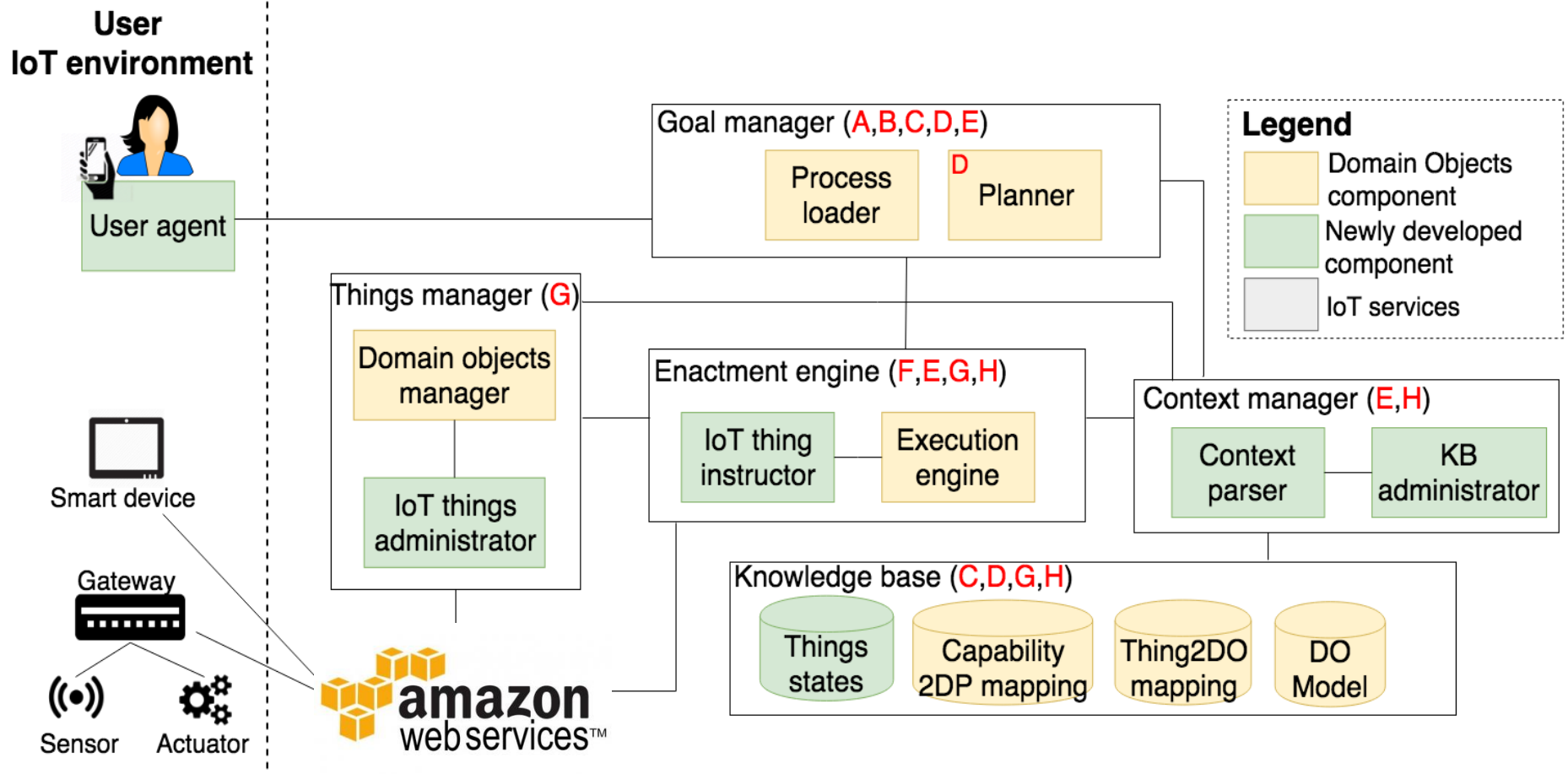
# IoT-FED Challenges & Process

## Research challenges:

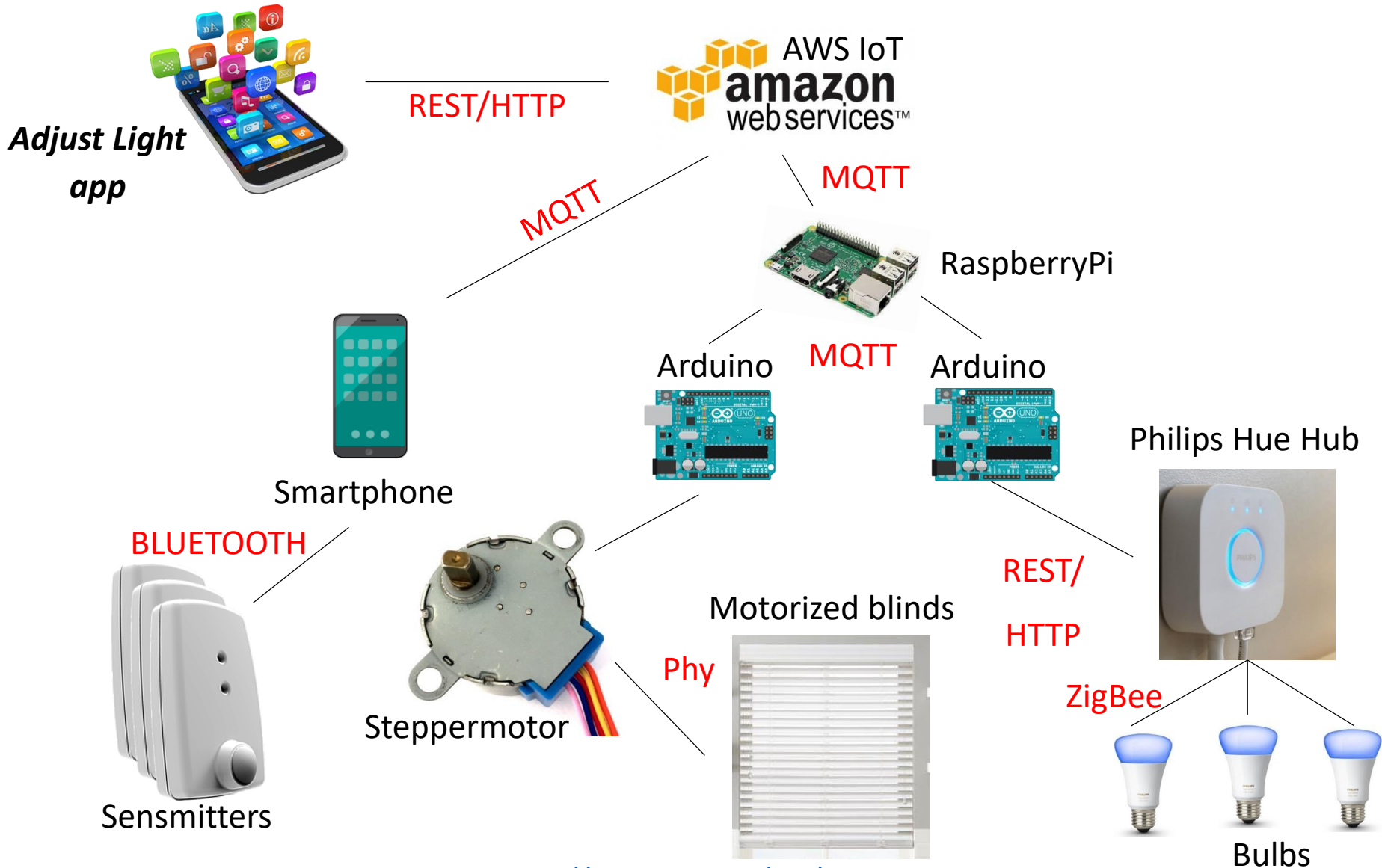
- High **heterogeneity** of *services/things* dynamically available in different and often unknown places (e.g., sensors, actuators, things APIs);
- High **complexity** and **variability** of the system behavior: it is not feasible to define a priori all the possible combinations of things to reach specific user goals (e.g., adjust light, adjust temperature);



