Smart Object Cooperation through Service Composition

P. Baglietto, M. Maresca - *CIPI University of Genoa*

M. Stecca - M3S srl

C. Moiso - *Strategy Telecom Italia*

Outline

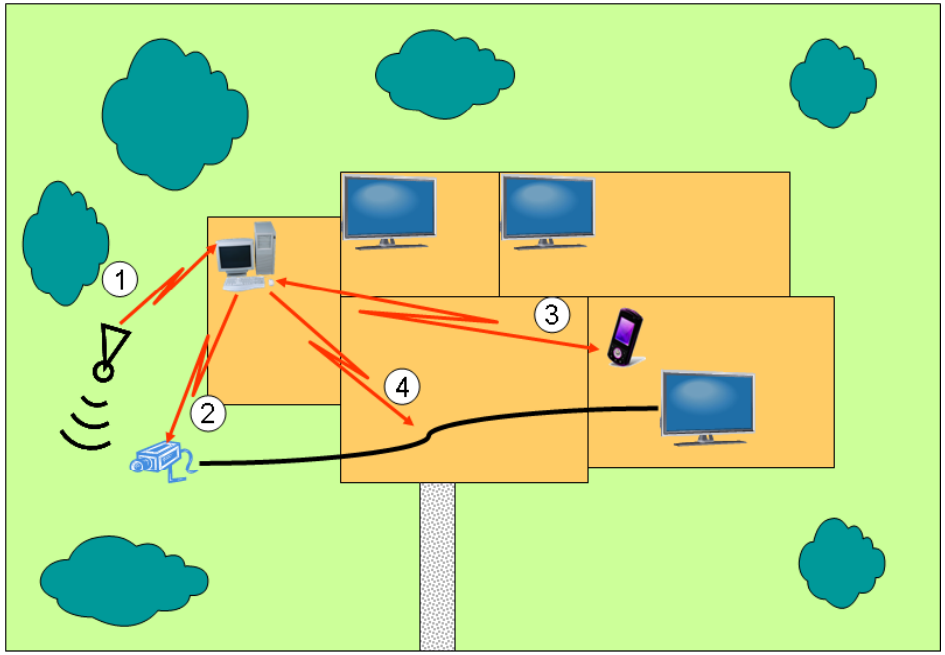
- Introduction
- Requirements
- A platform for Composite Service execution
- The SOCA Platform
- Prototype implementation
- Additional use cases
- Conclusion and Future Work

Introduction (1/2)

- The coordination of smart objects interacting in physical environments is one of the most interesting issues in Internet of Things
- Many mobile objects that people use in daily life (household electrical appliances, sensors, actuators, smartphones, etc.) dynamically join/leave smart objects “communities”
- Smart object coordination can take advantage of service composition techniques, in which composite services are implemented as collections of service components
- Challenges:
  - Different Protocols (e.g., SIP/SIMPLE, XMPP/JINGLE)
  - Asynchronous notification of state changes
  - Unpredictable behavior of Smart Objects
  - Integration with Web Services
- We extend an already existing solution to cope with smart objects cooperation

Introduction (2/2)

Reference Use Case



Components of the surveillance scenario

1. Motion Detection Sensor signals a potential intrusion
2. Video-camera is turned on
3. Interaction with the user through a Smartphone
4. TV-set – Video Camera connection setup

Requirements

1. ***Multiprotocol support*** (in particular SIP/SIMPLE vs. XMPP/JINGLE);

