ICAS 2013 The Ninth International Conference on Autonomic and Autonomous Systems

Service Components and Ensembles: Building Blocks for Autonomous Systems

- tutorial -

Nikola B. Šerbedžija

24 March 2013

Lisboa

Fraunhofer FOKUS

Outline

I. Introduction (definition, abstract, motivation, approach)
II. Requirements analyses
   - Practical examples
   - Requirements
III. Modeling
   - Approach
   - SCEL
   - Adaptation patterns
   - *X* Reasoning on system properties
IV. Deployment
   - JRESP
   - Implementation framework
V. Conclusion (discussion and further work)

case studies

Fraunhofer
FOKUS

I. Definition: Autonomous

au·ton·o·mous [aw-ton-uh-muhs]

- adjective
  1. Government .
     a. self-governing; independent; subject to its own laws only.
     b. pertaining to an autonomy.
  2. having autonomy; not subject to control from outside; independent: a subsidiary that functioned as an autonomous unit.
  3. Biology .
     a. existing and functioning as an independent organism.
     b. spontaneous.
- ***Origin:*** Greek autónomos with laws of one's own, independent, equivalent to auto- …

I. Definition Autonomous systems

- *Within the Internet, an **Autonomous System** (**AS**) is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet.*

- *"Autonomous systems represent the next great step in the fusion of machines, computing, sensing, and software to create intelligent systems capable of interacting with the complexities of the real world. Autonomous systems are the physical embodiment of machine intelligence".*

- *Autonomous systems with multiple sensory and effector modules face the problem of coordinating these components while fulfilling tasks such as moving towards a goal and avoiding sensed obstacles.*

- *Deals with adaptation, intelligence, sensing, robotics, agent technology, self-organization, dynamic and independent behavior, awareness, Pervasive services and mobile computing, self-management context-aware systems, no human intervention.*

AUTSY: Theory and Practice of Autonomous Systems

- Design, implementation and deployment of autonomous systems; Frameworks and architectures for component and system autonomy; **Design methodologies for autonomous systems; Composing autonomous systems; Formalisms and languages for autonomous systems; Logics and paradigms for autonomous systems**; Ambient and real-time paradigms for autonomous systems; Delegation and trust in autonomous systems; Centralized and distributed autonomous systems; Collocation and interaction between autonomous and non-autonomous systems; Dependability in autonomous systems; Survivability and recovery in autonomous systems; Monitoring and control in autonomous systems; Performance and security in autonomous systems; Management of autonomous systems; Testing autonomous systems; Maintainability of autonomous systems

TAAS

ACM Transactions on
Autonomous and Adaptive Systems

- *Many current Information and Communications Technology (ICT) systems and infrastructure, such as*
  - *the Web, Clouds, Grids and Enterprise Datacenters, Peer-to-Peer Systems, Social and Urban Computing Systems, Cooperative Robotic Systems, Distributed Service Systems, and Wireless and Mobile Computing Systems,*
- *have the characteristic of being*
  - ***decentralized, pervasive, and composed of a large number of autonomous entities.***
- *Often systems deployed on such infrastructure need to run in highly dynamic environments, where physical context, social context, network topologies and workloads are **continuously changing**. As a consequence, autonomic and adaptive behaviors become necessary aspects of such systems.*

EU, FP7 Awareness Initiative: Challenges

- 101 Awareness Challenges
- 72. To have good and sustainable test bed and test environment for experiments. Nenad Stojr
- 71. Introducing economic models. Ivova Brandic
- 70. Monitoring of large scale adaptive infrastructures and mobile devices. Ivova Brandic
- 69. To disambiguate the awareness concepts. Ramana Reddy
- 68. **Checking, requirements, model, verification and validation at runtime.** Hausi Muller
- 67. Representation and synchronization of requirements at runtime. Nelly Bencomo
- 66. To address real problems by means of exemplars. Luciano Baresi
- 65. To have intelligent runtime environments that support adaptation, keeping and managing t
- 64. To exploit a graphical language in order to achieve automatic generation of engines. Tom I
- 63. To have an appropriate mathematical base. Franco Bagnoli
- 62. To enable adaptive systems to learn online. Peter Lewis
- 61. How to describe and to compare information? Yvonne Bernard
- 60. **How to ensure safety and correctness?** Manuele Brambilla
- 59. How to manage the relationship between individual and group levels? Carlo Pinciroli
- 58. How to achieve adatpivity at runtime? Martin Wirsing
- 57. How to engineer decision systems? Henry Bensler
- 56. How to map raw data to knowledge? Emil Vassev
- 55. Dealing with high and low levels of contexts. Wei Dai
- 54. Considering sociological aspects besides technical aspects. Francois Toutain
- 53. Letting different systems interoperate and collaborate. Guillame Dugue
- 52. How to measure the level of awareness? E.g. the number of variables AND the algorithm t
- 51. Measuring and finding metrics for the different kinds of awareness. Franco Zambonelli
- 50. The difficulty of writing precise requirements about flexibility. Peter Lewis
- 49. The difficulty of proving all the properties of an emergent system. Jose Luis Fernandez
- 48. How to improve the communication between local and global systems in swarm robotics?:
- 47. Monitoring and controlling emergent properties and specifying and controlling adaptation:

Fraunhofer FOKUS

I. Abstract

- Developing **autonomous** systems requires **adaptable** and **context aware** techniques.
- The approach described here decomposes a complex system into **service components** – functionally simple **individual entities** enriched with local **knowledge** attributes.
- The internal components’ knowledge is used to dynamically construct **ensembles** of service components.
- Thus, ensembles capture **collective behavior** by grouping service components in many-to-many manner, according to their communication and operational/functional requirements.
- Linguistic constructs and software tools have been developed to support modeling, validation, development and deployment of autonomous systems. A strong pragmatic orientation of the approach is illustrated by a concrete application.

Keywords: *Engineering Complex Autonomous Systems, Awareness in software, Adaptive components, Reasoning about system properties, Case studies (Swarm robotics, Cloud Computing, E-mobility).*

Fraunhofer FOKUS

1. www.ascens-ist.eu/
2. http://www.aware-project.eu/
3. http://www.fokus.fraunhofer.de/de/quest/projekte/laufende_projekte/ascens/index.html

I. Motivation - System Needs

- Nowadays, we deal with distributed (software intensive) systems with a massive number of nodes with highly autonomic behavior still having harmonized global utilization of the overall system. Some features:
  - Self-awareness and adaptation while operating in unknown environments or reducing management costs.
  - Maintenance of major properties even when adapting, e.g., mutual exclusion, fault tolerance, optimal energy level, distributed access, etc.
- Grand challenge in software engineering – how to organize, program and reason about these systems
- Our everyday life is dependent on new technology which poses extra requirements to already complex systems:
  - we expect systems to adapt to changing demands over a long operational time and
  - we need reliable systems whose properties can be guaranteed
  - to optimize their energy consumption .

I. Approach

One engineering response to these challenges is to structure software intensive systems in ensembles of simple service components featuring autonomous and self-aware behavior.

- Modeling:
  - provide formalisms,
  - linguistic constructs and
  - programming tools

  featuring autonomous and adaptive behav…
- Integration of:
  - Functional-,
  - Operational- and
  - Energy- awareness

  to provide autonomous behavior with reduced energy consumption!

Awareness is the state or ability to perceive, to feel, or to be conscious of events, objects, or sensory patterns.

Service Components and Ensembles

ascens

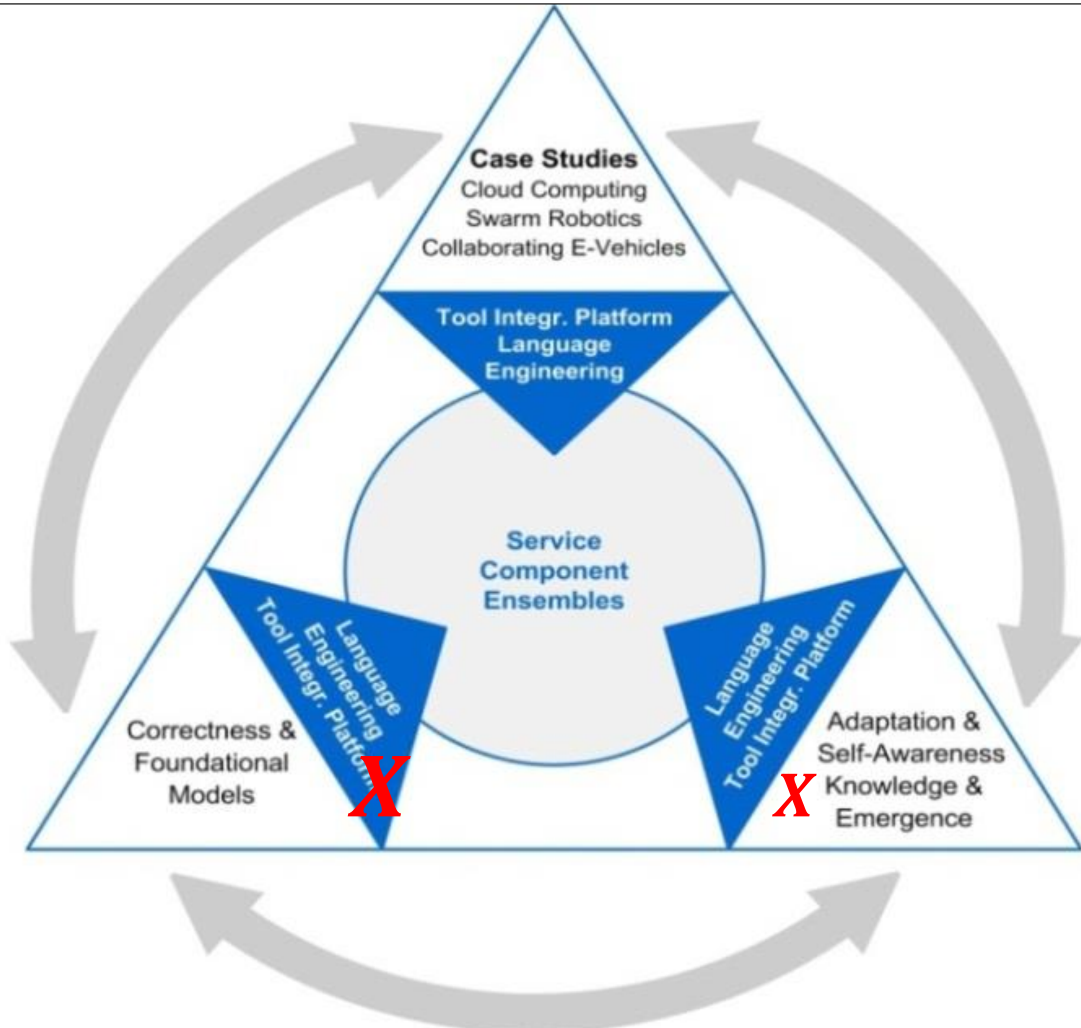
ensembles

- achieve an overall system's goal
- have a massive number of nodes
- operate in open and non-deterministic environments
- are built from self-aware components
- adapt dynamically to new conditions

engineering ensembles

- language for autonomous behavior
- X knowledge representation of self-aware components
- mechanisms for adaptation
- X verification using formal methods
- set of tools and tool integration platform

Overview Approach



II. Requirements Analyses

To explore the system requirements, three complex application domains are closely examined:

Swarm robotics

Cloud computing

E-mobility

1. www.ascens-ist.eu/
2. http://www.aware-project.eu/
3. http://www.fokus.fraunhofer.de/de/quest/projekte/laufende_projekte/ascens/index.html

II. Application Domain

- E-mobility is a vision of future transportation by means of electric vehicles network allowing people to fulfill their individual mobility needs in an environmental friendly manner (decreasing pollution, saving energy, sharing vehicles, etc.)
- Cloud computing is an approach that delivers computing resources to users in a service-based manner, over the internet, thus re-enforcing sharing and reducing energy consumption).
- Swarm robotics as a multi-robot system that through interaction among participating robots and their environment can accomplish a common goal, which would be impossible to achieve by a single robot.

At a first glance electric vehicular transportation, distributed computing on demand and swarm robotics have nothing really in common!

II. Major Application Characteristics

For modeling purposes the following characteristics are observed:

- Single entity (service components)
  - Individual goal
- Grouping (ensembles)
  - Global goal
- Self-awareness
- Adaptation
- Autonomous and collective behavior
- Optimization and
- Robustness

II. Common Characteristics

Comm. features	**Swarm Robotics**	***Cloud computing***	***E-Mobility***
Single entity	Individual robots	Computing resource	Driver, vehicle, park place, charging station
Individual goal	Performing certain task	Efficient execution	Individual route plan, optimize energy, …
Ensemble	A group of cooperative robots with a same task	application, cpu pool, …	Common rout, free vehicles, free park places, etc
Global goal	Coordinated and autonomous behavior	Resource availability, optimal throughput, …	Travel and journey optimization, low energy
Self-awareness	Knowledge about own capabilities	Available resources; computational requirements, …	Awareness of own state and restrictions
Adaptation	According to environmental changes, other entities, goals, etc	According to available resources	According to traffic, individual goals, infrastructure, resource availability
Autonomous vs. collective behavior	Optimal coordination of single entities in joint endeavor	Decentralized decision making, global optimization	Reaching all destinations in time, minimizing costs
Optimization	Time, energy, performance	Availability, computational task execution	Destination achievement in time, vehicle/infrastructure usage
Robustness	Hardware failures, sensory noise, limited sensory range and battery life	Failing resources	Range limitation, charging battery infrastructure resources

II. Common Characteristics (cont.)

This set of common features serve as a basis for modeling of such systems leading to a generic framework for developing and deploying complex autonomic systems.

Four major (autonomic system) principles are:

- Knowledge (facts about self- and surrounding)
- Adaptation (dynamic and long-term self-modification to changing surroundings)
- Self-awareness (re-examination of own state)
- Emergence (simple system elements construct complex entities).