# IoT-FED Architecture



# AL Prototype



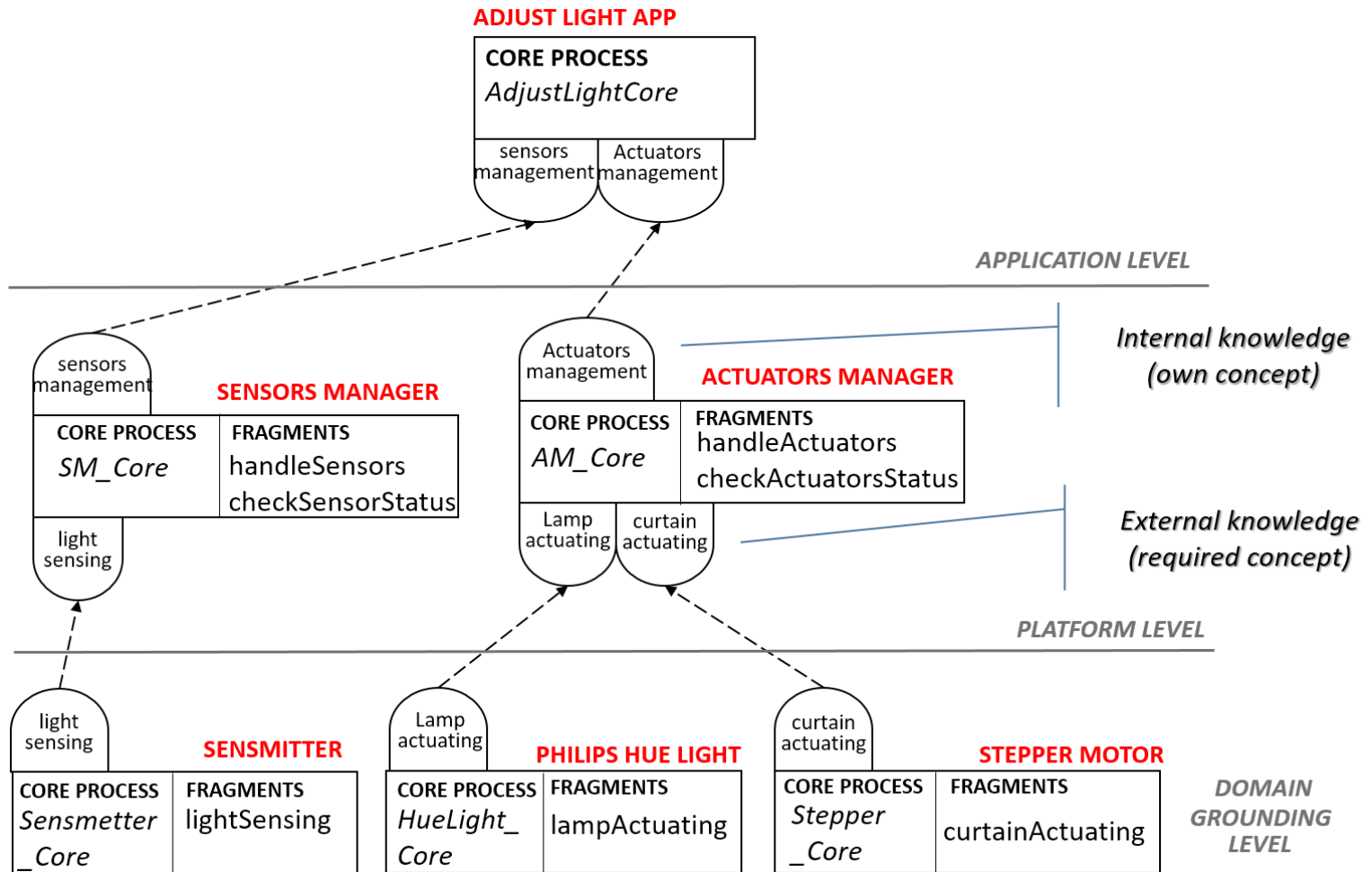
Prototype developed in the *IoTaP lab*: <http://iotap.mah.se/lab/>

Tutorial: A Design for Adaptation Framework for Self-Adaptive Systems

# IoT-FED Guideline

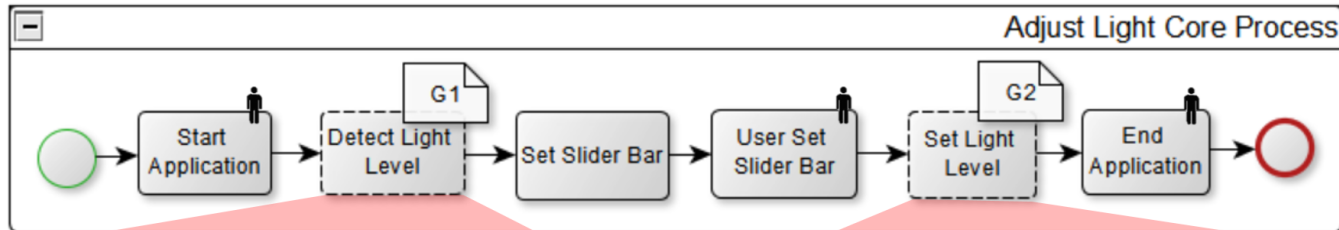
1. Register things in the AWS-IoT platform: by using templates, specifying searchable attributes automatically generated by the platform.
  1. (thing) location,
  2. capabilities,
  3. REST endpoint
2. Model things, services and applications as DOs: at each levels of the model, by specifying the *Capability2DP* and the *Thing2DO* mapping repository entries.

