Service Components and Ensembles: Building Blocks for Autonomous Systems

- tutorial -

Nikola B. Šerbedžija

24 March 2013
Lisboa
Outline

I. Introduction (definition, abstract, motivation, approach)

II. Requirements analyses
   • Practical examples
   • Requirements

III. Modeling
   • Approach
   • SCEL
   • Adaptation patterns
   • Reasons on system properties

IV. Deployment
   - JRESP
   - Implementation framework

V. Conclusion (discussion and further work)
I. Definition: Autonomous

autonomous [aw-ton-uh-muhs]
- adjective
   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.
   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
Many current Information and Communications Technology (ICT) systems and infrastructure, such as

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 Stojnic
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 the assumptions.
64. To exploit a graphical language in order to achieve automatic generation of engines. Tom Holvoet
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 adaptivity 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. Guillaume Dugue
52. How to measure the level of awareness? E.g. the number of variables AND the algorithm for the number of variables.
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? Manuele Brambilla
47. Monitoring and controlling emergent properties and specifying and controlling adaptation; current methods do not allow for this"
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).
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 behavior.

- **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

## 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
- Knowledge representation of self-aware components
- Mechanisms for adaptation
- 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/
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

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>SC: Service Component</th>
<th>Knowledge</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Symbol" /></td>
<td>Obstacles/bricks robots with a grip</td>
<td>Dimension, shape, weight Movements, grip capabilities, battery state</td>
<td>Protecting shape construction Cary the object for one to another location</td>
</tr>
<tr>
<td><img src="image2" alt="Symbol" /></td>
<td>Targets foraging robots</td>
<td>Location, weight, shape Movements, battery state</td>
<td>Movement</td>
</tr>
</tbody>
</table>

Swarm Robotics
## Cloud computing

<table>
<thead>
<tr>
<th>Symbol</th>
<th>SC: Service Component</th>
<th>Knowledge</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="symbol" alt="User applications" /></td>
<td>User applications</td>
<td>the requests for execution (in terms of CPU, minimal space, etc.).</td>
<td>Efficient execution.</td>
</tr>
<tr>
<td><img src="symbol" alt="Remote computer" /></td>
<td>Remote computer CPUs</td>
<td>processing capabilities and a current utilization</td>
<td>Optimal utilisation</td>
</tr>
<tr>
<td><img src="symbol" alt="Local memory" /></td>
<td>Local memory</td>
<td>Capacity, current occupancy</td>
<td>Balanced use</td>
</tr>
<tr>
<td><img src="symbol" alt="Local application services" /></td>
<td>Local application services</td>
<td>available apps at the local computer</td>
<td>Appies “advertising”</td>
</tr>
</tbody>
</table>
# E-mobility

<table>
<thead>
<tr>
<th>Symbol</th>
<th>SC: Service Component</th>
<th>Knowledge</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚶‍♂️</td>
<td>Users</td>
<td>Route plan</td>
<td>to reach different places in a given time.</td>
</tr>
<tr>
<td>🔋🚗</td>
<td>E-vehicles</td>
<td>occupancy and the battery state</td>
<td>to serve users plans, optimize energy consumption</td>
</tr>
<tr>
<td>💡充电桩</td>
<td>Charging stations</td>
<td>Capacity/Reservation plan</td>
<td>optimize its use (high throughput)</td>
</tr>
<tr>
<td>🏠</td>
<td>Park places</td>
<td>Capacity/Reservation plan</td>
<td>optimize its use</td>
</tr>
</tbody>
</table>

Nikola Šerbedžija, 25
# III Modeling Examples (Ensembles)

<table>
<thead>
<tr>
<th>E-Mobility</th>
<th>Cloud Computing</th>
<th>Swarm Robotics</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A user, 2 vehicles, 1 charging station and 3 parklaces</td>
<td>- A user application, 2 remote computers, with local memory of appropriate size and for supporting apples.</td>
<td>- A task: one obstacle, two robots, one target and three foraging robots</td>
</tr>
<tr>
<td>- 3 vehicles that are available for sharing</td>
<td>- 3 remote computers</td>
<td>- 3 free robots with a grip</td>
</tr>
<tr>
<td>- 3 users ready to share vehicles</td>
<td>- 3 different applications with similar processing and memory requirements</td>
<td>- 3 obstacles to be removed</td>
</tr>
<tr>
<td>- 4 basic service components: users, vehicles, charging stations and park places</td>
<td>- 4 basic service components: users applications, remote CPUs, local memory and appis</td>
<td>- 4 basic service components: obstacles, robots with a grip, targets, foraging robots</td>
</tr>
</tbody>
</table>
III SCEL: Modeling language

- A set of programming abstractions that permit to directly represent behaviors, knowledge and aggregations according to specific 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.