Modeling

- Control systems for the three application domains have many common characteristics: they are highly collective, constructed of numerous independent entities that share common goals. Their elements are both autonomous and cooperative featuring a high level of self-awareness and self-expressiveness.
- A control system built out of such entities must be robust and adaptive offering maximal utilization with minimal energy and resource use.

Fraunhofer FOKUS

Modeling: Service Components and Ensembles

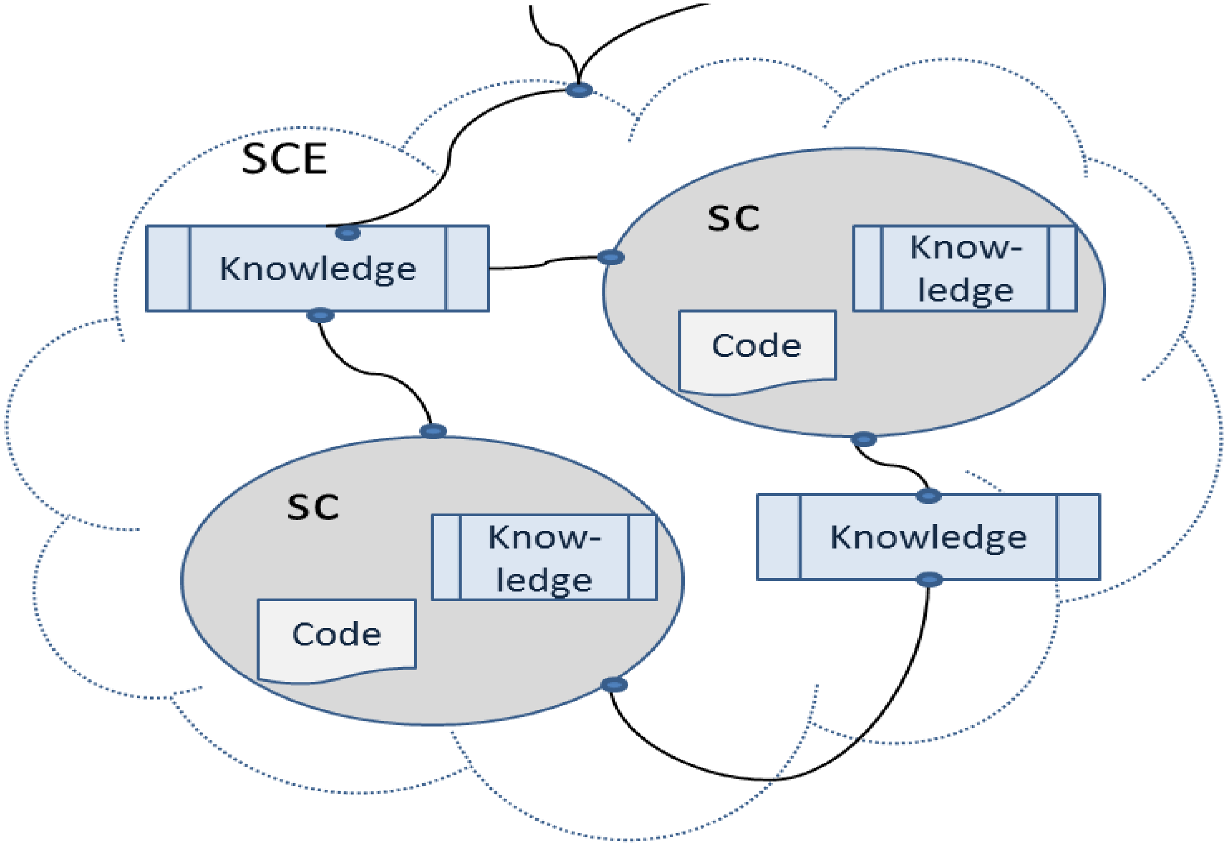
A complex system is decomposed in

- SCs - service components - major individual entities,
- SCEs - service component ensembles - composition structures that reflect communication

Further properties:

- SCs – are single system entities that have their requirements and functionality, usually representing their individual goals,
- SCEs –are collections of service components usually representing collective system goals (as means to dynamically structure independent and distributed system entities).

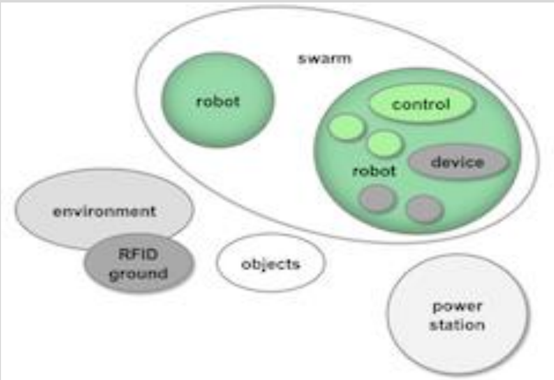
Modeling: Service Components and Ensembles



Case Studies

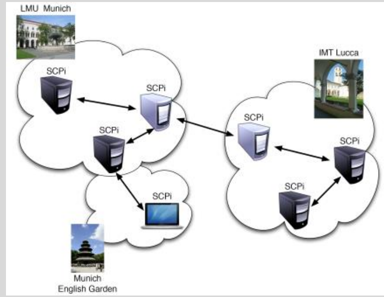
Ensembles of self-aware robots

used to perform the most dangerous activities, for example in a disaster recovery scenario: find and remove a dangerous object in presence of obstacles.



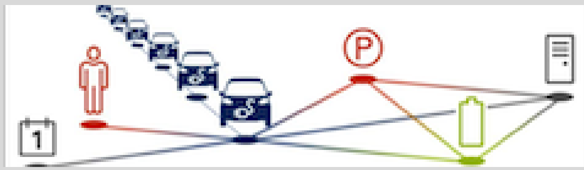
Resource ensembles as science clouds

science cloud platform as a Platform as a Service (PaaS) solution. One scenario considers that a science cloud platform goes offline, which means the applications there has to be made available oat one or more of other nodes

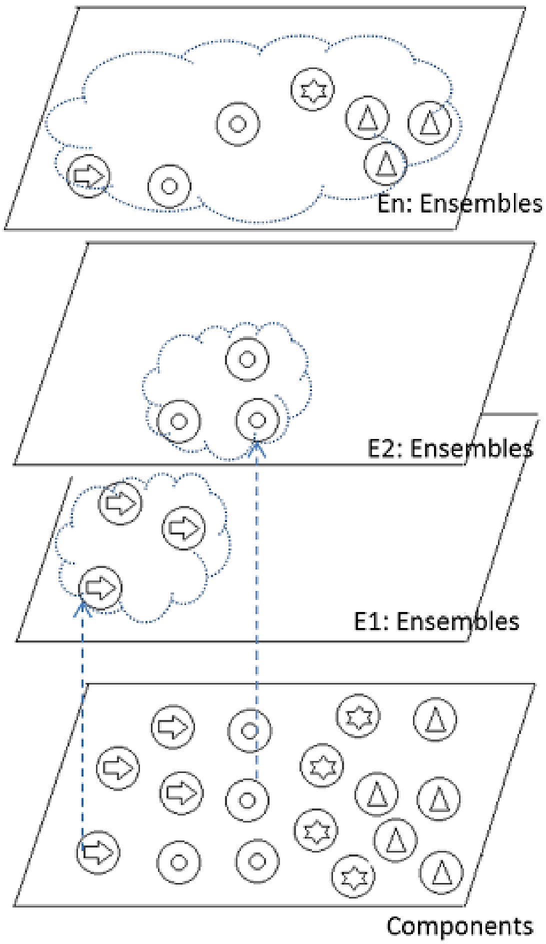


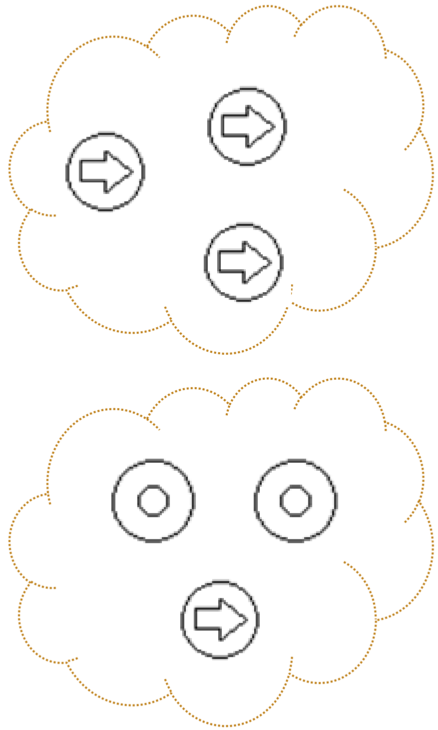
Ensembles of cooperative vehicles

for providing a user with a seamless daily travel plan, a sequence of destinations with possibly different travel modes and resource requirements



Ensembles Building



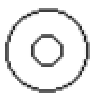
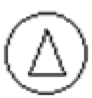


- Ensemble can be made of same service component types with common goal
- Ensemble can be made of different service component types with matching goals

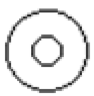
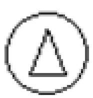
Goals can be defined by any function or predicate

Fraunhofer FOKUS

Swarm Robotics

Symbol	SC: Service Component	Knowledge	Goals
	Obstacles/ bricks	Dimension, shape, weight	Protecting shape construction
	robots with a grip	Movements, grip capabilities, battery state	Cary the object for one to another location
	Targets	Location, weight, shape	Movement
	foraging robots	Movements, battery state	Finding objects, Information propagation

Cloud computing

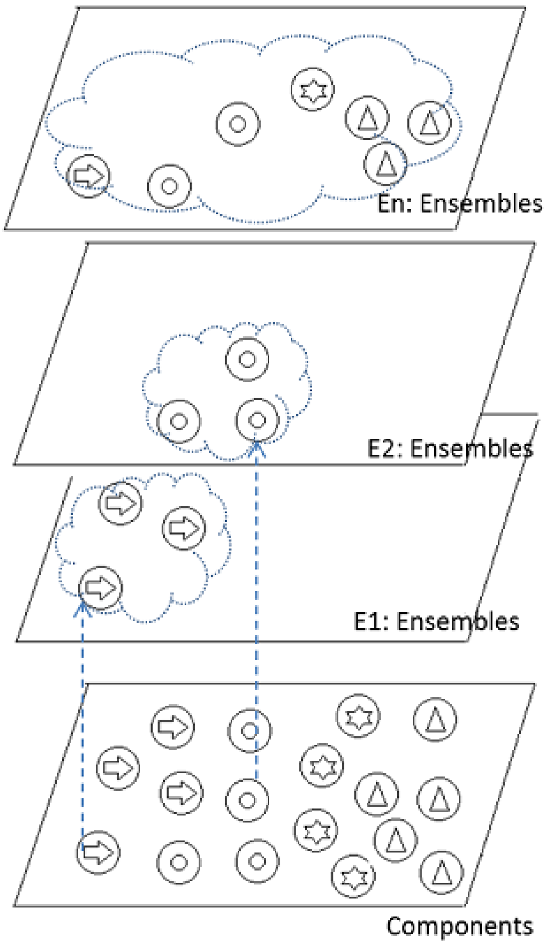
Symbol	SC: Service Component	Knwledge	Goals
	User applications	the requests for execution (in terms of CPU, minimal space, etc.).	Efficient execution.
	Remote computer CPUs	processing capabilities and a current utilization	Optimal utilisation
	Local memory	Capacity, current occupacy	Balanced use
	Local application services	available appis at the local computer	Appies “advertising”

Fraunhofer FOKUS

E-mobility

Symbol	SC: Service Component	Knwledge	Goals
	Users	Route plan	to reach different places in a given time.
	E-vehicles	occupancy and the battery state	to serve users plans, optimize energy consumption
	Charging stations	Capacity/ Reservation plan	optimize its use (high throughput)
	Park places	Capacity/ Reservation plan	optimize its use

Modeling Examples (Ensembles)



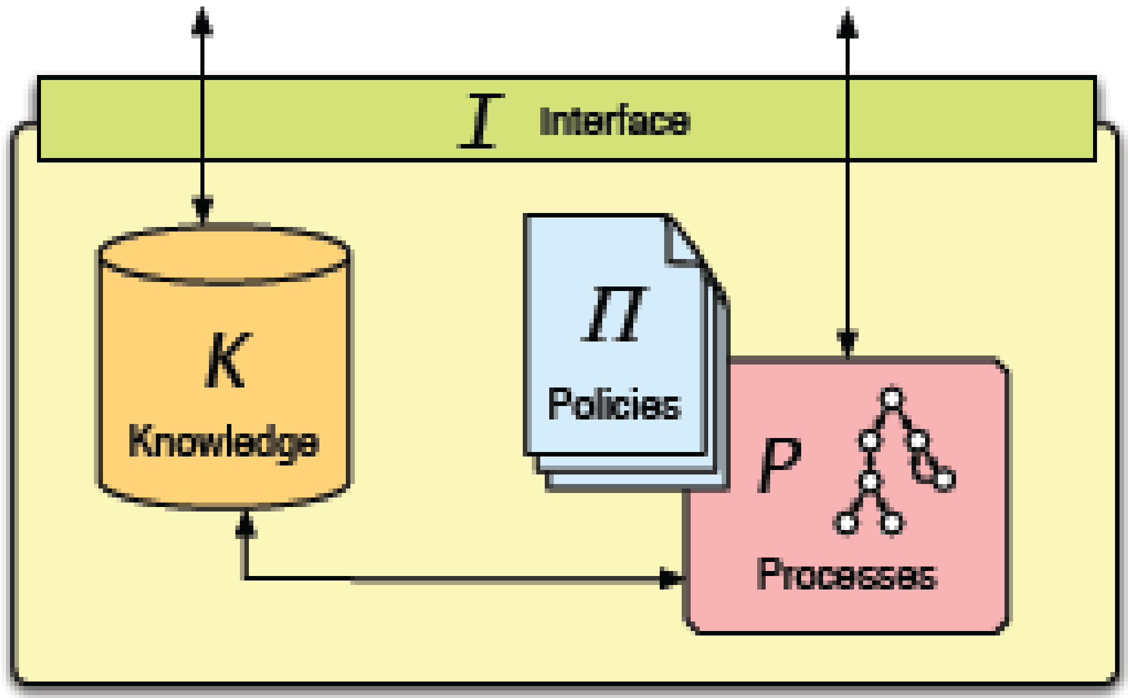
E-Mobility	Cloud Computing	Swarm Robotics
• A user, 2 vehicles, 1 charging station and 3 parklaces	• A user application, 2 ren computers, with local m of appropriate size and supporting apples.	• A task: one obstacle, two robots, one target and three foraging robots
• 3 vehicles that are ava for sharing	• 3 remote computers	• 3 free robots with a grip
• 3 users ready to share vehicles	• 3 different applications v similar processing and memory requirements	• 3 obstacles to be removed
• 4 basic service components: users, vehicles, charging stat and park places	• 4 basic service compon users applications, remo CPUs, local memory an	• 4 basic service components: obstacles, robots with a grip, targets, foraging robots

III SCEL: Modeling language

- A set of programming abstractions that permit to directly represent behaviors, knowledge and aggregations according to specic policies, and to support programming self- and context-awareness, and adaptation.
- The main novelty of the language is the way sets of partners are selected for interaction. The single component has the possibility of directly identifying the partners of a communication but can also select them by exploiting the notion of attribute-based communication.
- Ensembles are formed according to predicates over interfaces' attributes, representing specific properties, like spatial coordinates or group memberships, and properties that they can guarantee like security, trust level or response time.