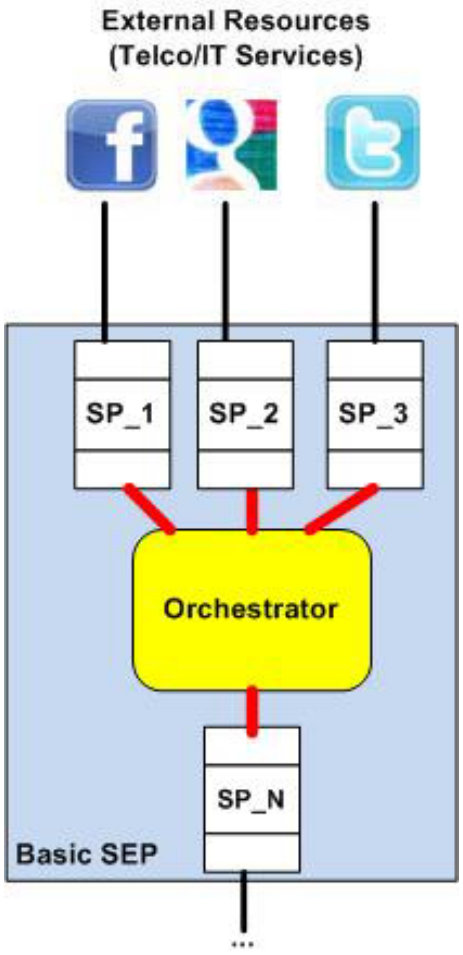
2. ***Aggregation of external services*** (belonging both to the Internet and to the Telecom sphere);

3. ***Dynamic activation of object-to-object communication streams:*** (either in 1st party mode by interacting with one of the objects to be interconnected or in 3rd party mode);

4. ***Dynamic smart object join and leave:*** mobile objects may join and leave the environment;

5. ***Dynamic change of state of smart objects:*** smart objects may change their status (e.g., enter a power-saving or a stand-by mode).

A platform for Composite Service execution (1/2)



We already designed a platform for Telco/IT service orchestration*

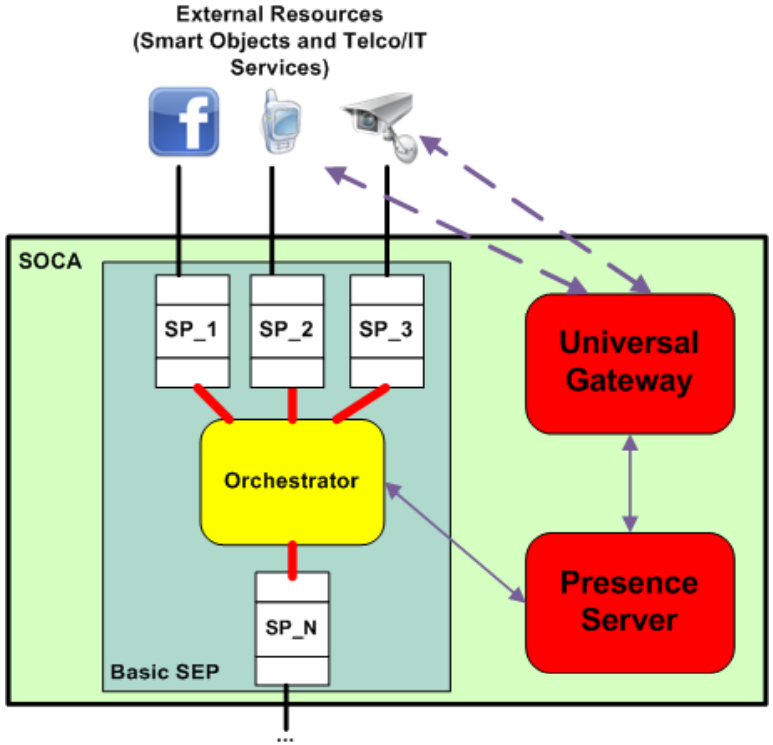
- It supports the combination of services implemented by means of different technologies (e.g., SOAP/RESTful Web Services, RSS Feeds, Atom) thanks to the presence of an "abstraction layer" (i.e., a set of Service Proxies - SPs);
- It allows to combine services belonging both to the Internet sphere and to the Telecom sphere;
- The basic components of the Composite Service Logic (CSL) interact through an "event/action" paradigm (see next slide);
- It supports scalability, fault tolerance, and security.