### SCEL Syntax

- **Systems:** \( S ::= C \mid S1 \parallel S2 \mid (\nu n)S \)

- **Components:** \( C ::= I[K,\Pi,P] \)

- **Processes:** \( P ::= \text{nil} \mid a.P \mid P1 + P2 \mid P1[P2] \mid X \mid A(p) \)

- **Actions:** \( a ::= \text{get}(T)@c \mid \text{qry}(T)@c \mid \text{put}(t)@c \mid \text{new}(l,K,\Pi,P) \)

- **Targets:** \( c ::= n \mid x \mid \text{self} \mid P \mid l.p \)

- 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 [^*].

SCEL Processes

\[
P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(p)
\]

Processes are the active computational units. Each process is built up from the inert process \text{nil} via action prefixing (a.P), nondeterministic choice (P_1 + P_2), controlled composition (P_1[P_2]), process variable (X), and parameterized process invocation A(p).

The construct P_1[P_2] 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.

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,\Pi,P])
    creates a new component I[K,\Pi,P]
SCEL Targets

c ::= n | x| self | P | l.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.
SCEL Systems and Components

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

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

\[ I.id \in \{n, m, o\} \]

\[ I.active = yes \land I.batteryLevel > low. \]
SCEL Modeling Example: Swarm Robotics

- Francesco Mondada, EPFL,
- Carlo Pincioli, ULB
Actual Robots

Foraging robots

Robots with a grip

Fraunhofer FOKUS
SCEL Example

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

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

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:

\[
AM \triangleq P_{\text{batteryMonitor}}[P_{\text{dataSeeker}}[P_{\text{targetSeeker}}]]
\]

Process \( P_{\text{dataSeeker}} \) is defined as follows:

\[
P_{\text{dataSeeker}} \triangleq \text{qry}(\text{"targetLocation"}, ?x, ?y)@I.task = \text{"task}_i\). \\
\text{put}(\text{"targetLocation"}, x, y)@self. \\
\text{get}(\text{"informed"}. \text{false}@self. \text{put}(\text{"informed"}. \text{true}@self \\

\[
P_{\text{randomWalk}} \triangleq \text{put}(\text{"direction"}, \text{random()} \cdot 2\pi)@self. \text{put}(\text{"termination"})@self
\]

\[
P_{\text{informed}} \triangleq \text{qry}(\text{"targetLocation"}, ?x, ?y)@self. \\
\text{put}(\text{"direction"}, \text{towards}(x, y))@self. \text{put}(\text{"termination"})@self
\]

jRESP Framework for SCEL

SCEL Processes (Threads)  Ports

Policies

Knowledge

Hardware/Virtual Machine

Input devices/Sensors
(GPS, Temperature, Battery level, CPU load...)

Output devices/Actuators

Networks

Fraunhofer FOKUS

Nikola Šerbedžija, 38
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

$$\text{qry}(\text{"lowBattery"}, \text{?low})@self. \text{ if } (\text{low}) \text{ then } \{ P_{then} \} \text{ else } \{ P_{else} \}$$

can be rewritten as follows:

$$\text{qry}(\text{"lowBattery"}, \text{true})@self. P_{then} + \text{qry}(\text{"lowBattery"}, \text{false})@self. P_{else}$$

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

$$P_{lowBattery} \triangleq \text{put}(\text{"stop"})@self. \text{qry}(\text{"gps"}, \text{?x}, \text{?y})@self. \text{put}(\text{"sos"}, x, y)@\mathcal{I}. \text{task = \"task}_i\text{"}). \text{get}(\text{"rescued"})@self. \text{put}(\text{"termination"})@self$$

$$P_{found} \triangleq \text{put}(\text{"stop"})@self. \text{qry}(\text{"gps"}, \text{?x}, \text{?y})@self. \text{put}(\text{"targetLocation"}, x, y)@self. \ldots \text{ execute task i } \ldots$$

$$P_{informed} \triangleq \text{qry}(\text{"targetLocation"}, \text{?x}, \text{?y})@self. \text{put}(\text{"direction"}, \text{towards}(x, y))@self. \text{put}(\text{"termination"})@self$$

$$P_{randomWalk} \triangleq \text{put}(\text{"direction"}, \text{random()} \cdot 2\pi)@self. \text{put}(\text{"termination"})@self$$

jRESP: Implementation of Robot Scenario

```java
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")
            Self.SELF );
    }
}
```
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
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