# AL Design with Domain Objects

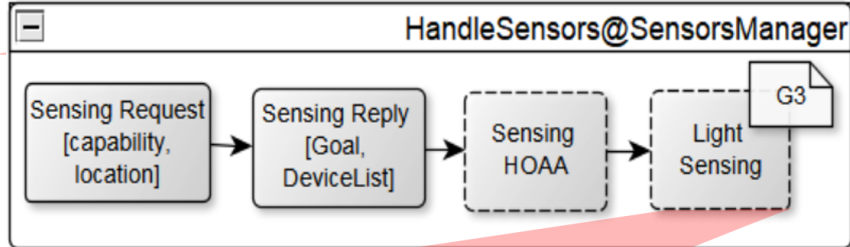




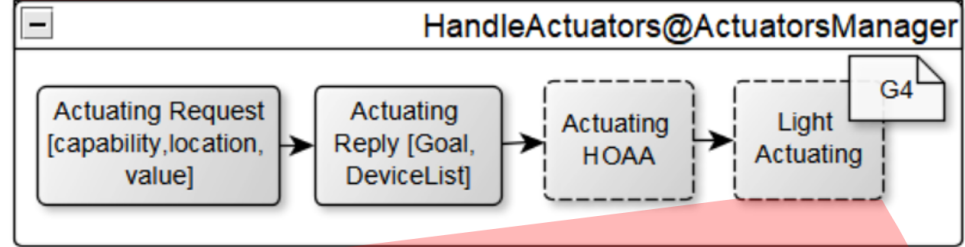
# AL Runtime Execution



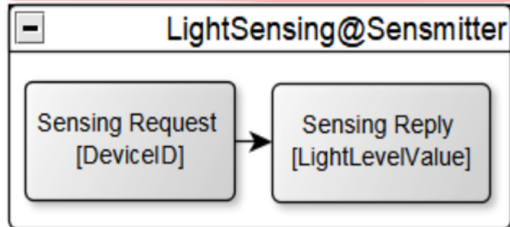
## STEP 1: PLAN FOR G1



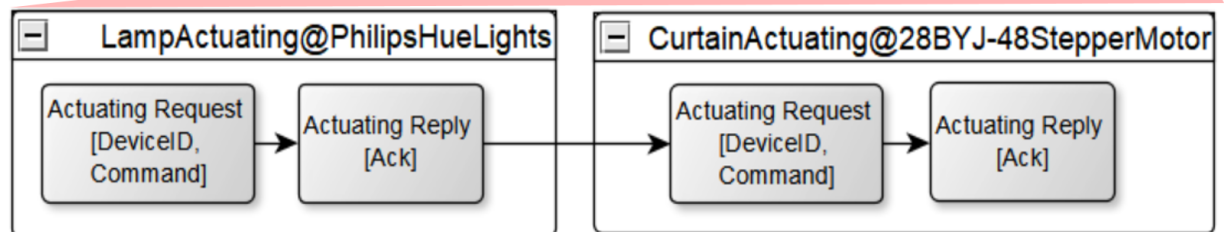
## STEP 3: PLAN FOR G2



## STEP 2: PLAN FOR G3



## STEP 4: PLAN FOR G4



## LEGEND

**G1: SM** = SENSORS DISCOVERED

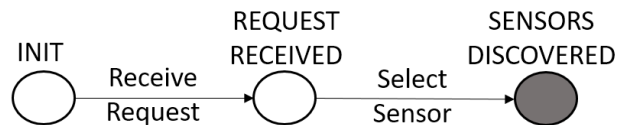
**G2: AM** = ACTUATORS DISCOVERED

**G3: LS** = LIGHT DATA SENT

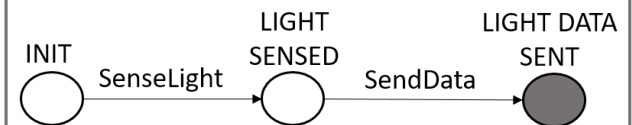
**G4: LA** = LAMP CMD EXECUTED && **CA** = CURTAIN CMD EXECUTED