*Stecca M. et al, "Scalable Orchestration of Telco/IT Mashups", ICIN 2009

A platform for Composite Service execution (2/2)

- The system allows to fulfil requirements 1 and 2, namely:
  - ***Multiprotocol support***;
  - ***Aggregation of external services.***

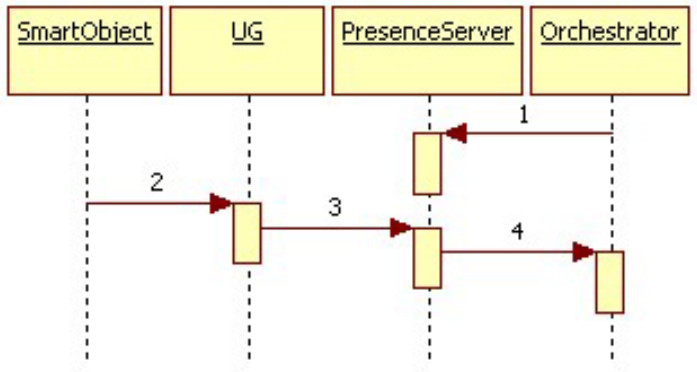
The SOCA platform (1/4)



- We extended the basic architecture introducing two components:
  - Universal Gateway - UG;
  - Presence Server – PS.
- The **S**mart **O**bject **C**ooperation **A**rchitecture - SOCA fulfils all the requirements listed on Slide 5.

The SOCA platform (2/4)

- The Universal Gateway:
  - supports signaling and media content streaming among smart objects;
  - ensures the interoperability among smart objects based on different protocols (e.g., SIP/SIMPLE and XMPP/JINGLE);
  - might interact with the Presence Server in order to discover the protocol implemented by a specific object.
- The Presence Server:
  - manages an inventory of available objects dynamically joining/leaving the community (e.g., wearable sensors, smartphones, cameras, introduction/ removal of devices) and change their status (e.g., turned on/off, online/offline);
  - Is needed to keep trace of smart objects availability and status changes;
  - Interacts with the Orch component in order to notify the availability and the status changes of the devices. Thanks to the PS, the Orch is constantly aware of the current state of each smart object.

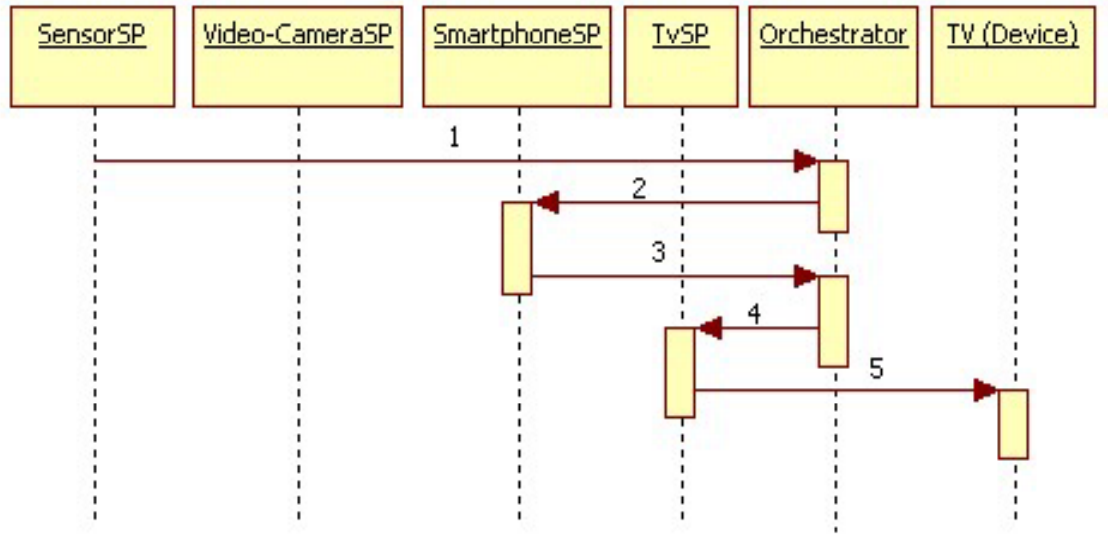


The SOCA platform (3/4)

What if a smart object is not available at run-time?

