An execution platform for Service Composition

Outline

- Introduction
- Overview of the platform
- Service Composition Execution Platform
- The SEP & the IoT

Introduction (1/4)

The OPUCE Project

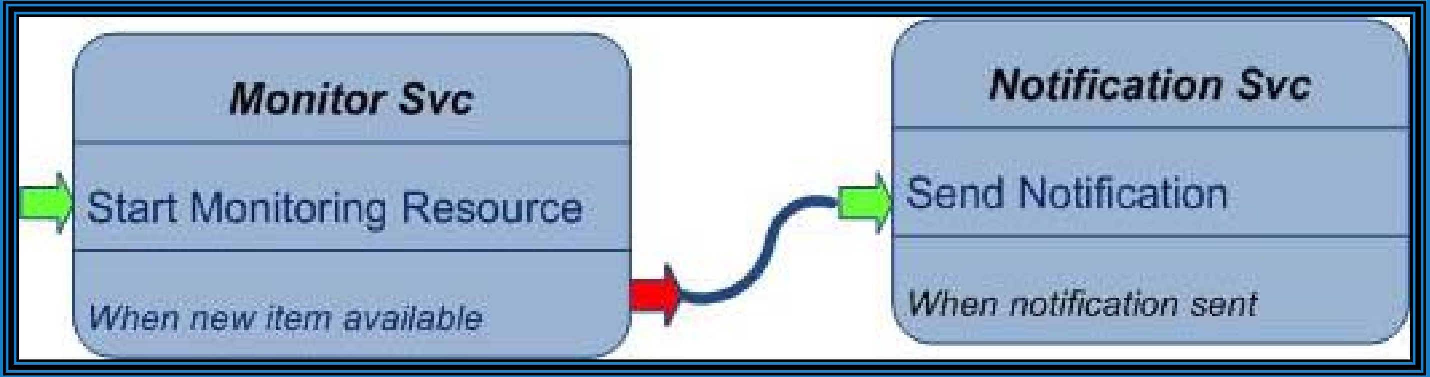
- We started working on Service Compositions in the context of the FP6 project called OPUCE (Open Platform for User-centric service Creation and Execution)
- Coordinator: Telefonica
- Other Partners: Telecom Italia, Portugal Telecom Inovacao, Alcatel Lucent, NEC, Huawei, UPM, etc.
- We led task 3.4 (Service Execution Platform)

Reference Scenario

- Availability of contents and services through technologies typical of the Web 2.0 philosophy such as RSS Feed, Atom, REST-WS, SOAP-WS, etc. See programmableweb.com for a list of more than 5000 Web APIs
- Availability of tools for the rapid development of convergent Composite Services (a.k.a. Mashups) that combine different resources such as Yahoo Pipes!, JackBe Presto, etc.
- New business models (e.g., Apple Store for applications, pay-as-you-go Cloud Services, etc.)
- Open Data*
- Internet of Things*

Introduction (2/4)

Service Taxonomy and Mashup Patterns: the "Monitor Resource + Notification" pattern