SCEL: Modeling language (cont.)

- Behaviors describe how computations progress.
- Interface provides a set of attributes characterising the component itself
- Knowledge is represented through items containing either application data or awareness data
- Policies control and adapt the actions of the different components in order to guarantee achievement of specific goals or satisfaction of specific properties
- Attribute based communication



•Ensembles are formed according to predicates over attributes

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

III SCEL Syntax

- Systems: S ::= C | S1 ǁ S2 | (*vn*)S
- Components: C ::= *I*[*K*,∏,*P*]
- Processes: P ::= **nil** | a.P | P1 + P2 | P1[P2] | X | A(p)
- Actions: a ::= **get**(T)@c | **qry**(T)@c | **put**(t)@c | **new**(*I*,*K*,∏,*P*)
- Targets: c ::= *n* | *x*| **self** | P | *I.p*
- To execute SCEL programs, the jRESP framework has been developed. This is a Java runtime environment providing means to develop autonomic and adaptive systems programmed in SCEL [*].

M. Loreti. jRESP: a run-time environment for scel programs. Technical Report (September 2012) http://rap.dsi.unifi.it/scel/.

SCEL Processes

P ::= nil | a.P | P1 + P2 | P1[P2] | X | A(p)

Processes are the active computational units. Each process is built up from the inert process **nil** via action prefixing **(a.P),** nondeterministic choice **(P1 + P2)**, controlled composition **(P1[P2]),** process variable (X), and parameterized process invocation A(p).

The construct P1[P2] abstracts the various forms

- of parallel composition commonly used in process calculi. Process variables can support higher-order communication, namely the capability to exchange (the code of) a process, and possibly execute it, by first adding an item containing the process to a knowledge repository and then retrieving/withdrawing this item while binding the process to a process variable.

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

SCEL Actions

- Actions and targets. Processes can perform five different kinds of actions:
  - get(T)@c, qry(T)@c and put(t)@c

  are used to manage shared knowledge repositories by withdrawing/retrieving/adding information items from/to the knowledge repository c. These actions exploit templates T as patterns to select knowledge items t in the repositories. They heavily rely on the used knowledge repository and are implemented by invoking the handling operations it provides.

  - fresh(n)

  introduces a scope restriction for the name n so that this name is guaranteed to be fresh, i.e. different from any other name previously used.

  - new(I/[*K*,∏,*P*])

  creates a new component *I*[*K*,∏,*P*]

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

SCEL Targets

$$c ::= n \mid x \mid \mathrm{self} \mid P \mid I.p$$

Different entities may be used as the target c of an action. Component names are denoted by n, n0, . . . , while variables for names are denoted by x, x0, ...

The distinguished variable **self** can be used by processes to refer to the name of the component hosting them.

The target can also be a predicate P or the name p of a predicate exposed as an attribute in the interface I of the component that may dynamically change.

A predicate could be a boolean-valued expression obtained by applying standard boolean operators to the results returned by the evaluation of relations between attributes and expressions.

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

SCEL Systems and Components

- Systems aggregate components through the composition operator $\_ \parallel \_$. It is also possible to restrict the scope of a name, say n, by using the name restriction operator (vn)_ .
- Thus, in a system of the form $S_1 \parallel (\nu n)S_2$, the effect of the operator is to make name n invisible within S1.

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

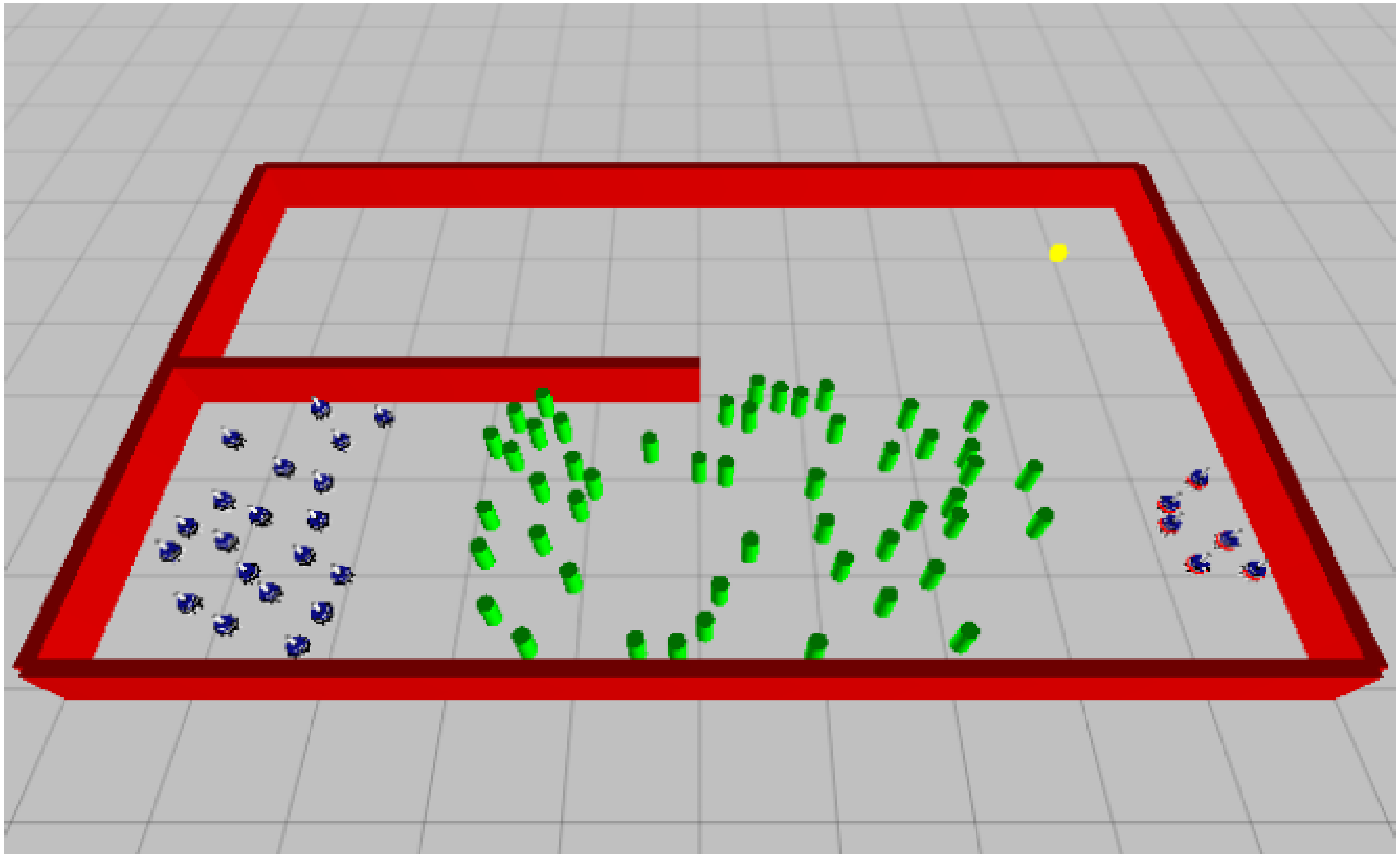
Building Ensembles

- Thus, actions put(t)@n and put(t)@P give rise to two different primitive forms of communication: the former is a point-to-point communication, while the latter is a sort of group-oriented communication.
- The set of components satisfying a given predicate P used as the target of a communication action can be considered as the ensemble with which the process performing the action intends to interact.
- For example, the names of the components that can be members of an ensemble can be fixed via the predicate
- $\mathcal{I}.id \in \{n, m, o\}$
- $\mathcal{I}.active = yes \wedge \mathcal{I}.batteryLevel > low.$

Fraunhofer FOKUS

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

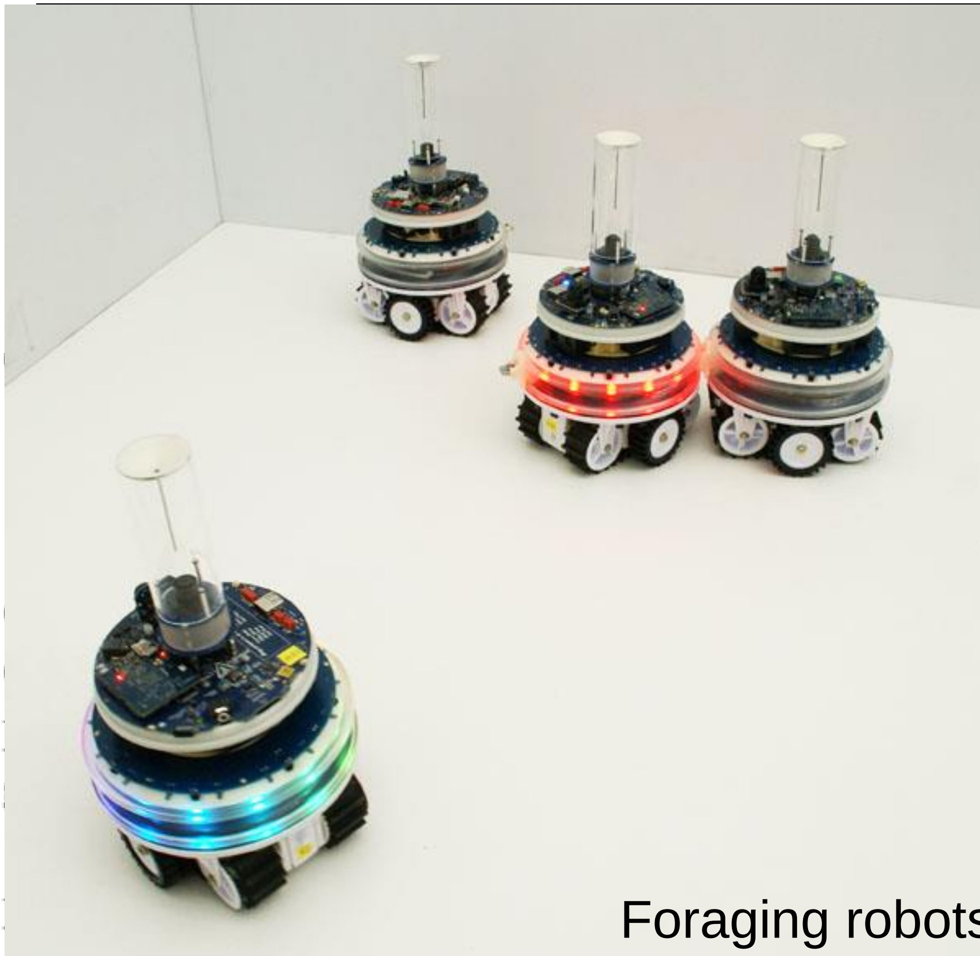
SCEL Modeling Example: Swarm Robotics



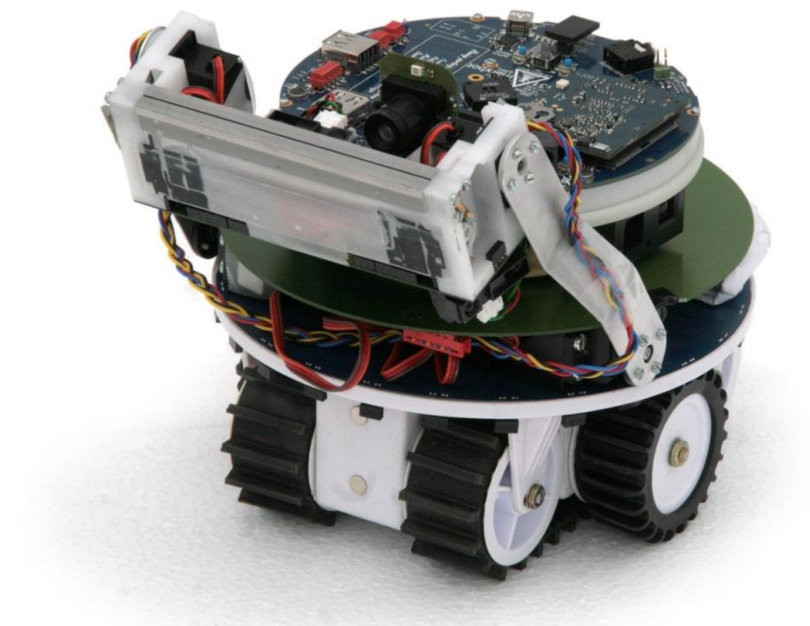
Fraunhofer FOKUS

- Francesco Mondada, EPFL,
- Carlo Pinciroli, ULB

Actual Robots



Foraging robots



Robots with a grip

SCEL Example

Each robot is rendered in SCEL as a component $\mathcal{I}[\mathcal{K}, \Pi, (AM[\,ME\,])]$ where the managed element ME is as follows:

- $ME \triangleq \mathbf{qry}(\text{"}controlStep\text{"}, ?X)@\mathsf{self}.\,(\mathbf{get}(\text{"}termination\text{"})@\mathsf{self}.ME)[\,X\,]$

This process retrieves from the knowledge repository the process implementing the current control step and bounds it to a variable X, executes the retrieved process and waits until it terminates.