## DOMAIN PROPERTIES

### SENSORS MANAGEMENT (SM)

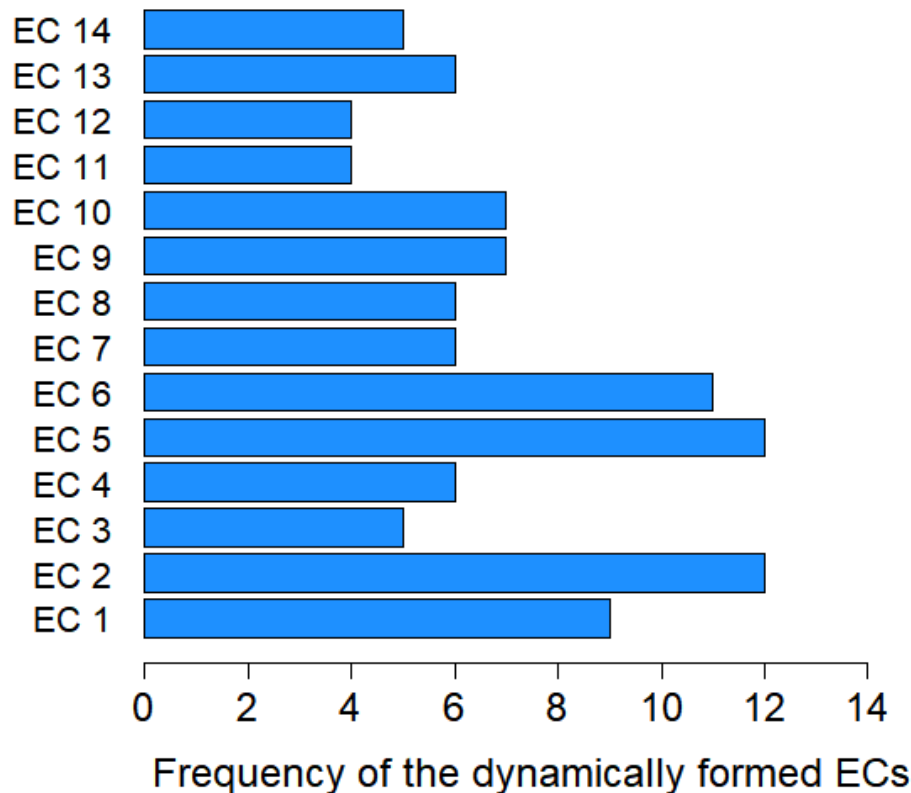


### LIGHT SENSING (LS)



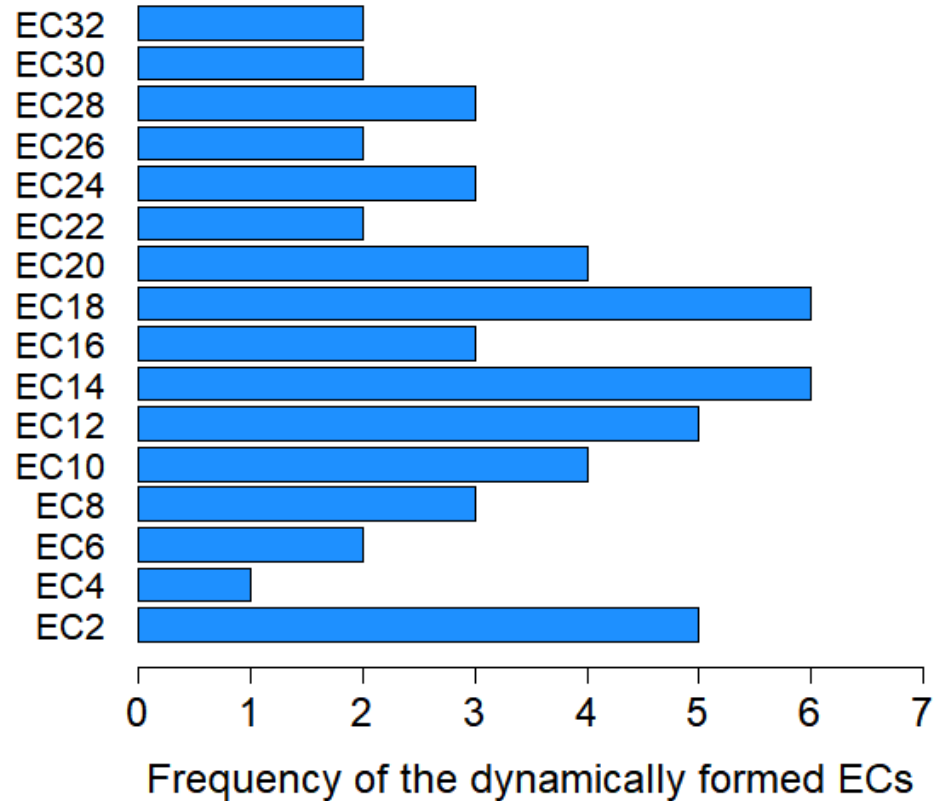
# Experimental Results

Frequency of the formed ECs after 100 runs of the AL-app.



## AL-hotel room:

- 6 devices + user's smartphone
- 14 different ECs in 100 runs



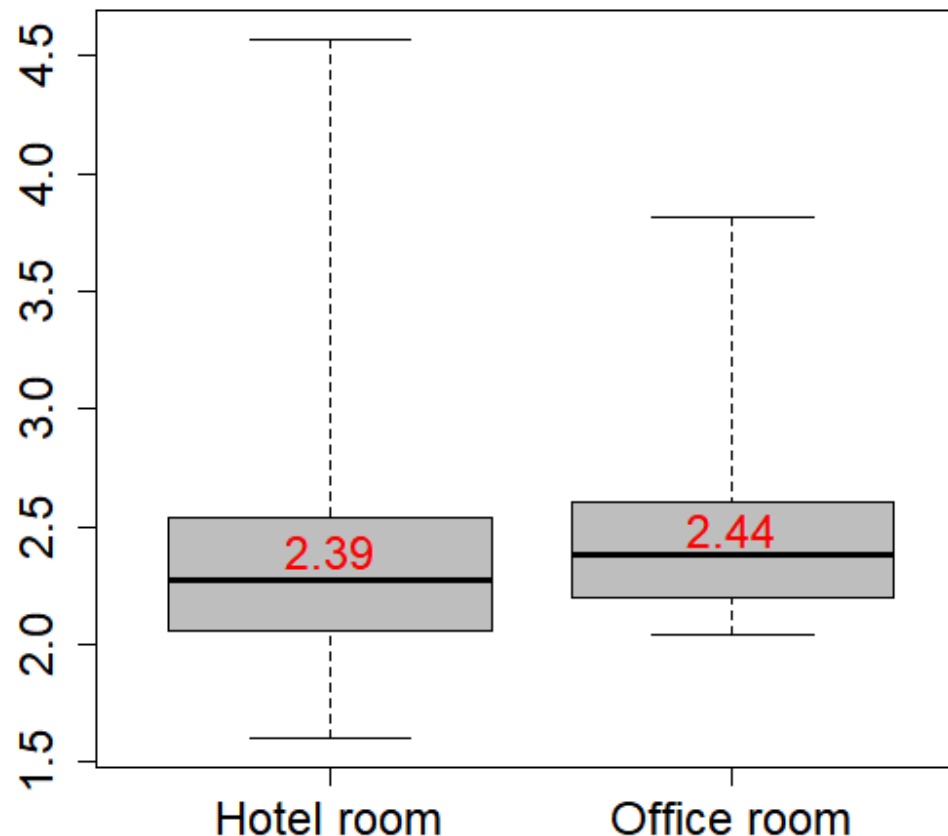
## AL-office room:

- 8 devices + user's smartphone
- 32 different ECs in 100 runs

# Experimental Results

## ECs execution time.

Average time (in secs) for forming and enacting ECs over the 100 runs of the AL app.



# Collective Adaptive Systems

# Selfish vs Collective Adaptation

- So far we have considered **Selfish Adaptation**:  
“an entity adapts by considering only its own objectives.”
- However:
  - an adaptation might be “good” for one entity but not for other entities in the same system;
  - an adaptation in an entity might trigger other adaptations in different parts of the system (chains of adaptations).