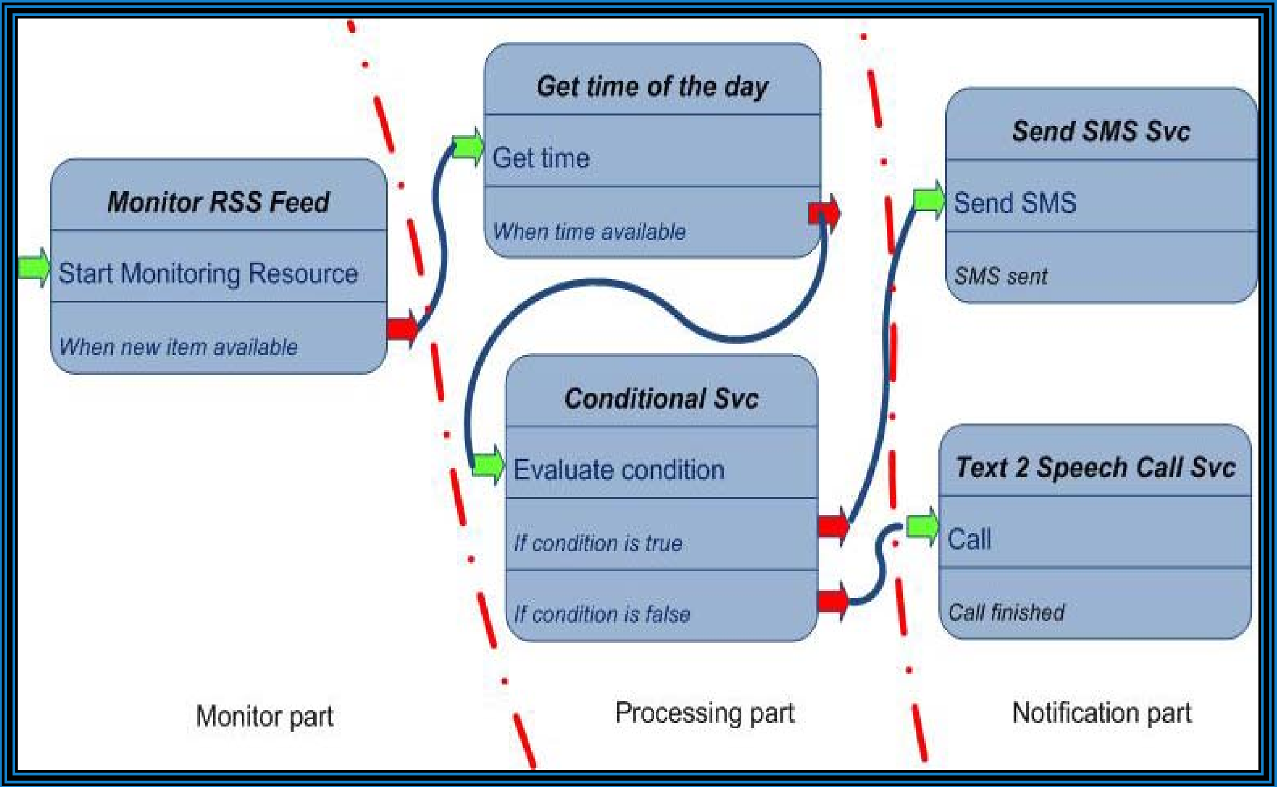


Examples:

- ***Reminder Mashup***
- ***Alert Mashup***
- ***Notification of interesting events Mashup***

Introduction (3/4)

Service Taxonomy and Mashup Patterns: the "Monitor Resource + Processing + Notification" pattern



Examples:

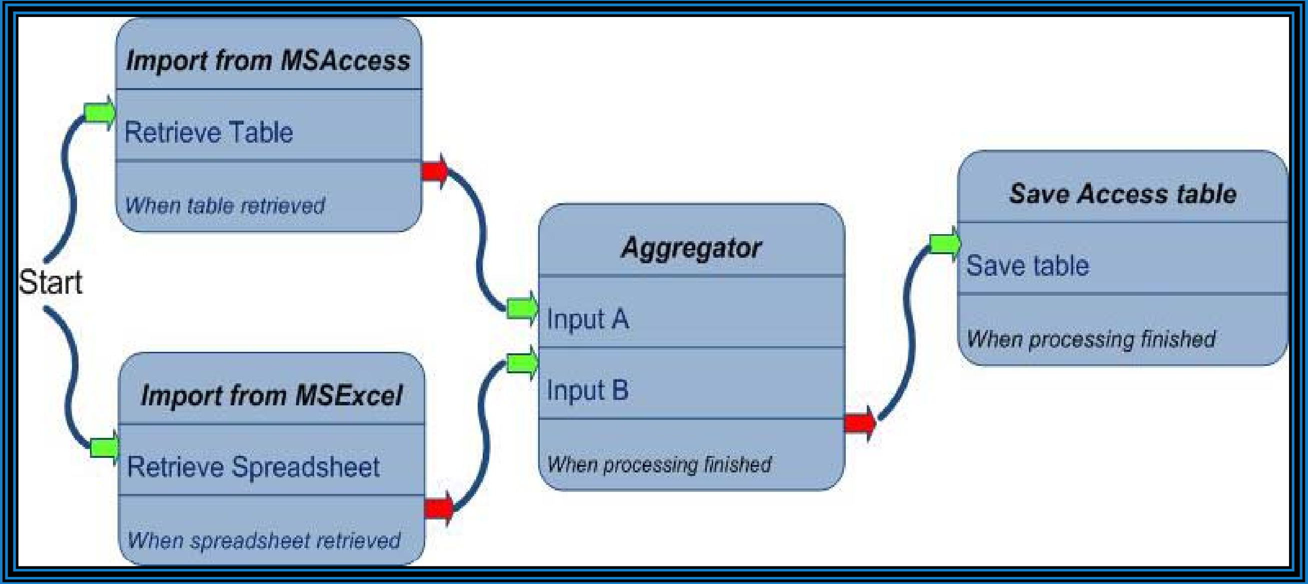
- Time or Location or Presence dependent Notification Mashup
- Non repudiation Mashup