- The autonomic manager AM is defined as follows:

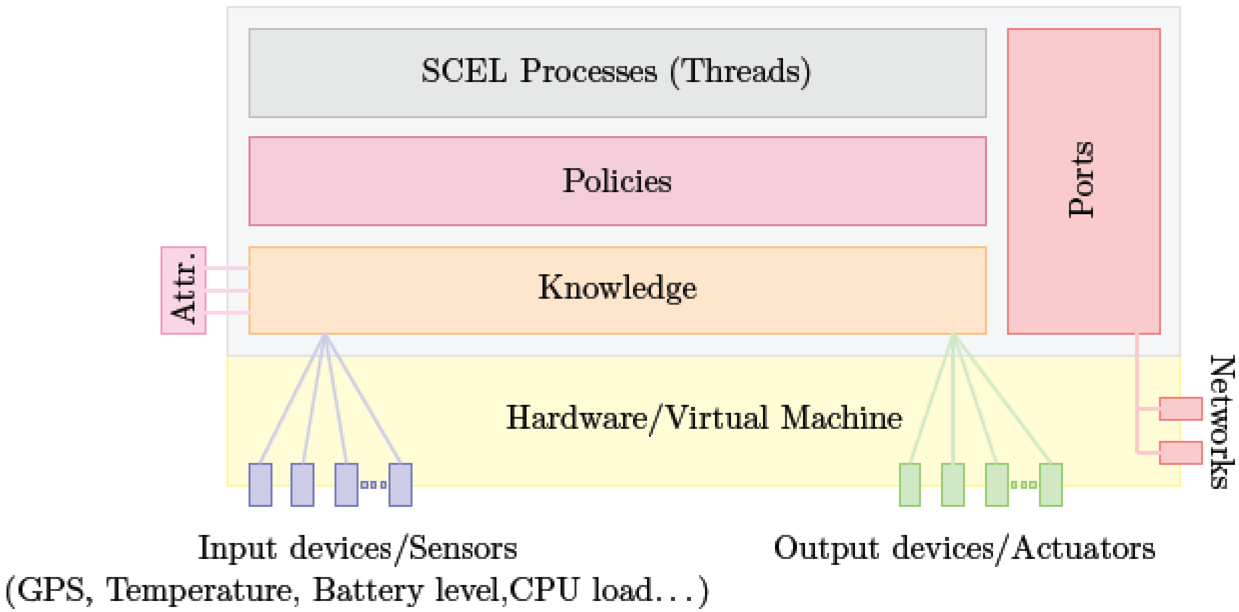
$$AM \triangleq P_{batteryMonitor}[\,P_{dataSeeker}[\,P_{targetSeeker}\,]\,]$$

Process $P_{dataSeeker}$ is defined as follows:

$$\begin{aligned}
P_{dataSeeker} \triangleq\ & \mathbf{qry}(\text{"}targetLocation\text{"}, ?x, ?y)@(\mathcal{I}.task = \text{"}task_i\text{"}).\\
& \mathbf{put}(\text{"}targetLocation\text{"}, x, y)@\mathsf{self}.\\
& \mathbf{get}(\text{"}informed\text{"}, \mathsf{false})@\mathsf{self}.\,\mathbf{put}(\text{"}informed\text{"}, \mathsf{true})@\mathsf{self}\\
P_{randomWalk} \triangleq\ & \mathbf{put}(\text{"}direction\text{"}, random() \cdot 2\pi)@\mathsf{self}.\,\mathbf{put}(\text{"}termination\text{"})@\mathsf{self}\\
P_{informed} \triangleq\ & \mathbf{qry}(\text{"}targetLocation\text{"}, ?x, ?y)@\mathsf{self}.\\
& \mathbf{put}(\text{"}direction\text{"}, towards(x, y))@\mathsf{self}.\,\mathbf{put}(\text{"}termination\text{"})@\mathsf{self}
\end{aligned}$$

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

jRESP Framework for SCEL



Fraunhofer FOKUS

SCEL: Complete Robot Scenario

For the sake of readability, in the definition of process $P_{targetSeeker}$, we have exploited an **if–then–else** construct, which however can be easily rendered in SCEL. For example, the term

$$\mathbf{qry}(\text{“}lowBattery\text{”}, ?low)@\mathsf{self}.\ \mathbf{if}\ (low)\ \mathbf{then}\ \{\ P_{then}\ \}\ \mathbf{else}\ \{\ P_{else}\ \}$$

can be rewritten as follows:

$$\mathbf{qry}(\text{“}lowBattery\text{”}, \mathsf{true})@\mathsf{self}.\ P_{then} \ + \ \mathbf{qry}(\text{“}lowBattery\text{”}, \mathsf{false})@\mathsf{self}.\ P_{else}$$

The processes executed by the managed element ME at each control step are as follows:

$$\begin{aligned}
P_{lowBattery} &\triangleq \mathbf{put}(\text{“}stop\text{”})@\mathsf{self}.\ \mathbf{qry}(\text{“}gps\text{”}, ?x, ?y)@\mathsf{self}.\\
&\quad\ \mathbf{put}(\text{“}sos\text{”}, x, y)@(\mathcal{I}.task = \text{“}task_i\text{”}).\\
&\quad\ \mathbf{get}(\text{“}rescued\text{”})@\mathsf{self}.\ \mathbf{put}(\text{“}termination\text{”})@\mathsf{self}\\[1ex]
P_{found} &\triangleq \mathbf{put}(\text{“}stop\text{”})@\mathsf{self}.\ \mathbf{qry}(\text{“}gps\text{”}, ?x, ?y)@\mathsf{self}.\\
&\quad\ \mathbf{put}(\text{“}targetLocation\text{”}, x, y)@\mathsf{self}.\ \ldots\ execute\ task\ i\ \ldots\\[1ex]
P_{informed} &\triangleq \mathbf{qry}(\text{“}targetLocation\text{”}, ?x, ?y)@\mathsf{self}.\\
&\quad\ \mathbf{put}(\text{“}direction\text{”}, towards(x, y))@\mathsf{self}.\ \mathbf{put}(\text{“}termination\text{”})@\mathsf{self}\\[1ex]
P_{randomWalk} &\triangleq \mathbf{put}(\text{“}direction\text{”}, random() \cdot 2\pi)@\mathsf{self}.\ \mathbf{put}(\text{“}termination\text{”})@\mathsf{self}
\end{aligned}$$

Fraunhofer FOKUS

R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report,.

jRESP: Implemengationm of Robot Scenario

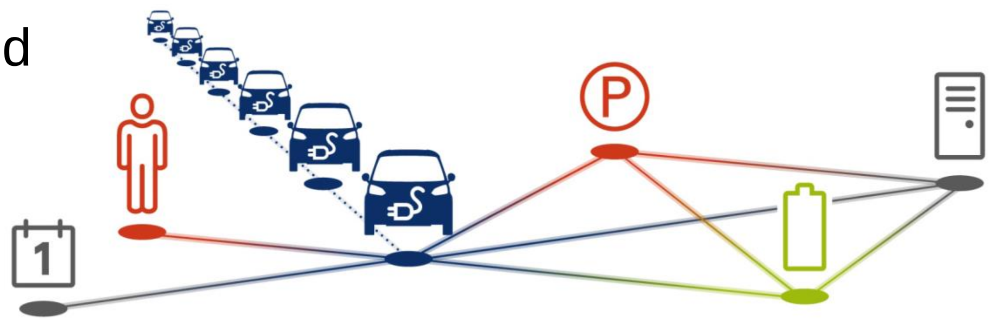
```
Tuple t = query( new Template(
                    new ActualTemplateField("lowBattery") ,
                    new FormalTemplateField(Boolean.class)) ,
                 Self.SELF);
boolean low = t.getElementAt(Boolean.class,1);
if (low) {
  get( new Template(
                 new ActualTemplateField( "controlStep" ) ,
                 new FormalTemplateField( Agent.class ) ) ,
      Self.SELF );
  put( new Tuple( "controlStep" , new LowBattery() ) , Self.SELF );
  query( new Template(
                 new ActualTemplateField("lowBattery") ,
                 new ActualTemplateField(false)) ,
       Self.SELF );
} else {
  t = query( new Template(
                new ActualTemplateField("target") ,
                new FormalTemplateField(Boolean.class)) ,
           Self.SELF );
  boolean found = t.getElementAt(Boolean.class, 1);
  if (found) {
    get( new Template(
                   new ActualTemplateField( "controlStep" ) ,
                   new FormalTemplateField( Agent.class ) ) ,
        Self.SELF );
    put( new Tuple( "controlStep" , new Found() ) , Self.SELF );
    doTask();
  } else {
    t = query( new Template(
                   new ActualTemplateField("informed")
```

SCEL: E-Mobility Example

- **Components and their interactions**
- Travel desires of drivers
- Individual EVs and EV fleets
- Traffic and road network
- Charging and energy network

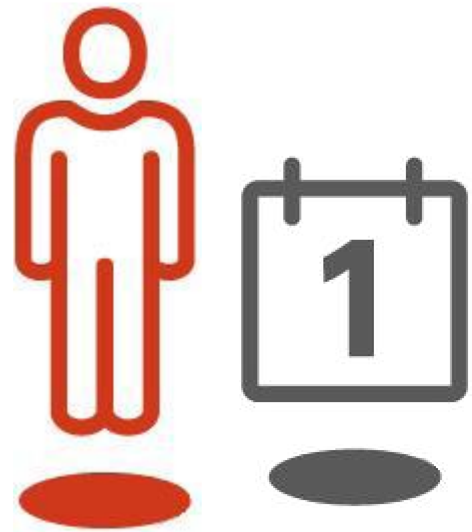
Challenges

- Intelligent knowledge distribution
- Predicting e-vehicle travel time and energy
- Travel planning



User Perspective

- **Goals**
  - Guarantee to reach each user destination in time
  - Maximize the fulfillment rate of the user preferences
- **Awareness issues**
  - User schedule
  - User preferences
- **Task**
  - Planning the user's journey
- **Actions**
  - Acquire and distribute information
  - Manage temporal conflicts of the user schedule
  - Manage travel modalities



Fraunhofer FOKUS

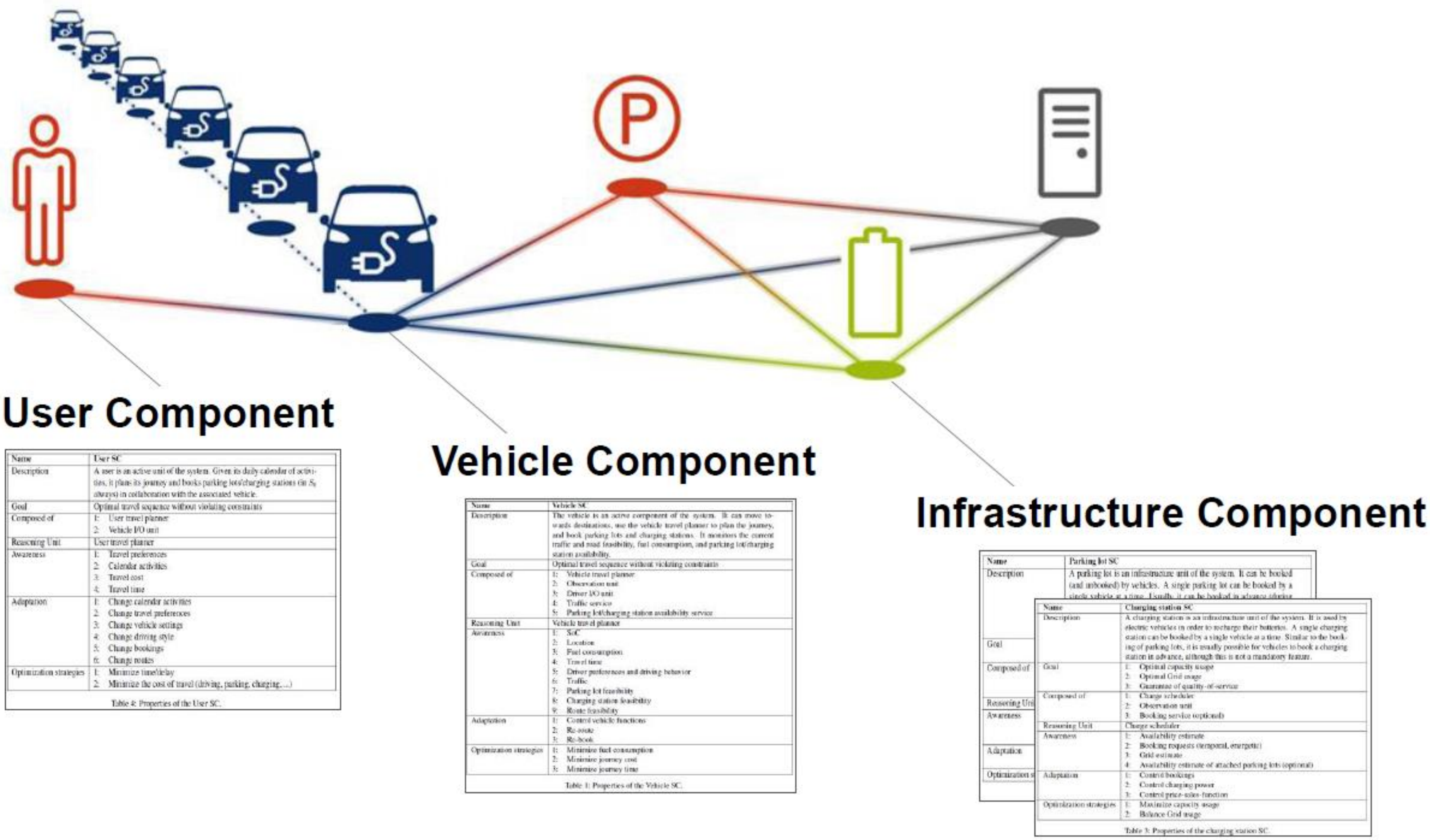
Vehicle Perspective

- **Goal**
  - Optimal travel sequence without violating constraints
- **Awareness issues**
  - Current and predicted internal vehicle states
  - Current and future trips
- **Task**
  - Planning of the vehicle journeys
- **Actions**
  - Acquire and distribute information
  - Planning individual vehicle trips
  - Planning resource usage (parking, charging slots)