# Collective Adaptive Systems

- **Collective Adaptive Systems (CAS)** consist of a multitude of distributed agents (i.e., computers, services, devices, humans, networks) that cooperate to accomplish shared tasks.
- They actually form *socio-technical systems* whose distributed agents interact *with* and *within* the environment.
- CAS operate under constant perturbations due to unexpected changes in the environment and in the behavior of its participants.
- Engineering CAS has many benefits. However, most existing distributed systems work under an architectural model in which the **adaptation knowledge is logically centralized**.
  - i.e., the control of adaptation is exerted centrally.

# Collective Adaptive Systems

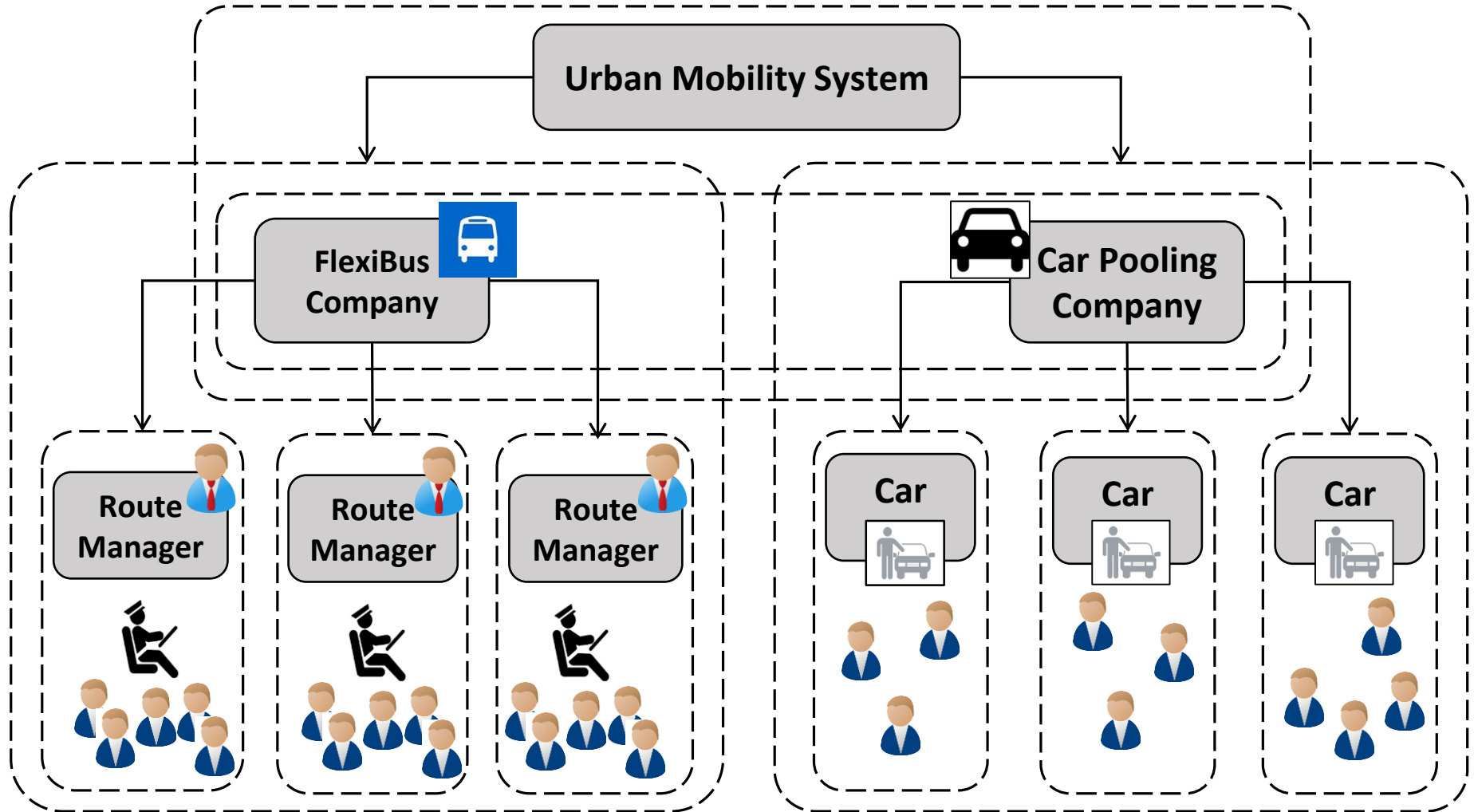
- **Collective Adaptive Systems (CAS)** consist of a multitude of distributed agents (i.e., computers, services, devices, humans, networks) that cooperate to accomplish shared tasks.
- For a collective system to be resilient,  
adaptation must be collective.
- changes in the environment and in the behavior of its participants.
- Engineering CAS has many benefits. However, most existing distributed systems work under an architectural model in which the **adaptation knowledge is logically centralized**.
  - i.e., the control of adaptation is exerted centrally.