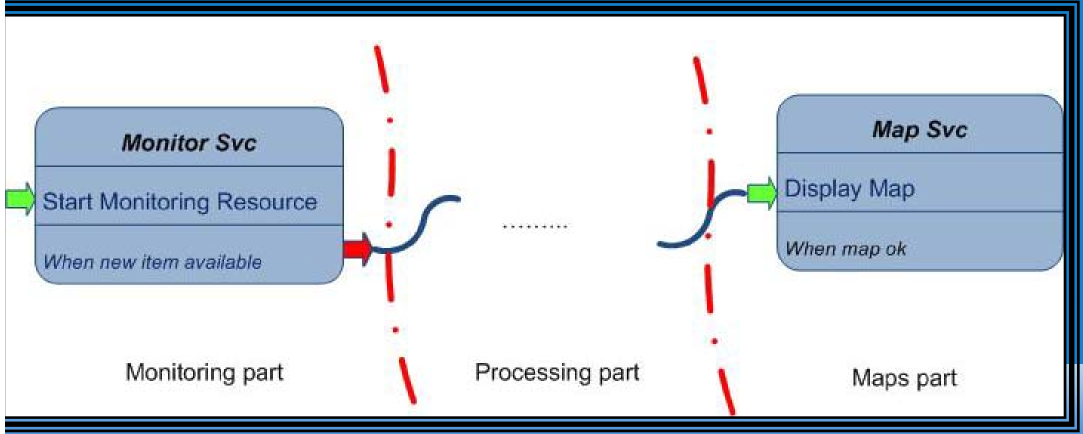
Introduction (4/4)

Service Taxonomy and Mashup Patterns: traditional Data Mashup and "something" + Map Mashup

Examples:

- Data Fusion
- Data Migration
- …



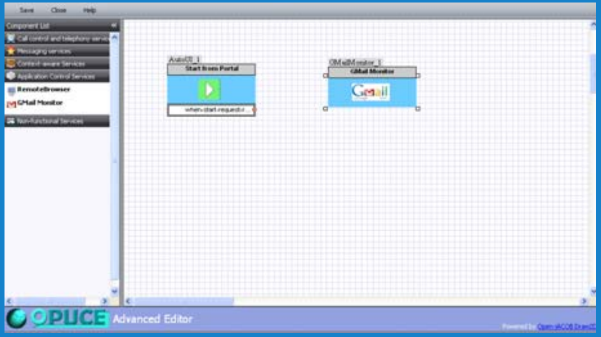


Examples:

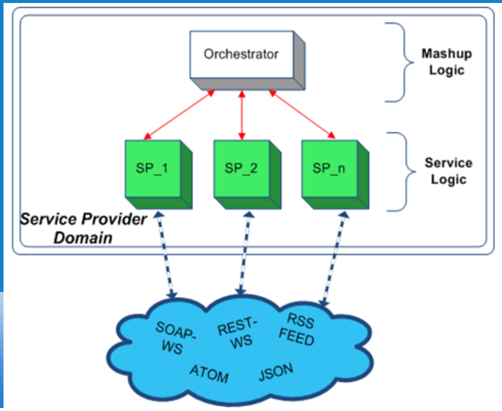
- Traffic on Map
- Wheather on Map
- …

Platform overview (1/6)