Infrastructure Perspective

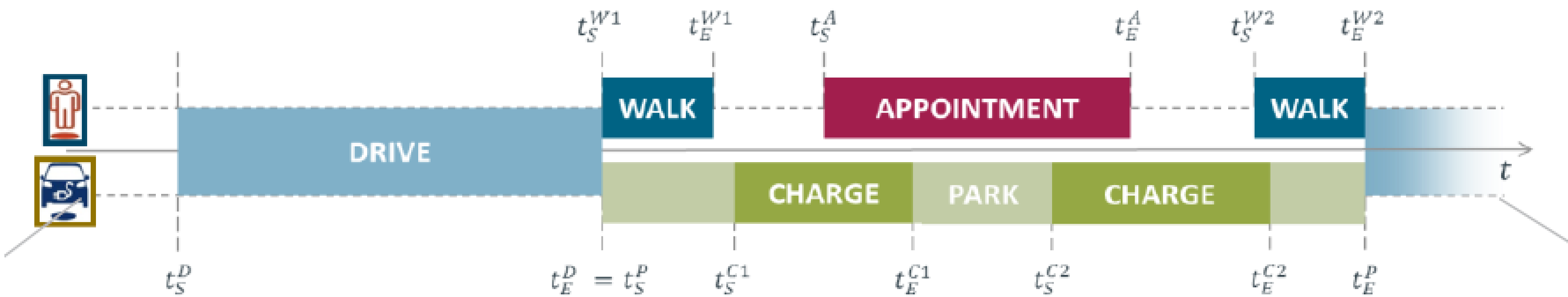
- **Goal**
  - Optimal capacity usage of the infrastructure resource
  - Guarantee quality-of-service
- **Awareness issues**
  - Bookings
  - Availability estimate
  - Price-sales-function for infrastructure demand
- **Task**
  - Supply and demand management
- **Actions**
  - Acquire and distribute information
  - Manage bookings
  - Manage pricing

E-Mobility Service Components



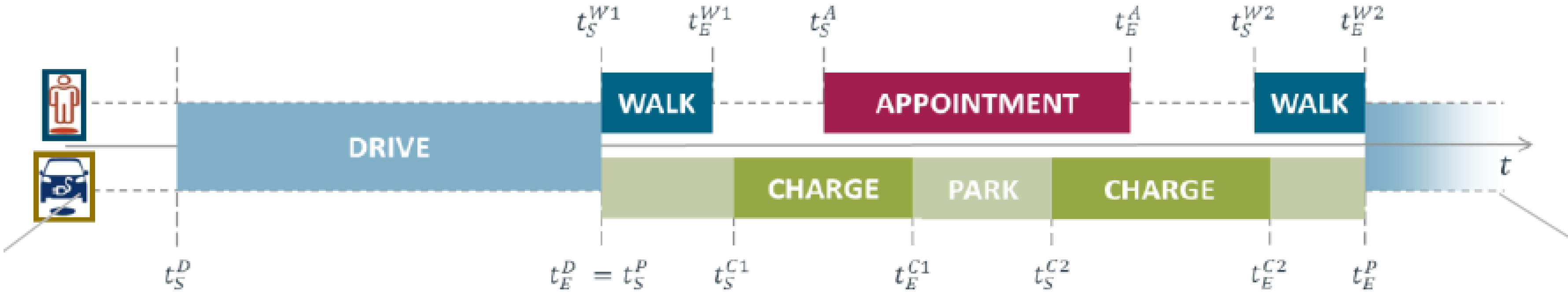
Soft Constraint Logic Programming

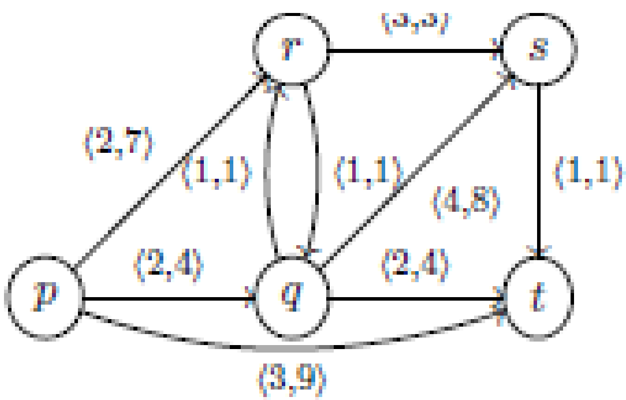
- Formalization of the eMobility planning problem
- Multi-criteria shortest path problem on the trip-level
- SCLP model on the journey level to find non-dominated optimal journeys



Fraunhofer FOKUS

Modeling the Jorney





Road Network

Loc.	Start. time	Dur.
p	7	1
r	11	2
t	18	3

Appointments

Name	Spots	Loc.
csp1	7	p
csr1	4	r
csr2	0	r

Charging Stations

Programming Model

CIAO Prolog:

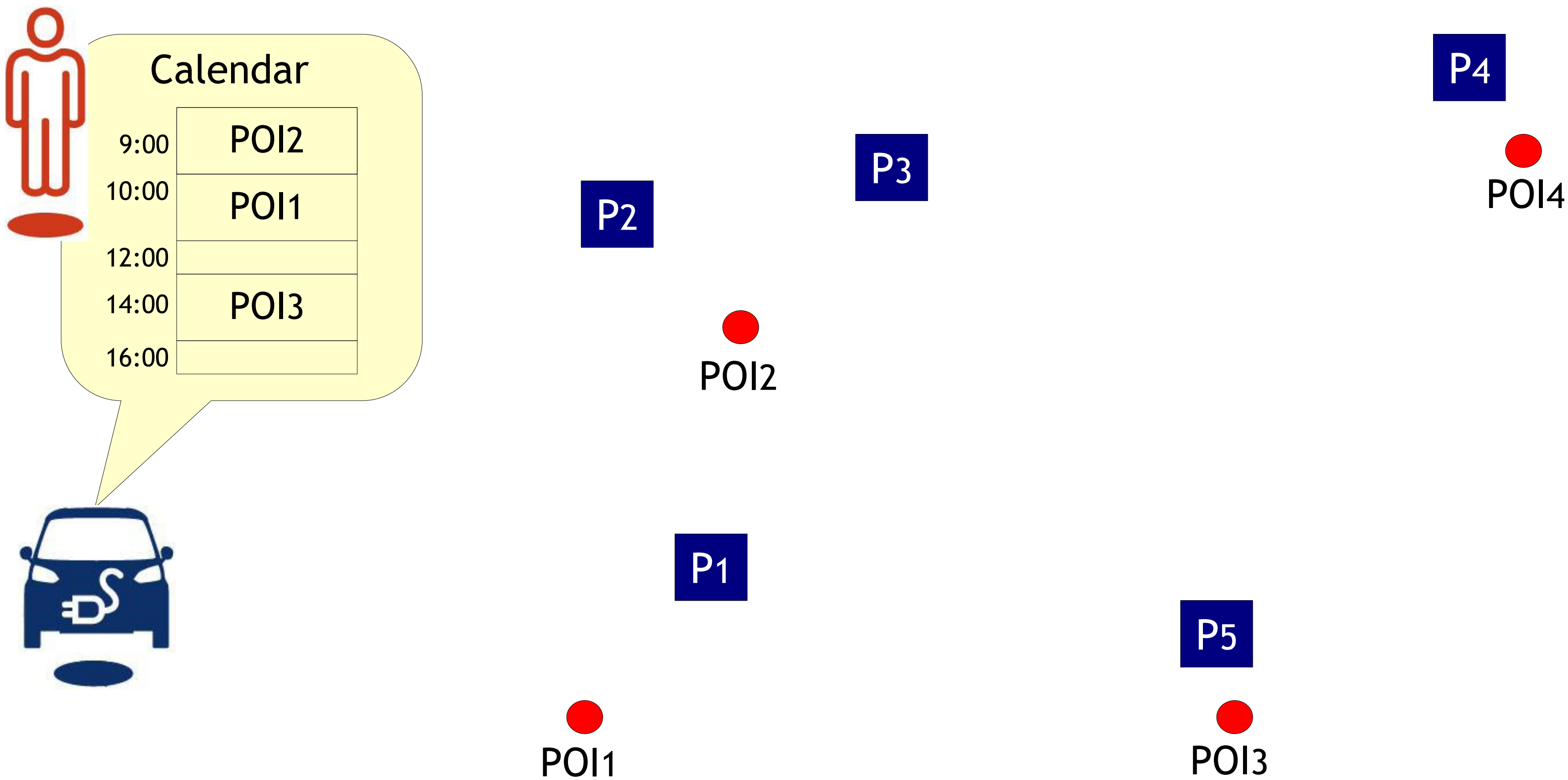
```
:-module(journey,_,_).
:-use_module(library(lists)).
:-use_module(library(aggregates)).
:-use_module(paths).

plus([],L,[]).
plus([[P,T,E,ChEv]|RestL],L,
      [[P,T,E,ChEv]|BestPaths]):-
  nondominated([P,T,E],L),
  plus(RestL,L,BestPaths).
plus([[P,T,E,ChEv]|RestL],L,
                    BestPaths):-
  \+nondominated([P,T,E],L),
  plus(RestL,L,BestPaths).


nondominated([P,T,E],[]).
nondominated([P,T,E],
          [[P1,T1,E1,ChEv1]|L]):-
  \+minPair([T1,E1],[T,E]),
  nondominated([P,T,E], L).


appointment(p,7,1).
appointment(r,11,2).
appointment(t,18,3).

chargingStation(csp1,7,p).
chargingStation(csr1,4,r).
chargingStation(csr2,0,r).
```

```
times([T1,E1],[T2,E2],[T3,E3]):-
  T3 = T1 + T2,
  E3 = E1 + E2.

journey([X,Y],[P],[],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  path(X,Y,P, [X],[T,E],SoC),
  timeSum(Tx,Dx,T,ArrT), ArrT=<Ty.

journey([X,Y],[P],[[X,ID]],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  \+path(X,Y,P, [X],[T,E],SoC),
  chargingStation(ID,Spots,X),Spots>0,
  newSoC(SoC,Dx,NewSoC),
  path(X,Y,P, [X],[T,E],NewSoC),
  timeSum(Tx,Dx,T,ArrT), ArrT=<Ty.

journey([X|[Y|Z]],[P|LP],ChEv, [T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  path(X,Y,P, [X],[T1,E1],SoC),
  timeSum(Tx,Dx,T1,ArrT), ArrT=<Ty,
  journey([Y|Z],LP,ChEv,[T2,E2],(SoC-E1)),
  times([T1,E1],[T2,E2],[T,E]).

journey([X|[Y|Z]],[P|LP],[[X,ID]|ChEv],[T,E],SoC):-
  appointment(X,Tx,Dx), appointment(Y,Ty,Dy),
  \+path(X,Y,P, [X],[T1,E1],SoC),
  chargingStation(ID,Spots,X), Spots>0,
  newSoC(SoC,Dx,NewSoC),
  path(X,Y,P, [X],[T1,E1],NewSoC),
  timeSum(Tx,Dx,T1,ArrT), ArrT=<Ty,
  journey([Y|Z],LP,ChEv,[T2,E2],(NewSoC-E1)),
  times([T1,E1],[T2,E2],[T,E]).
```

```
journeys(Places,EV,BestJourneies):-
findall([P,T,E,ChEv],journey(Places,P,ChEv,[T,E],SoC),ResL),
plus(ResL,ResL,BestJourneies).
```

Fraunhofer FOKUS

SCEL Modelling: Main Scenario



POIj = j-th point of interest

Pi = i-th parking lot

Involved entities

VEHICLE:

- Asks information to parking lots close to the POIs
- Provides this information to the planner, which generates the plan (i.e. the list of parking lots to be reserved)
- Books the planned parking lots
- Monitors the execution of the plan

PARKING LOT:

- Manages (accepts) the requests of booking

Scenario in SCEL: Components

Vehicles and parking lots are SCEL components

running the following processes:

= ContactParkingLots[Planner[Book[MonitorPlanExecution]]]

P1 = ProvideParkingData[ManageBookings]