# From Selfishness to Collective Goodness

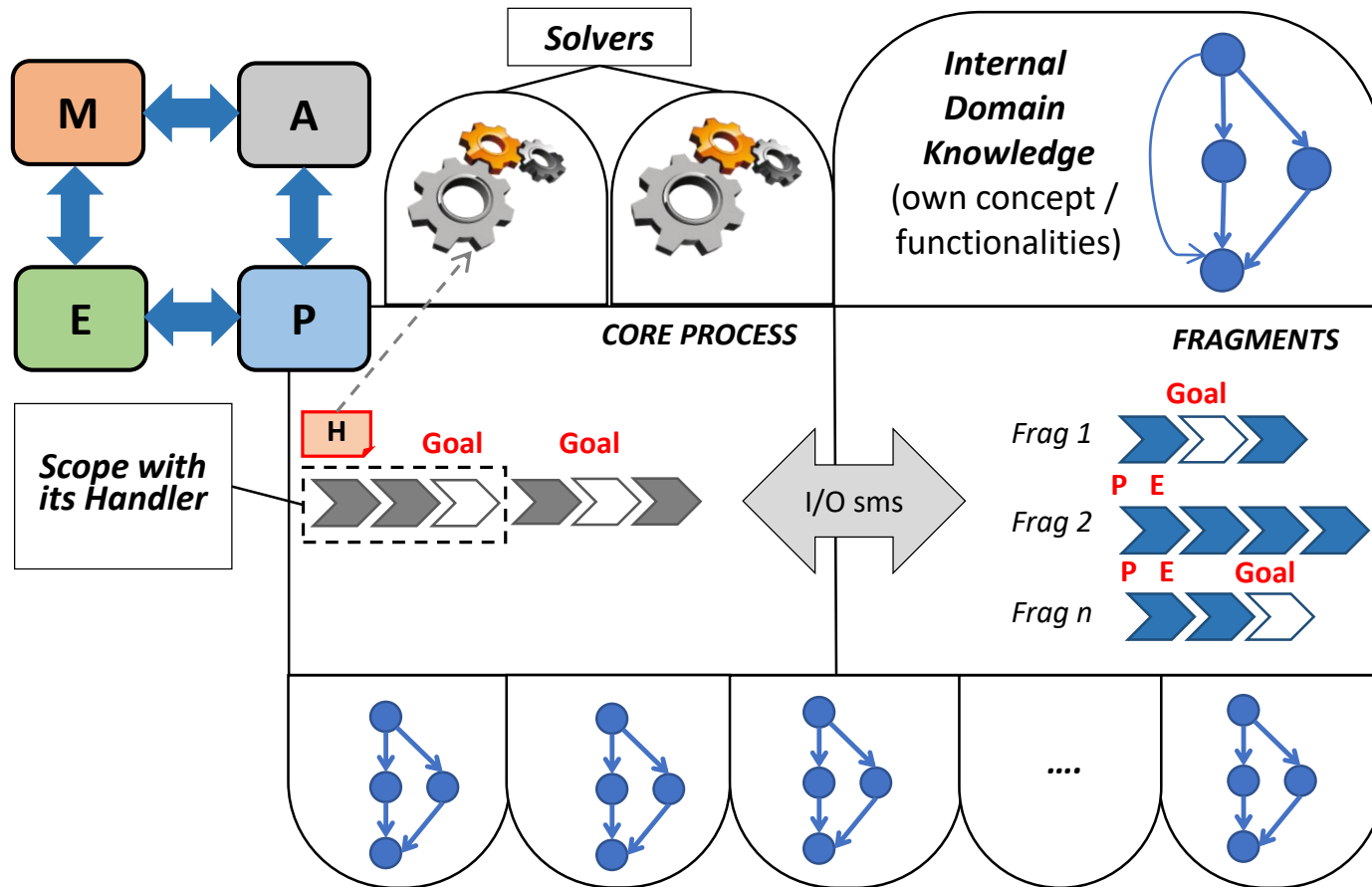
- Each CAS participant cannot adapt its behavior in complete isolation.
  - Each adaptation may negatively affect the goals of other participants (i.e., **conflicting goals**).
  - Each adaptation can compromise the whole collaboration.
- Every adaptation **must be** resolved at the scale of the whole Ensemble.
  - Adaptations **may need** to be simultaneously applied to the behavior of multiple participants.
  - Adaptations **must respect** the specific conditions that participation to the Ensemble imposes.



# Urban Mobility Collective Adaptive System



# Domain Objects for CAS



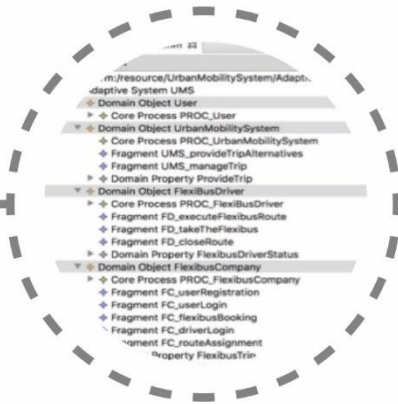
# CAS Demo

- A video featuring the DeMOCAS solutions within the ALLOW Ensembles Project –
  - <http://www.allow-ensembles.eu/>
- can be viewed at the link: [https://youtu.be/H0\\_LjptwZDg](https://youtu.be/H0_LjptwZDg)

# Tool Support

CASTLE: A Domain-Specific  
Language for modelling  
Collective Adaptive Systems.

<https://github.com/das-fbk/DeMOCAS>



*Smart Urban Mobility* System supports citizens moving in a city, exploiting in a *synergistic manner* the *heterogeneous* city transport facilities, while providing accurate, real-time and *customized mobility services* for the whole travel duration.

<https://github.com/das-fbk/CAS-DSL>

**DeMOCAS:** a framework for the execution of service-based Collective Adaptive Systems.

*ALLOW*  
*Ensembles*



A Domain Specific Language for Engineering Collective  
Adaptive Systems

# CASTIE

(Collective) Adaptive Service-Based Systems

... and Collective Adaptation



Decentralized and **collective** decision management strategy for **intra/inter ensembles** adaptation.

Multi-entities Collaboration...



**Aggregation** of autonomous but collaborative entities dynamically grouped into **ensembles**.

... and Adaptation



Advanced techniques for **dynamic adaptation** to handle context changes affecting the services execution.

Single-entity Specialization...



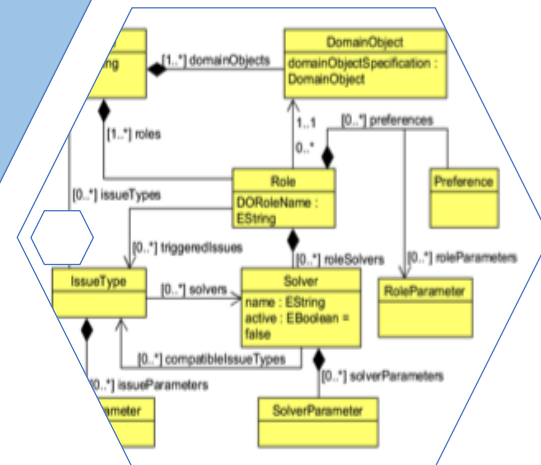
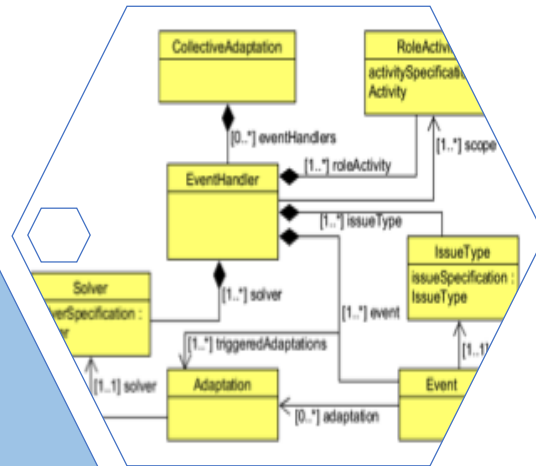
Real-time and **customized services behavior**, through **behavior specialization**, supporting the entity for the whole execution.