- We envision three different policies, namely:
  - The Orchestrator may “skip” the interaction with the unavailable device for this execution.
  - The Orchestrator may “abort” the execution of the service and send an alarm to the appropriate users.
  - The Orchestrator may “delay” the progress of the Composite Service to the time at which the smart object currently missing will become available/active again.

→ The UML diagram shows the interactions among components in the reference scenario example.



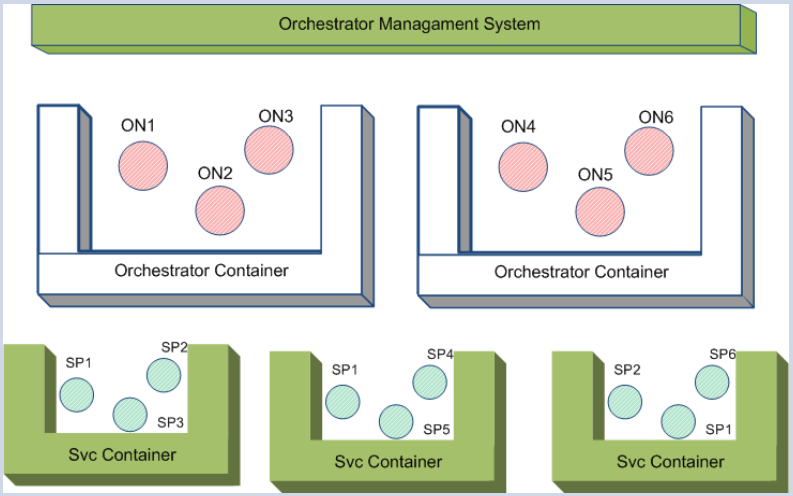
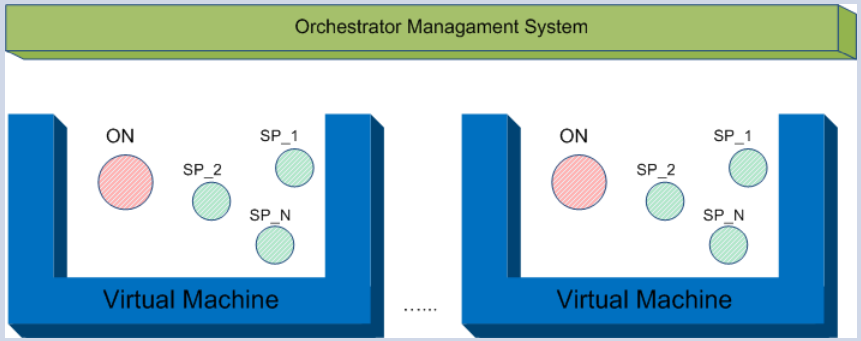
The SOCA platform (4/4)

Requirement fulfillments

Requirement	Solution
Multiprotocol support	Abstraction layer (Service Proxy)
Aggregation of external services	Abstraction layer (Service Proxy) + Orchestrator
Dynamic activation of object-to-object communication streams	Abstraction layer (Service Proxy) + Universal Gateway
Dynamic smart object join and leave	Orchestrator + Presence Server
Dynamic change of state of smart objects	Orchestrator + Presence Server

Prototype implementation (1/4)

We implemented two different versions of the SOCA platform

Framework-based version	Monolithic version
	
• Replicated instances of the Orchestrator and SPs are deployed on different nodes • A Session is usually executed on different nodes to balance the workload • Communication overhead due to the framework • Framework-based Fault Tolerance	• Each node runs a “complete” SEP • A Session is completely executed on one node • NO Communication overhead due to the framework (the components are implemented as POJO) • Fault Tolerance given by the Virtualization Environment

Prototype implementation (2/4)