Scenario in SCEL: vehicle component

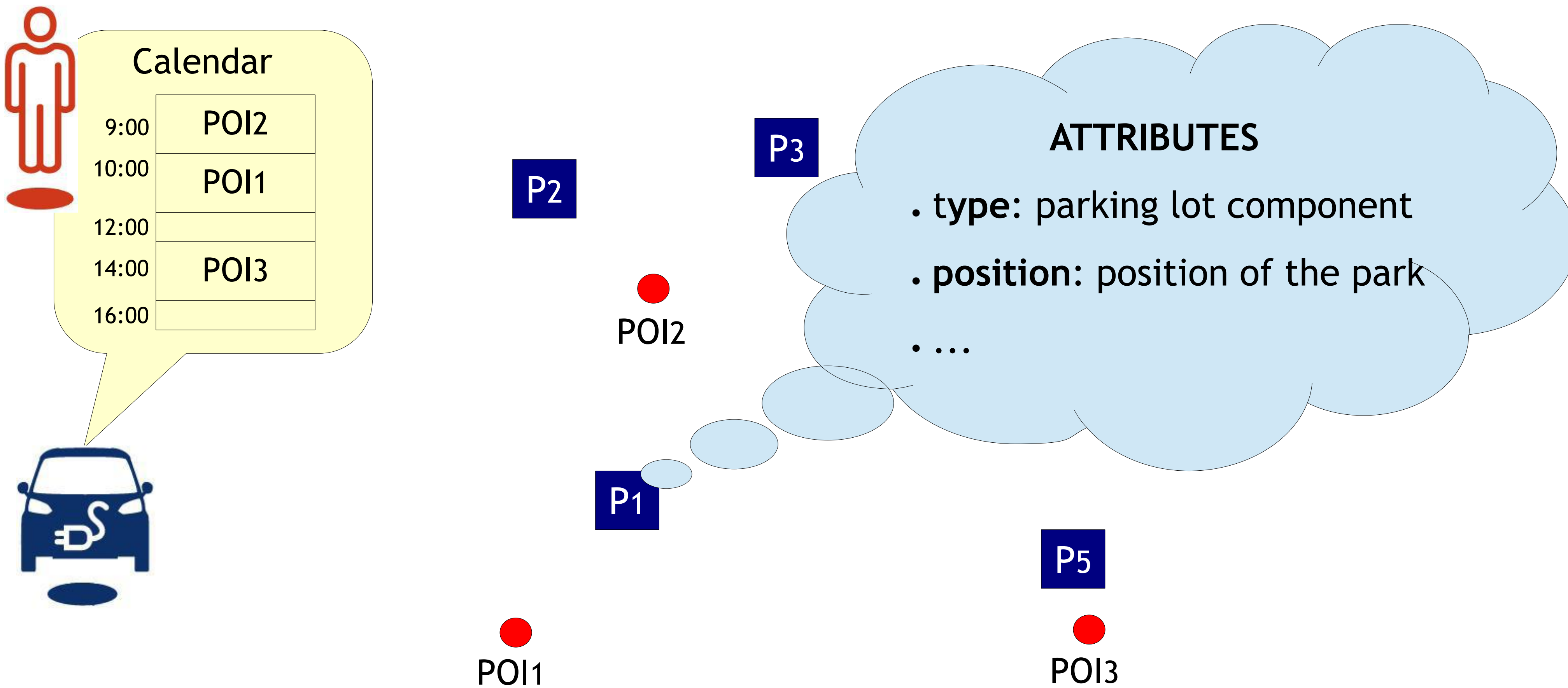
VEHICLE:

- Asks information to parking lots close to the POIs
- Provides this information to the planner, which generates the plan (i.e. the list of parking lots to be reserved)
- Books the planned parking lots
- Monitors the execution of the plan

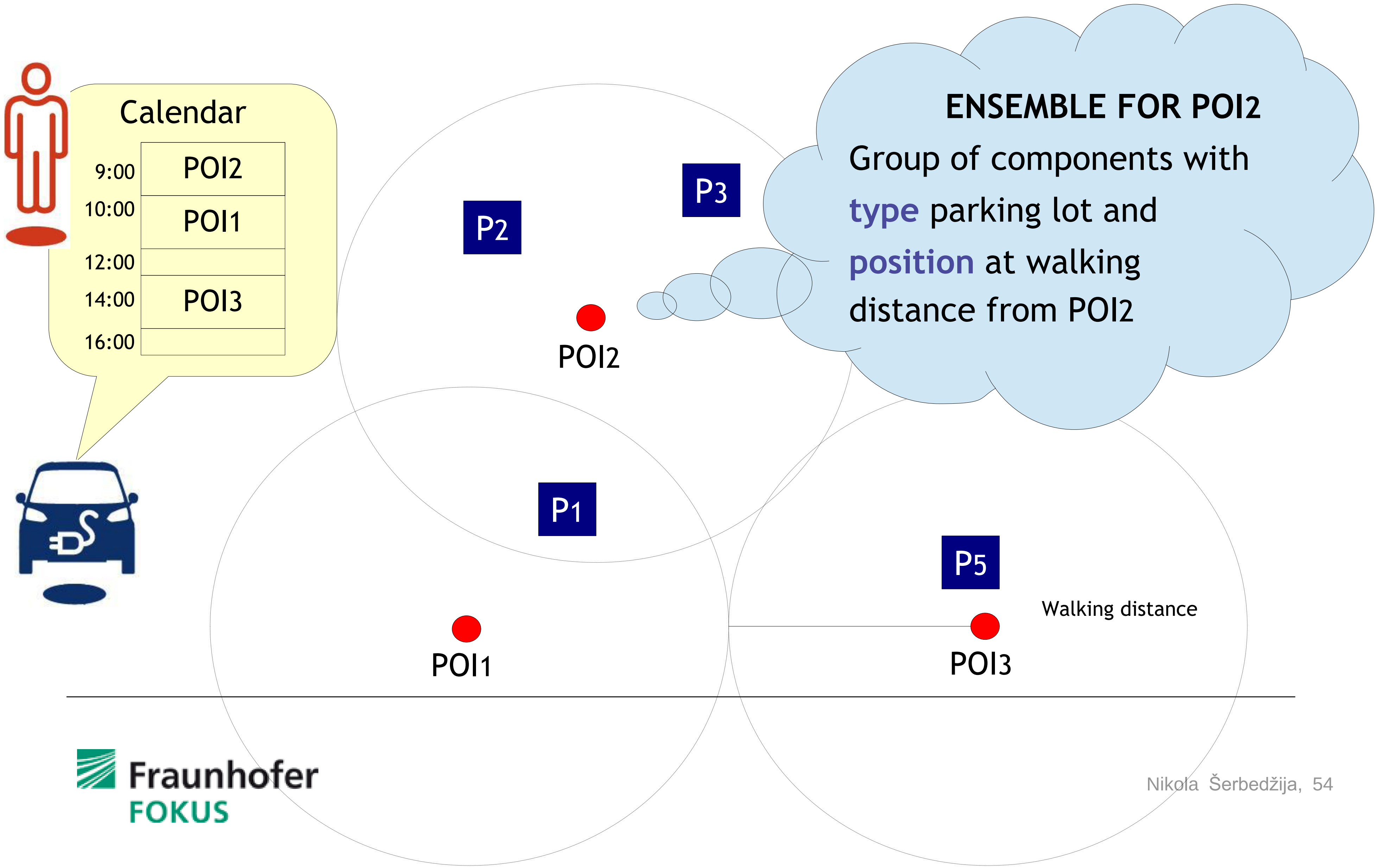
PARKINGLOT:

- Manages (accepts) the requests of booking

Parking Lots close to POIs as Ensembles

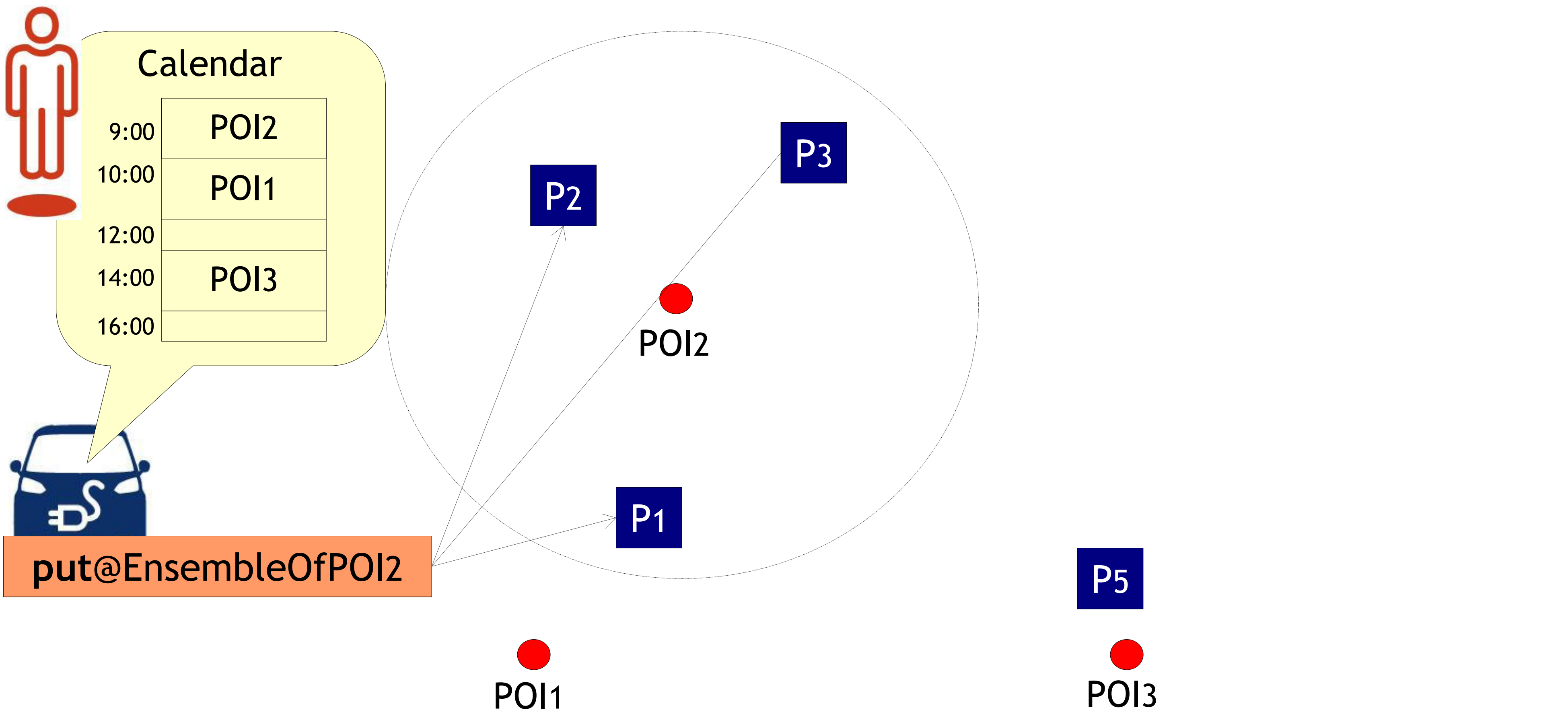


Parking Lots close to POIs as Ensembles



Fraunhofer FOKUS

Parking Lots close to POIs as Ensembles



Vehicle component

VEHICLE = ContactParkingLots[Planner[Book[MonitorPlanExecution]]]

```
ContactParkingLots =
//read the size of the calendar (i.e. the list of appointments)
qry("calendarSize", ?n)@self .
//scan the calendar
for(i := 0 ; i < n ; i ++){
        //read an appointment of the calendar
        qry("calendar", i, ?poi, ?poiPos, ?when, ?howLong)@self .
        //contact the parking lots near to the POI (resorting to attribute-based communication)
        put("searchPLot", self, poi)@{l.type="PLot" ∧ walkingDistance(poiPos,l.pos)}
}
//signal the completion of the phase of requirement of data to the parking lots
put("dataRequestSent")@self
```

Ensemble predicate

Fraunhofer FOKUS

Vehicle component (cont.)

VEHICLE = ContactParkingLots[Planner[Book[MonitorPlanExecution]]]

Planner =
//wait the completion of the phase of requirement of data to the parking lots
get(“dataRequestSent”)@self .
// we intentionally leave unspecified this process
//input: collection of tuples of the form ⟨poi, pLotId, pLotInfo⟩ received from the pLots
//output: list of chosen planned pLots,
i.e. ⟨□planListSize”, n⟩ ⟨□plan”, 0, pLotId0, when0, howLong0⟩ ... ⟨□plan”, n - 1, ...⟩
//signal the conclusion of the planning phase
put(“planningCompleted”)@self

Vehicle component (cont.)

VEHICLE = ContactParkingLots[Planner[Book[MonitorPlanExecution]]]

```
Book =
//wait the completion of the planning phase
get("planningCompleted")@self .
//read the size of plan list (i.e. the pLots to be booked)
get("planListSize", ?n)@self .
//scan the plan
for(i := 0 ; i < n ; i ++){
        //read an entry of the plan list
        get("plan", i, ?pLot, ?when, ?howLong)@self .
        //send the booking request to the pLot
        put("book", self, when, howLong)@pLot .
        //wait for the reply of pLot (we assume that booking requests always succeed)
        get("bookingOutcome", true)@self .
        //store the reservation in the list of reservations
        put("reservation", i, pLot, when, howLong)@self
}
//close the list of reservations
put("reservationListSize", n)@self .
//signal the conclusion of the booking phase
put("bookingCompleted")@self
```

Fraunhofer FOKUS

Vehicle component (cont.)