Collective  
Adaptation  
Strategies

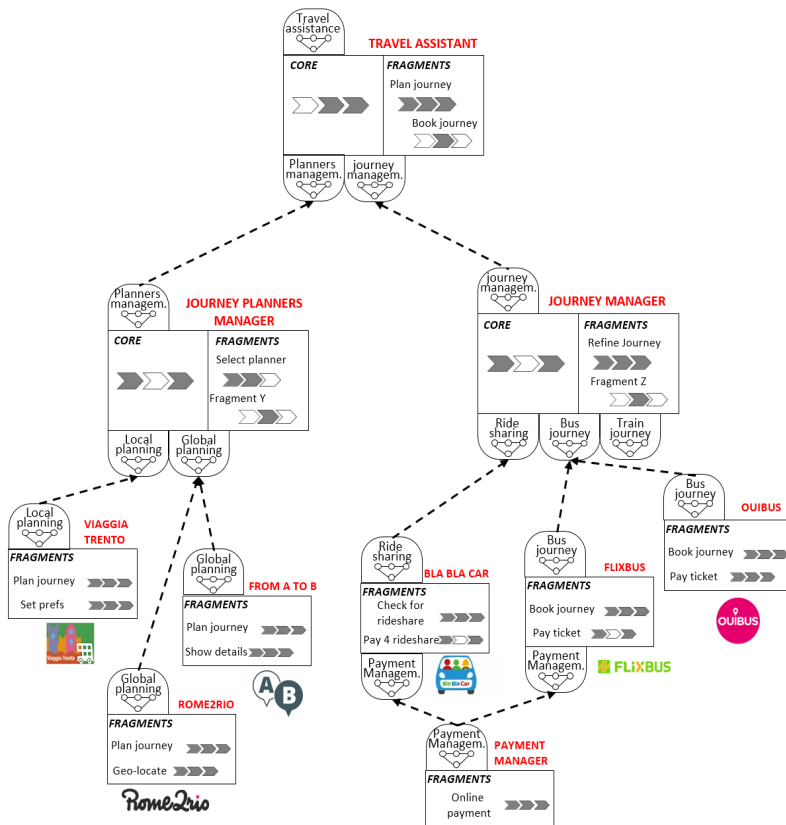
Ensembles

Adaptive  
Agents and  
Services

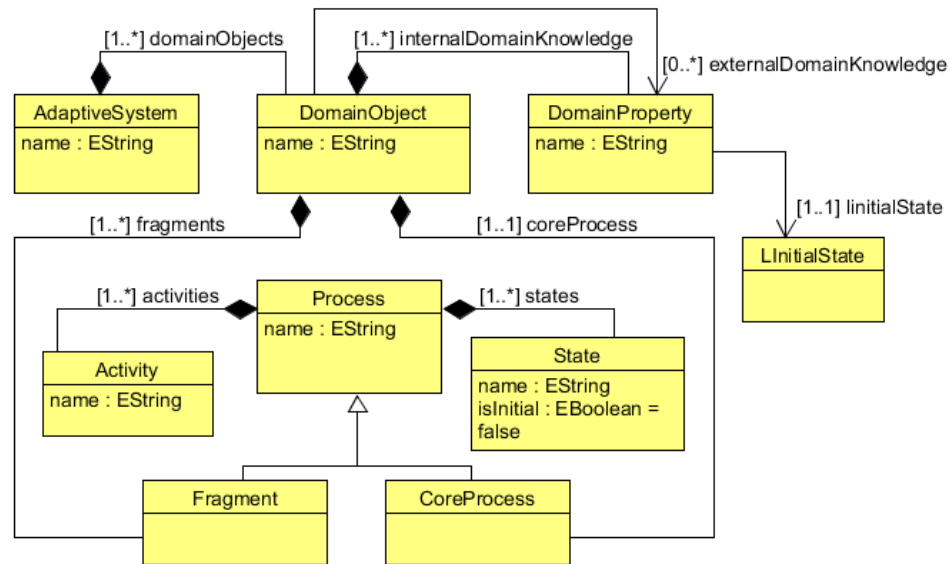


# CASTLE

An **adaptive system** is realized as a *dynamic network of domain objects*.