The Pseudocode of the ***original*** event routing algorithm implemented in the Orchestrator

```
Input Params:
   Session_ID, Event_ID, SenderSP_ID, Props
Upon Event reception:
1. Retrieve the CSL using Session_ID as a key;
2. Search for next action(s) to be invoked in the
   retrieved CSL using the <SenderSP_ID, Event_ID> pair
   as a key;
3. Update the session properties appropriately (e.g.,
   copy the output properties of the previous SP in the
   input properties of the following SP);
4. Invoke the action(s) retrieved at point;
```

Prototype implementation (3/4)

The Pseudocode of the ***NEW*** event routing algorithm implemented in the Orch

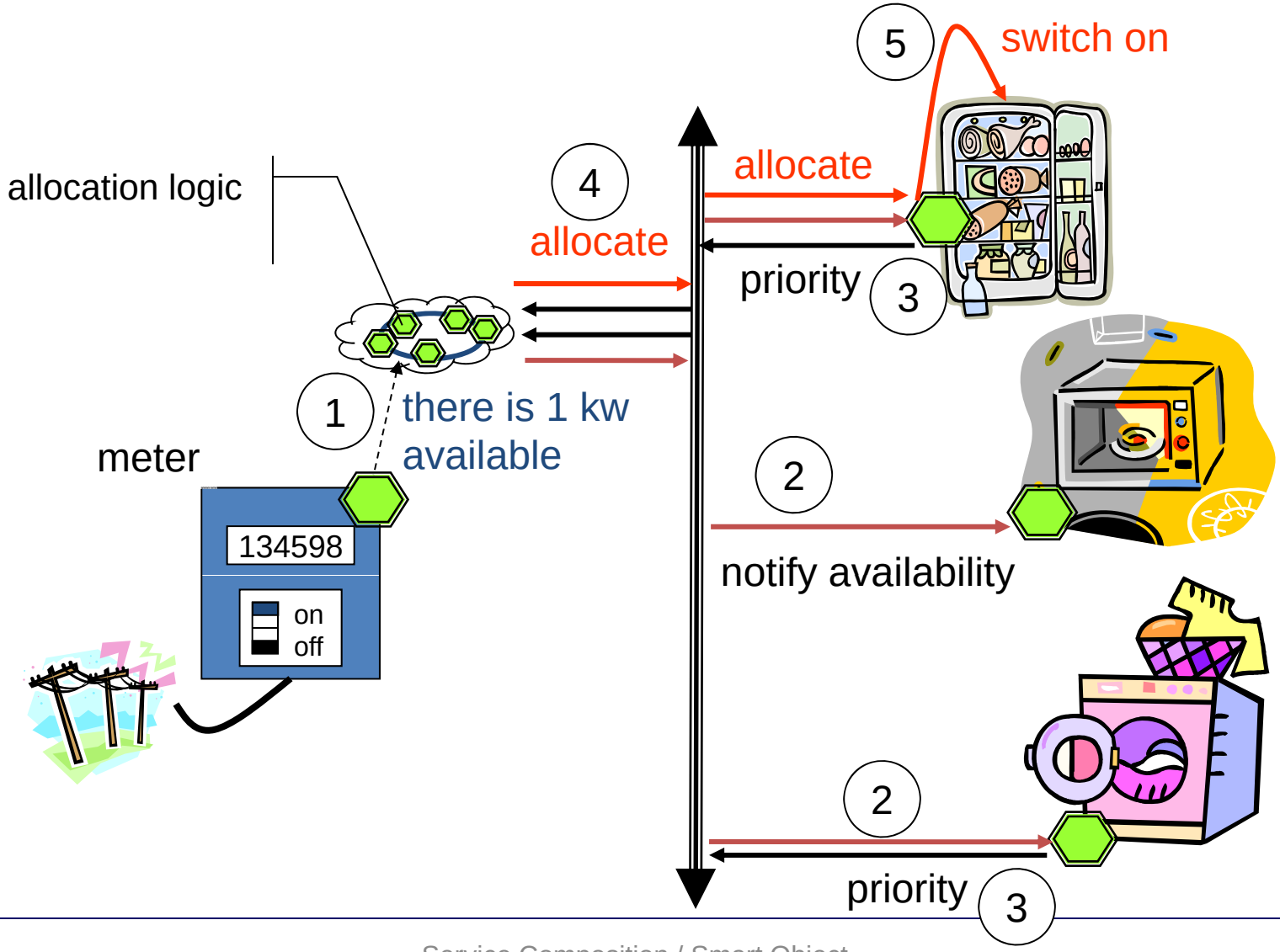
```
Input Params: Session_ID, Event_ID, SenderSP_ID, Props Upon Event reception:
1.  Retrieve the CSL using Session_ID as a key;
2.  Search for next action(s) to be invoked in the CSL obtained at point 1
    using the pair <SenderSP_ID, Event_ID> as a key;
3.  Update the session properties appropriately (e.g., copy the output
    properties of the previous SP in the input properties of the following
    SP);
4.  If the next action involves an interaction with a Smart Object:
     4.1 Retrieve the status of the Smart Object from the SOST;
     4.2 If the Smart Object is available;
         Invoke the action(s) retrieved at point 2;
     4.3 Else
         Retrieve the policy associated to the current Session_ID from the PT;
         4.3.1  Switch (Policy)
           Case (Skip):
             •   Retrieve from the CSL the action(s) related to the SP following
                 the SP interacting with the missing device;
             •   Invoke the action(s) retrieved in the previous step;
           Case (Abort):
             •   Destroy the current Session;
             •   Notify the user about Session abortion;
           Case (Delay):
             •   Add a new entry <Session_ID, SessionInfo, WaitingForDeviceID>
                 to the PST;
```