User Component

Vehicle Component

Infrastructure Component
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
Modeling the Journey

- Drive
- Walk
- Appointment
- Charge
- Park
- Charge

Road Network

<table>
<thead>
<tr>
<th>Loc.</th>
<th>Start. time</th>
<th>Dur.</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>18</td>
<td>3</td>
</tr>
</tbody>
</table>

Appointments

<table>
<thead>
<tr>
<th>Name</th>
<th>Spots</th>
<th>Loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>csp1</td>
<td>7</td>
<td>p</td>
</tr>
<tr>
<td>csr1</td>
<td>4</td>
<td>r</td>
</tr>
<tr>
<td>csr2</td>
<td>0</td>
<td>r</td>
</tr>
</tbody>
</table>

Charging Stations
Programming Model

CIAO Prolog:

```prolog
:-module(journey,[]).
:-use_module(library(lists)).
:-use_module(library(aggregate)).
:-use_module(paths).

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

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

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

journey([[X,Y],[P],[[I,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),
pay(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], [T,E], SoC),
timeSum(Tx, Dx, T, ArrT), ArrT=<Ty,
journey([[Y,Z],[LP,ChEv, [T2,E2], (SoC-E1)],
times([T1,E1],[T2,E2],[T3,E3]):-
T3 = T1 + T2,
E3 = E1 + E2.

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),
pay(X, Y, P, [X], [T1, E1], NewSOC),
timeSum(Tx, Dx, T, ArrT), ArrT=<Ty,
journey([[Y,Z],[LP,ChEv, [T2,E2], (NewSOC-E1)],
times([T1,E1],[T2,E2],[T3,E3]):-
T3 = T1 + T2,
E3 = E1 + E2.
```

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

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

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

Fraunhofer
FOKUS
```
SCEL Modelling: Main Scenario

Calendar

<table>
<thead>
<tr>
<th>9:00</th>
<th>POI2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00</td>
<td>POI1</td>
</tr>
<tr>
<td>12:00</td>
<td>POI3</td>
</tr>
<tr>
<td>14:00</td>
<td></td>
</tr>
<tr>
<td>16:00</td>
<td></td>
</tr>
</tbody>
</table>

POI\(j\) = j-th point of interest

P\(i\) = i-th parking lot

Fraunhofer FOKUS

Nikola Šerbedžija, 49
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:

- $P_1 = \text{ContactParkingLots[Planner[Book[MonitorPlanExecution]]]}$
- $P_1 = \text{ProvideParkingData[ManageBookings]}$
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

Calendar

<table>
<thead>
<tr>
<th>Time</th>
<th>POI1</th>
<th>POI2</th>
<th>POI3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ATTRIBUTES

- **type**: parking lot component
- **position**: position of the park

Nikola Šerbedžija, 53
Parking Lots close to POIs as Ensembles

Calendar

<table>
<thead>
<tr>
<th>Time</th>
<th>POI1</th>
<th>POI2</th>
<th>POI3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td></td>
<td>POI2</td>
<td></td>
</tr>
<tr>
<td>10:00</td>
<td>POI1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ENSEMBLE FOR POI2
Group of components with type parking lot and position at walking distance from POI2

Ensemble for POI2:
- P1
- P2
- P3

Walking distance

Fraunhofer FOKUS

Nikola Šerbedžija, 54
Parking Lots close to POIs as Ensembles

Calendar

<table>
<thead>
<tr>
<th>Time</th>
<th>POI1</th>
<th>POI2</th>
<th>POI3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parking Lots close as Ensembles

**put@EnsembleOfPOI2**
Vehicle component

\[ \text{VEHICLE} = \text{ContactParkingLots} \circ \text{Planner} \circ \text{Book} \circ \text{MonitorPlanExecution} \]
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
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
    //send the booking request to the 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
VEHICLE = ContactParkingLots[Planner[Book[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
  // display to the user the information about the next reservation
  // 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
Parking Lot Component

PARKINGLOT = ProvideParkingData[ManageBookings]

ProvideParkingData =
   // get a request of data about the parking lot
   // 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
   // accept and store the booking
   put("booking", when, howLong, requester)@self.
   put("bookingOutcome", true)@requester.
   // handle next request
   ManageBooking
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
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 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.

---

Franco Zambonelli, UNIMORE, Italy
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.

Franco Zambonelli, UNIMORE, Italy
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 development 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:
- 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).