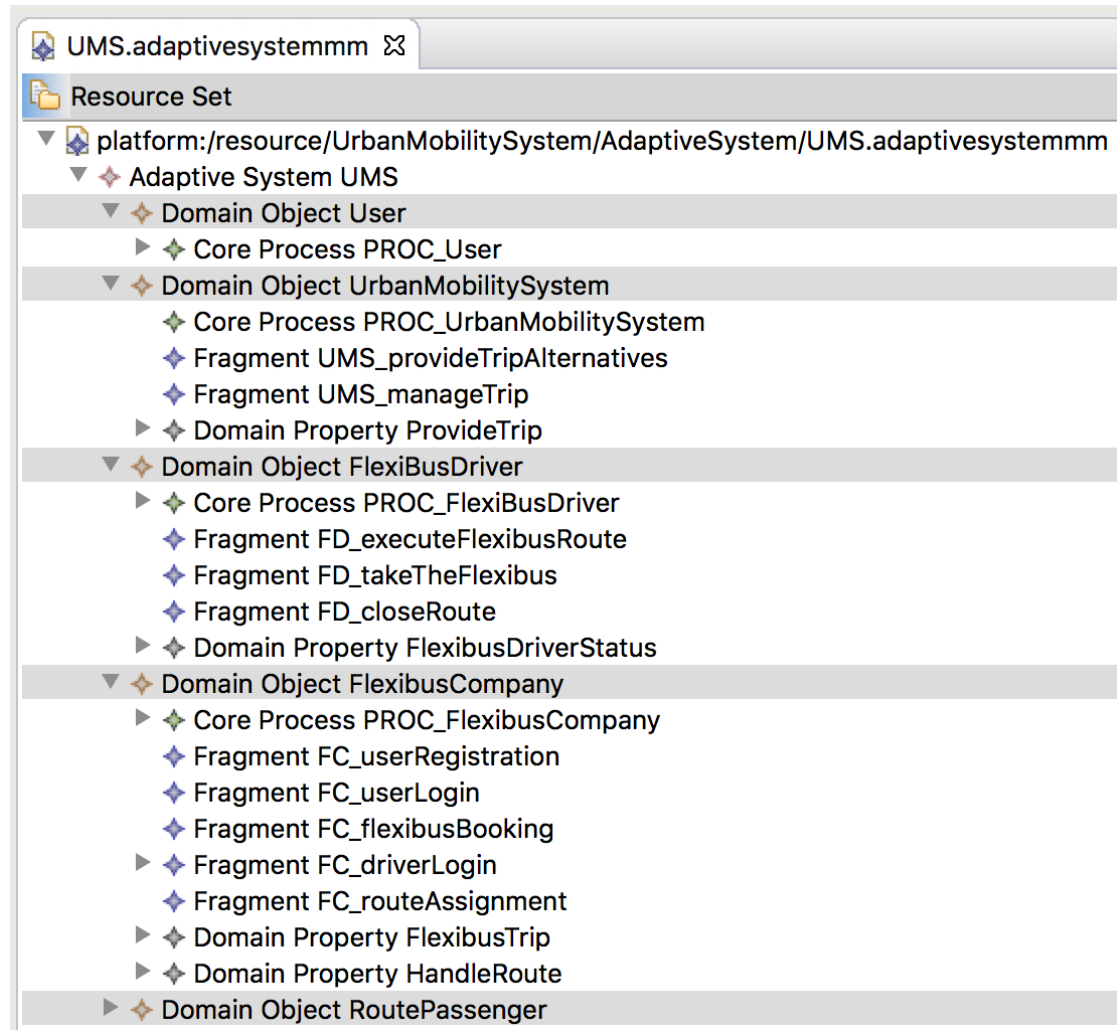
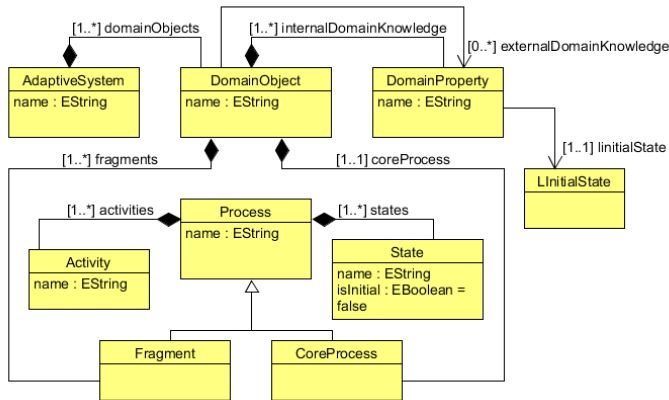


## Adaptive System Modeling Language





# Adaptive System Modeling Language



# CAStlE Demo

- We provide some demonstrative videos showing how to install<sup>5</sup> and run<sup>6</sup> the tool, whose code is available as GIT repository<sup>7</sup>.
- 5) <https://youtu.be/EkLBh1RStgQ>
- 6) <https://youtu.be/jC76LDCb078>
- 7) <https://github.com/das-fbk/CAS-DSL>

# Future Work

- *User-centricity* in service-oriented systems.
- Applying the approach to *Internet of Things domains*.
- Support for *other forms of run-time adaptation*:
  - e.g., reaction to context changes observed by monitoring the environment.
- *Service evolution*.

F. Alkhabbas, M. De Sanctis, R. Spalazzese, A. Bucchiarone, P. Davidsson, A. Marconi:  
*Enacting Emergent Configurations in the IoT through Domain Objects*. In 16th International  
Conference on Service-Oriented Computing - ICSOC 2018: To appear.

# Research Challenges

- Application of multi-agents decision making techniques for the collective adaptation;
- Extension of the collective adaptation with learning techniques for optimization purposes;
- Non-functional decision and adaptation;
- ...

# Publications List

2018

- ❑ F. Alkhabbas, **M. De Sanctis**, R. Spalazzese, A. Bucchiarone, P. Davidsson, A. Marconi: *Enacting Emergent Configurations in the IoT through Domain Objects*. In 16th International Conference on Service-Oriented Computing - ICSOC 2018: **To appear**.

2017

- ❑ A. Bucchiarone, **M. De Sanctis**, A. Marconi: *ATLAS: A World-wide Travel Assistant exploiting Service-based Adaptive Technologies*. In 15th International Conference on Service-Oriented Computing - ICSOC 2017: 561-570 - Industrial Track.
- ❑ A. Bucchiarone, A. Cicchetti, **M. De Sanctis**: *Towards a Domain Specific Language for Engineering Collective Adaptive Systems*. In 2nd eCAS Workshop on Engineering Collective Adaptive Systems – eCAS@SASO 2017: 19-26.
- ❑ A. Bucchiarone, A. Cicchetti, **M. De Sanctis**: *CASTLE: a tool for Collective Adaptive Systems Engineering*. In 11th IEEE International Conference on Self-Adaptive and Self-Organizing Systems – SASO 2017: 385-386 - Demo Track.

2016

- ❑ A. Bucchiarone, **M. De Sanctis**, A. Marconi: *Decentralized Dynamic Adaptation for Service-based Collective Adaptive Systems*. In First workshop on Adaptive Service-Oriented and Cloud Applications – ASOCA@ICSOC 2016: 5-20.
- ❑ A. Bucchiarone, **M. De Sanctis**, A. Marconi, A. Martinelli: *DeMOCAS: Domain Objects for Service-based Collective Adaptive Systems*. In 14th International Conference on Service-Oriented Computing – ICSOC 2016: 174-178 – Demo Track - **Best Demo Award**.

# Publications List

2016

- ❑ A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, P. Traverso: *Incremental Composition for Adaptive By-Design Service Based Systems*. In 23rd IEEE International Conference on Web Services - ICWS 2016: 236-243.

2015

- ❑ M. De Sanctis, K. Geihs, A. Bucchiarone, G. Valetto, A. Marconi, M. Pistore: *Distributed Service Co-evolution Based on Domain Objects*. In 12th International Workshop on Engineering Service-Oriented Applications - WESOA@ICSOC 2015: 48-63.
- ❑ A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, P. Traverso: *Design for Adaptation of Distributed Service-Based Systems*. In 13th International Conference on Service-Oriented Computing - ICSOC 2015: 383-393.

2014

- ❑ A. Bucchiarone, M. De Sanctis, M. Pistore: *Domain Objects for Dynamic and Incremental Service Composition*. At the 3rd European Conference on Service-Oriented and Cloud Computing - ESOC 2014: 62-80.

# Tutorial: A Design for Adaptation Framework for Self-Adaptive Systems

*Martina De Sanctis*

# Thank you!