Prototype implementation (4/4)

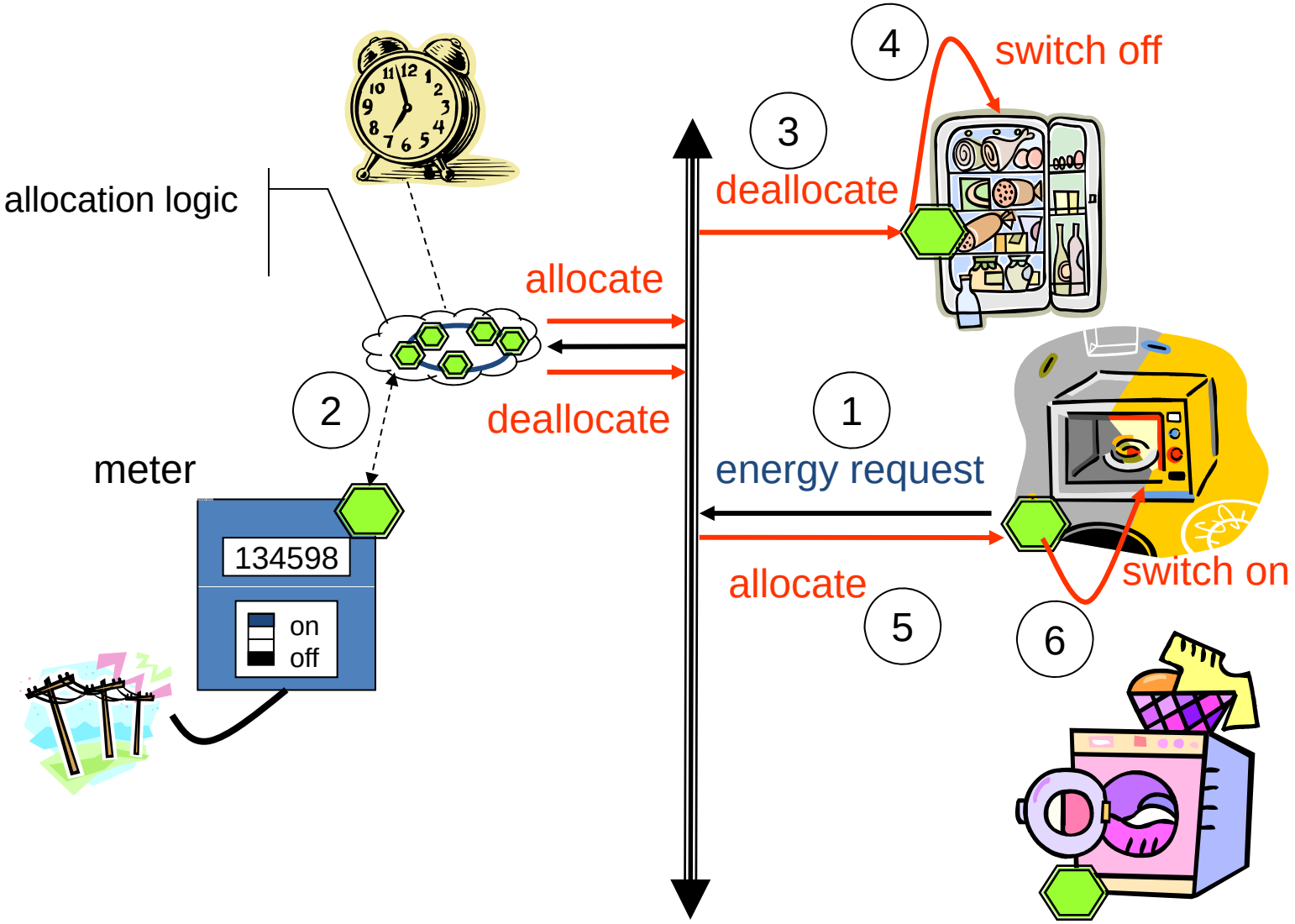
The Pseudocode of the method invoked by the Orchestrator during the interaction with the PS

```
Input Params:
    Device_ID, NewDeviceStatus
1. Update the entry related to the device
   identified by the parameter Device_ID (i.e.,
   set the status to NewDeviceStatus) in the
   SOST;
2. If (NewDeviceStatus==AVAILABLE)
    2.1 Retrieve all the sessions waiting for the
        availability of that device in the
        Pending SessionIDs Table using Device_ID
        as a key;
    2.2 For each entry retrieved at point 2.1
        • Extract the Session_ID and the
          SessionInfo fields from the entry;
        • Restore the session state stored in
          the SessionInfo field;
        • Continue the execution of the session
          identified by the Session_ID field;
        • Remove the entry from the PST;
```

Another use case: energy allocation (1/2)



Another use case: energy allocation (2/2)



Some advances

- Natively multi-protocol:
  - not a preferred protocol (e.g., SIP, XMPP, etc.);
- Open to different deployment options:
  - at the edge or in the “cloud”;
- Exploitation of a programmable (via GUI) “general purpose” service composition environment;
- Mix of service-to-object and object-to-object coordination;
- Seamless composition of Objects’ features and virtual services available over the Web.

Conclusion & Future Work

- We listed a set of requirements to be fulfilled by a platform coping with smart object cooperation;
- We discussed how the Service Composition paradigm can support the cooperation of smart objects to develop applications in the domain of the Internet of Things;
- We extended the platform that we developed in the past for Service Composition with two additional components, namely the Universal Gateway and the Presence Server, in order to support smart object cooperation effectively;
- Future works:
  - enhancing SOCA platform with additional features like the dynamic binding of smart objects

    E.g., all the object of type ‘x’, the ‘nearest’ object, the ‘cheapest’ object, etc.
  - investigating deployment options, e.g.:
    - on gateways (e.g., home/access gateway), on servers in the “cloud”, on devices (e.g., smartphones or tablets);
    - impacts in the deployment of adaptation functions for near-range/long-range protocols

The End

Michele Stecca

m.stecca@cipi.unige.it

Follow me on Twitter: **@*steccami***