VEHICLE = ContactParkingLots[Planner[Book[$\boxed{\text{MonitorPlanExecution]]]}}$

```
MonitorPlanExecution =
//wait the completion of the booking phase
get("bookingCompleted")@self .
//read the size of reservation lists (i.e. the pLots to be visited)
get("reservationListSize", ?n)@self .
//scan the reservation list
for(i := 0 ; i < n ; i ++){
        //read a reservation
        get("reservation", i, ?pLot, ?when, ?howLong)@self .
        //display to the user the information about the next reservation
        put("reservation", ?pLot, ?when, ?howLong)@screen .
        //wait for the arrival to the parking lot (signalled by the user)
        get("arrivedAt", pLot)@self .
}
//signal the conclusion of the plan execution phase
put("planExecuted")@self
```

Fraunhofer FOKUS

Parking Lot Component

PARKINGLOT = ProvideParkingData[ManageBookings]

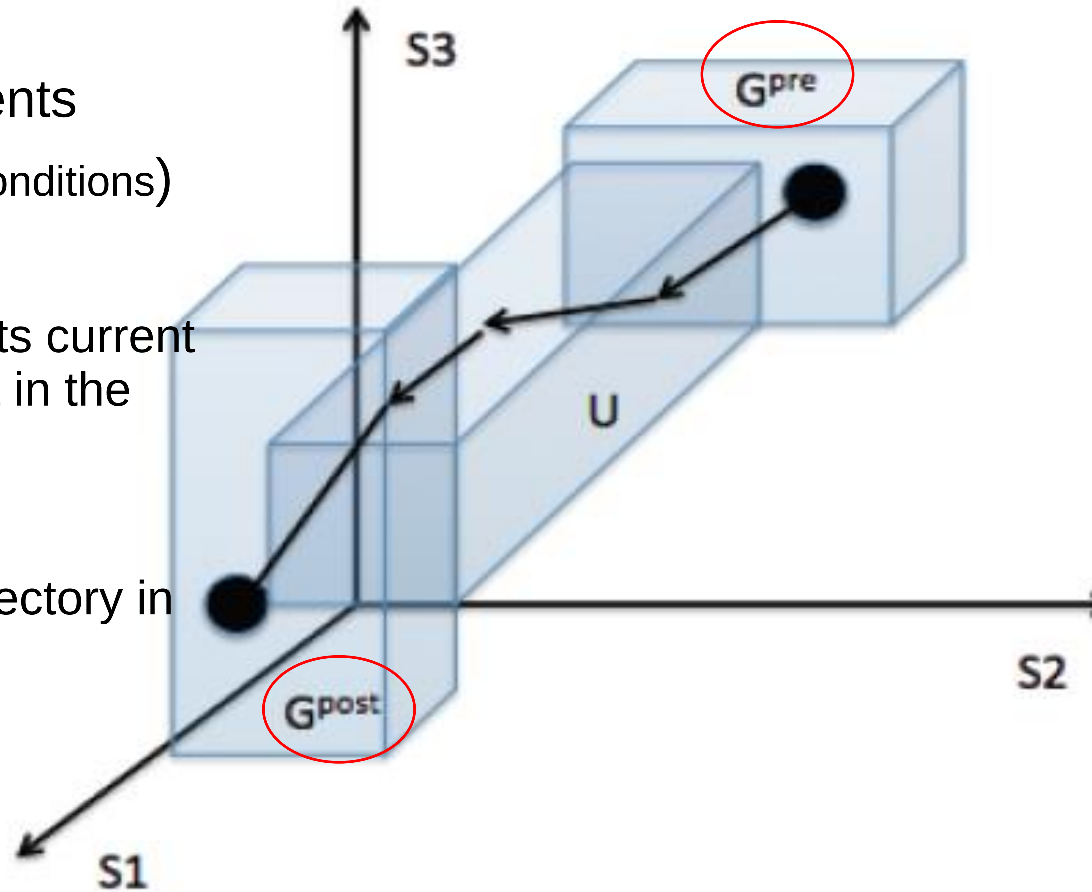
```
ProvideParkingData =
//get a request of data about the parking lot
get("searchPLot", ?requester, ?poi)@self .
//provide the requested data (we intentionally leave unspecified the provided informations)
plotInfo := ...
put(poi, self, plotInfo)@requester .
//handle next request
ProvideParkingData
```

```
ManageBookings =
//get a booking request
get("book", ?requester, ?when, ?howLong)@self .
//accept and store the booking
put("booking", when, howLong, requester)@self .
put("bookingOutcome", true)@requester .
//handle next request
ManageBooking
```

Fraunhofer FOKUS

SOTA (State of The Affairs) Adaptation Model

- A general n-dimensional model for modeling the adaptation requirements
- SOTA goals (states) and utilities (conditions)
- Self-awareness:
  - Ability to autonomously recognize its current position and direction of movement in the SOTA space
- Self-adaptation:
  - Ability to dynamically direct the trajectory in the SOTA space
- Need for feedback loops
  - SOTA self-adaptation patterns



The trajectory of an entity in the SOTA space

A Simulation Tool for Adaptation Pattern

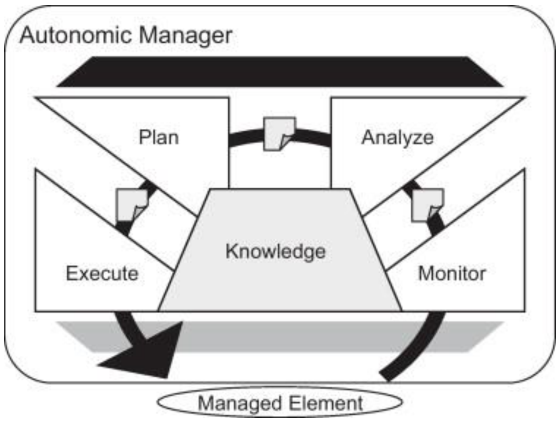
- Eclipse-based simulation plug-in for the engineering (i.e. explicit modeling, simulating and animating, and validating) of SOTA patterns based on feedback loops
  - Validation of the approach:
    - E-mobility case study's individual driver planning scenario (basic scenario)
  - Environment used:
    - IBM Rational Software Architect Simulation Toolkit 8.0.4

Key Goals of the Plug-in

- **Modeling** of the SOTA patterns using UML 2–patterns' structural & behavioral information modelled using activity, sequence and composite structure diagrams
- Visual **animation** of the SOTA patterns' behavior during execution to expose the runtime view (next element to execute, executed element, active states, tokens)
- Animating **composite structure** of the SOTA patterns, e.g. interaction messages and token flows, and execution history information
- Model-level **debugging** and detailed control of **execution** of the patterns, e.g. breakpoints, stepping, suspend, resume, terminate
- **Run-time prompting** during patterns simulation

Notion of Feedback Loops Explored in SOTA

- Extends the IBM's **MAPE-K** adaptation model (monitor, analyze, plan and execute over a knowledgebase) with multiple and interacting feedback control loops
- Feedback structure with multiple control loops :
  - Intra-loops: adaptation coordination between sub-loops within a single feedback loop
  - Inter-loops: adaptation coordination between multiple feedback loops
- Loops interact using three mechanisms :
  - Stigmergy: loops act on a shared subsystem
  - Hierarchy: an outer loop controls an inner loop
  - Direct interaction: managers communicate with each other
- Feedback loop types: positive and negative



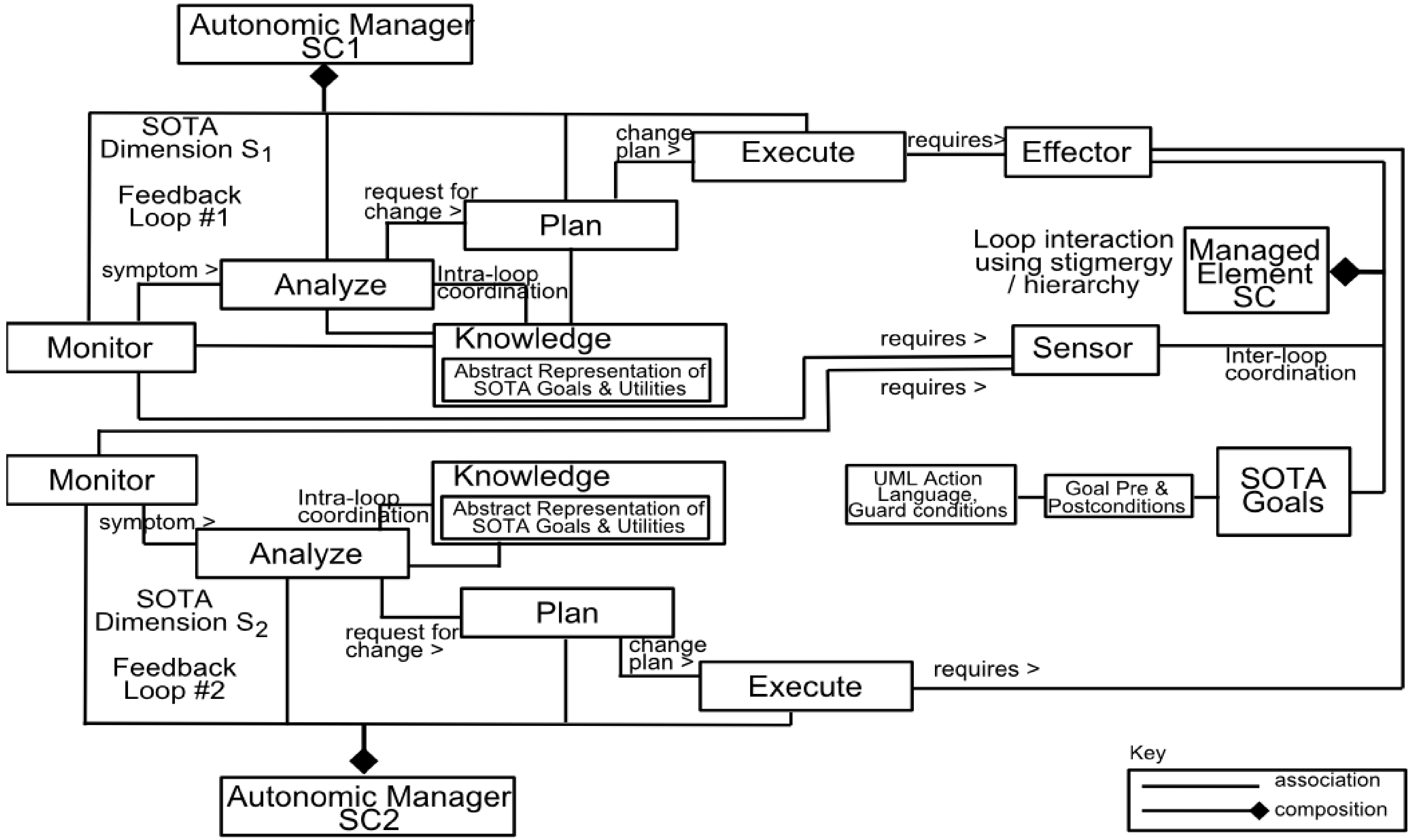
A Key SOTA Pattern

- **Decentralised SC pattern**
  - External, explicit feedback loop
  - Managed Element SC :
    - Sensors, effectors and SOTA goals
  - Autonomic Manager (AM) :
    - Handles adaptation activities of the managed element on a particular SOTA awareness dimension
    - AM has IBM's MAPE-K model (with *intra-loops* within a loop)
    - More AMs?

      Increases the autonomicity of the managed element SC

      Each AM closes a feedback loop (loops interact using *stigmergy, hierarchy* and *direct interaction)*

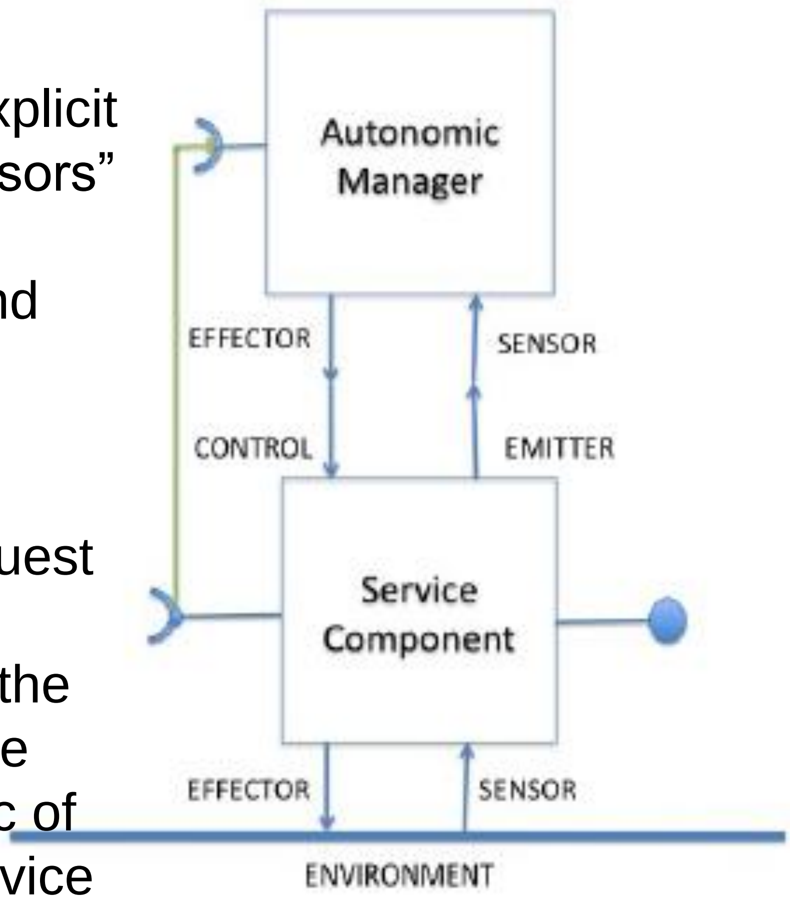
Autonomic SC Pattern with 2 Managers



Dhaminda Abeywickrama

Autonomic Service Component Pattern

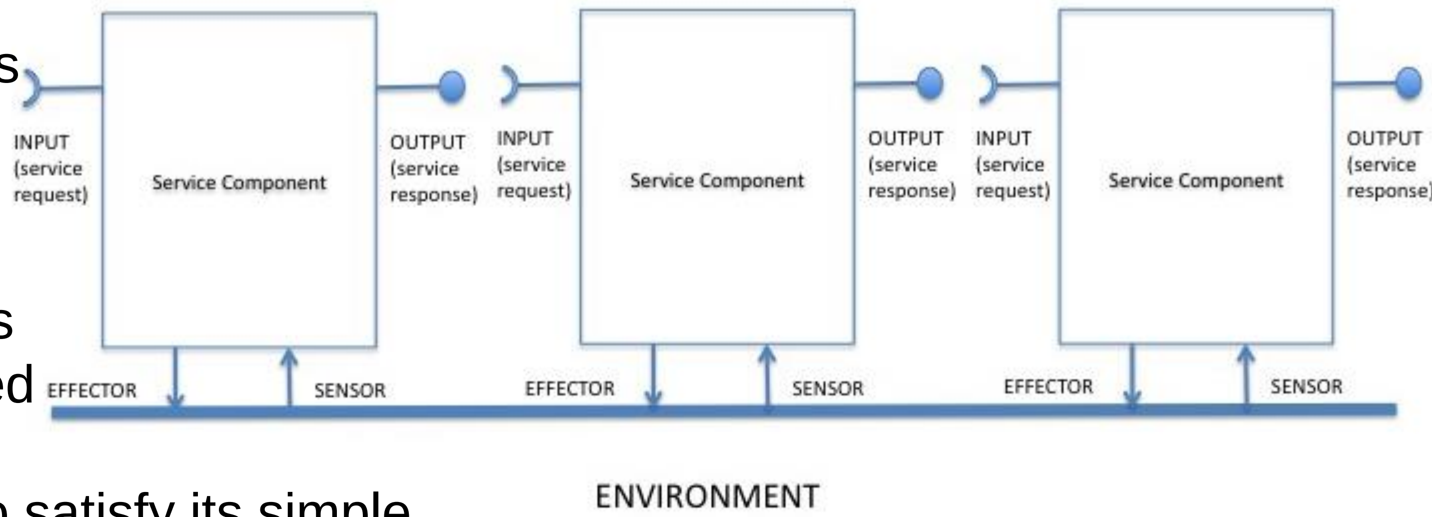
Behaviour:
This pattern is designed around an explicit autonomic feedback loop. Using “sensors” the SC and the AM can perceive the different events in the environment and the changes in the environment itself.



The AM perceives not only the environment, but also the service request made at the component and its logic. Having its internal goals and utilities, the AM manages the adaptation inside the component, maybe changing the logic of choosing actions in response to a service request.

Reactive Stigmergy Service Components Ensemble

Behaviour:

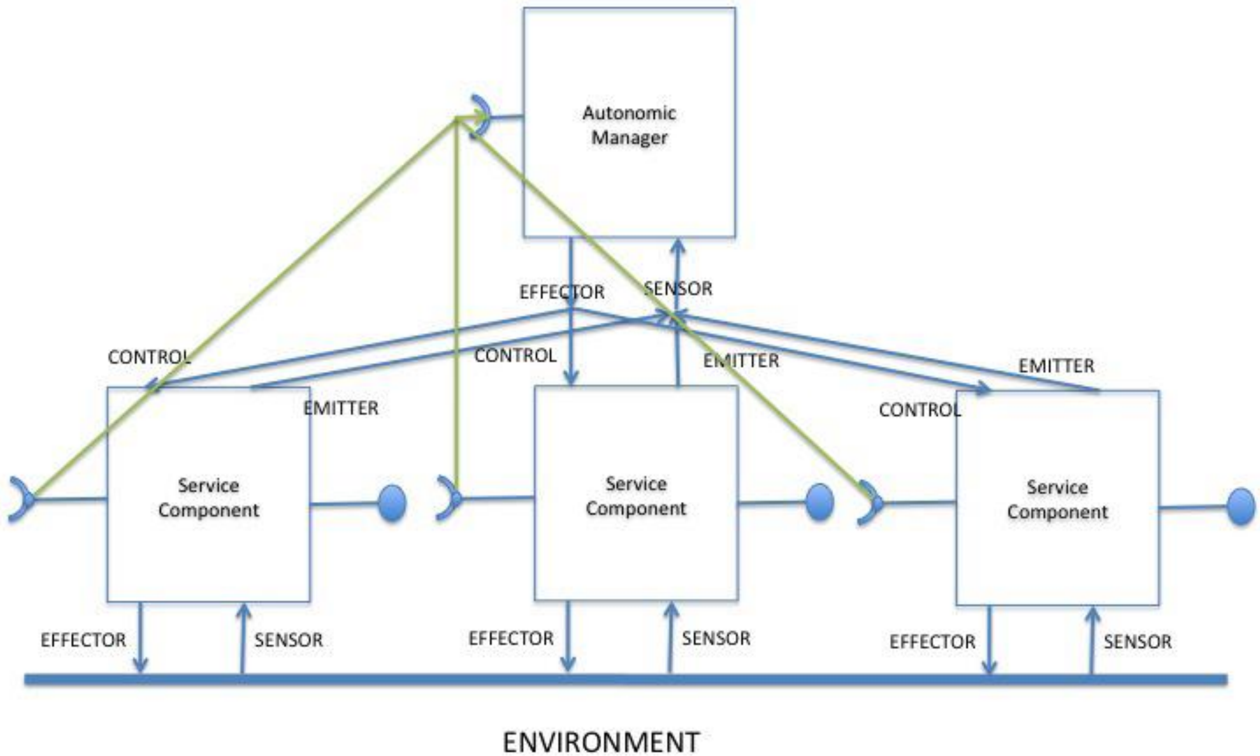


This pattern has not a direct feedback loop. Each single component acts like a bioinspired component (e.g. an ant). To satisfy its simple goal, the SC acts in the environment that senses with its “sensors” and reacts to the changes in it with its “effectors”. The different components are not able to communicate one with the other, but are able to propagate information (their actions) in the environment. Than they are able to sense the environment changes (other components reactions) and adapt their behaviour due to these changes.

Centralised AM Service Components Ensemble

Behaviour:

This pattern is designed around an unique feedback loop. All the components are managed by a unique AM that "control" all the components behaviour and, sharing knowledge about all the components, is able to propagate adaptation.



SOTA Example: E- mobility

- Shift from vehicle to mobility purchasing
- Meet consumer expectations in resource-constraint mobility
- Manage infrastructure availability in resource-constraint mobility

Innovation:

•The entities of the mobility system are heterogeneous, interactions are complex and knowledge is distributed

- Flexible adaptation in a dynamic environment

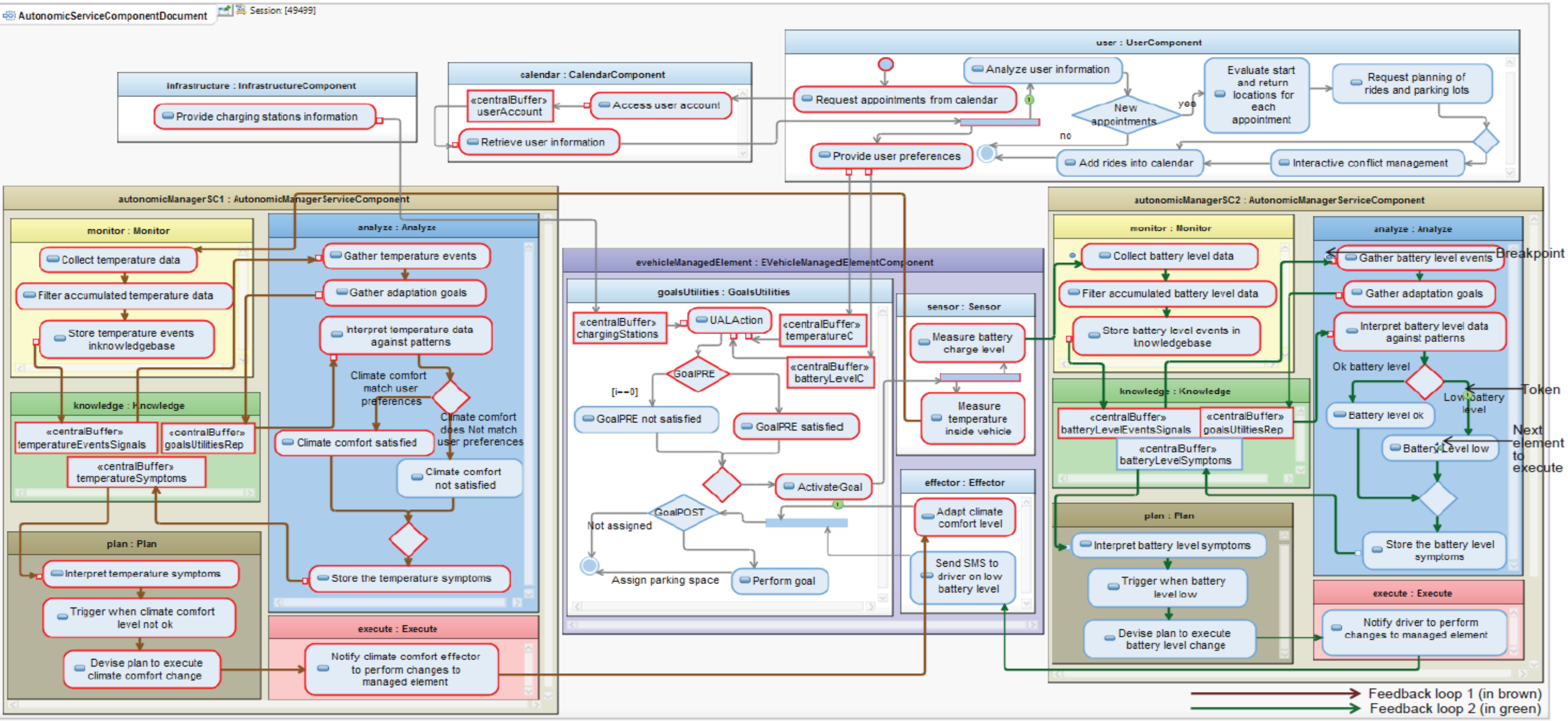
Goal:

- Self-organizing vehicles interacting with an intelligent infrastructure

SOTA Simulation of the E-Mobility System

- Each SC and SCE of the case study scenario is described using:
  - SOTA goals and utilities
  - Awareness being monitored by the managers for a managed element
  - Any contingencies that can occur
  - Corresponding self-adaptive actions using SOTA feedback loops
- Adaptation handling :
  - Separate Autonomic Managers (AMs) for each SOTA awareness dimension
    - E.g. electric vehicle has AMs to handle adaptation of battery state of charge, climate comfort requirements
  - High-level AMs to handle adaptation activities involved in multiple components such as the user and the electric vehicle
    - E.g. routing

Simulation: SOTA Decentralized SC pattern simulated for e-mobility



III Conclusion: Development Approach

Softvar develoment is an iterative process that proposes a doubly connected design-runtime lifecycle for the development of service component ensembles (SCE)

Phases and tools for the design :

Requirements Engineering *for building a conceptual and operational framework to be used to elicit and rationally represent ensembles requirements:*

- ***SOTA*** *for adaptation requirements*

Modeling/Programming *for the specification and coding of SCEs:*

- *Agamemnon, BIP, KnowLang, Maude, POEM, SCEL, jRESP, Java*

Verification/Validation *supporting formal proofs of SCEs' models and code:*

X
- *BIP D-Finder, GMC, Iliad, jSAM, MESS, LTSA*

  (model checking, deadlock finder, modelchecker for C, Integrated dDevelopment Environment …,)

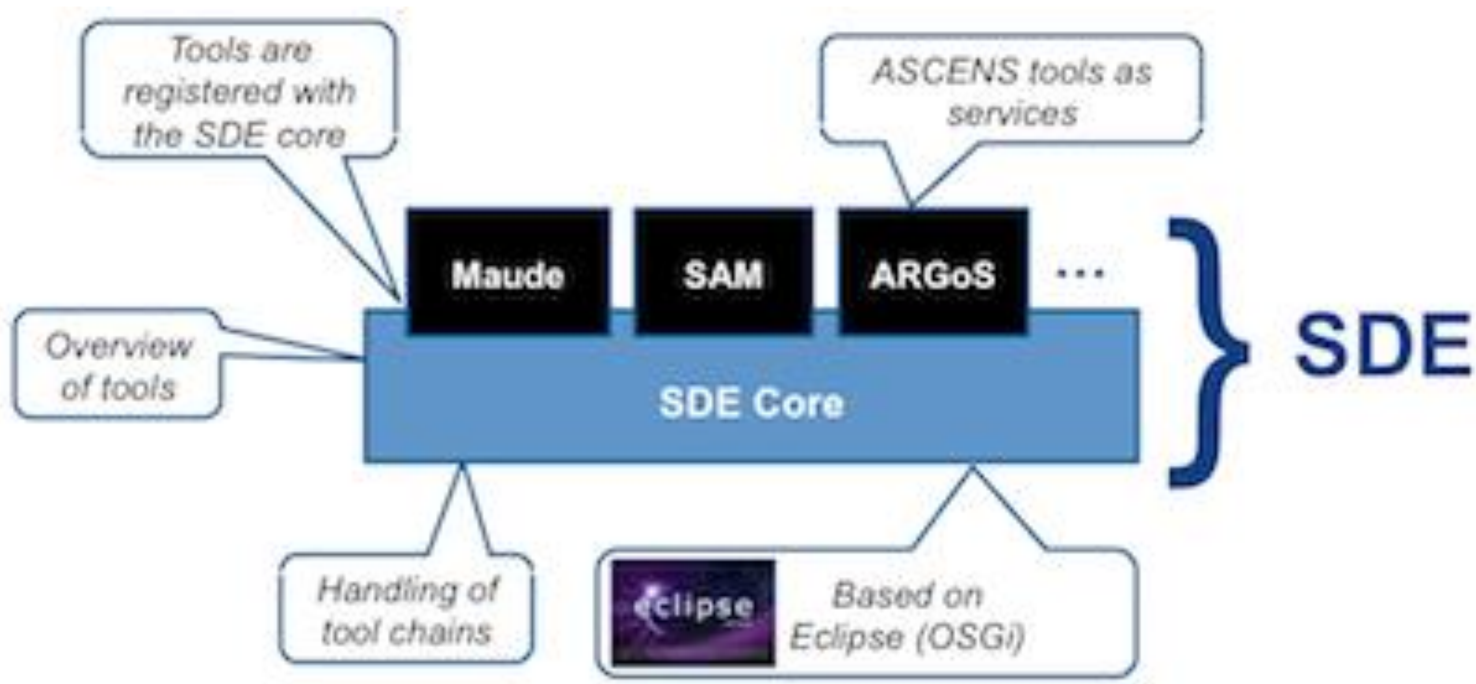
BIP - rigorous checking of the consistency between the different design steps

Maude - a high-performance reflective language

POEM - a toolkit for modeling, visual debugging, developing, and deploying applications

Tool Support

Tool integration platform
the service development environment (SDE) enables loosely coupled tools to work together by building tool chains



Further Work

- Self-aware systems [www.ascens-ist.eu]
- User behavior
  - Task oriented
  - Goal oriented
  - Socially acceptable
- Individual adaptation
- Collective adaptation
- Trust issues
- Ethical Issues
  - Privacy
  - Impact
  - Individual/Social consequences

Acknowledgement

Most of the work presented here has been done under the ASCENS project (project number FP7- 257414) [7], funded by the European Commission within the 7th Framework Programme (see the web address and consortium at the picture at the right). Special thanks go to the developers group of the SCEL language (Rocco De Nicola from IMT Lucca and his group).