Service Creation Platform



Service Repository



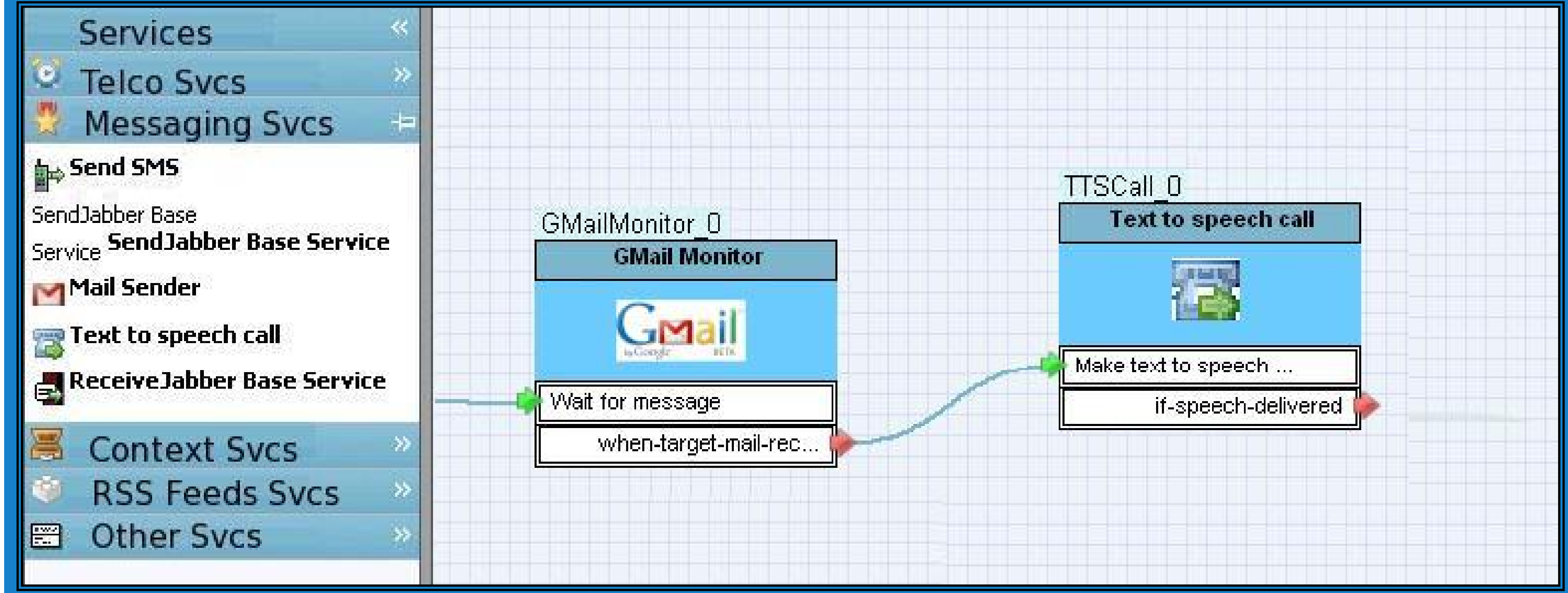
Service Execution Platform

Platform overview (2/6)
Service Composition Lifecycle

1. A SC is created by means of a graphical Service Creation Platform and stored as a XML file in a repository.
2. The new SC is deployed into a software system - the **Service Execution Platform** – that is in charge of executing the composite services.
3. The SC is now ready to be used by final users. There are two phases to take into account:
   - **Activation:** the SEP is configured by the user (i.e., he sets the input properties of the service) to be ready to execute the SC when a new initial event occurs.
   - **Execution:** when a SC previously activated by the user receives a new initial event, the actual execution of the service logic takes place.

Platform overview (3/6)

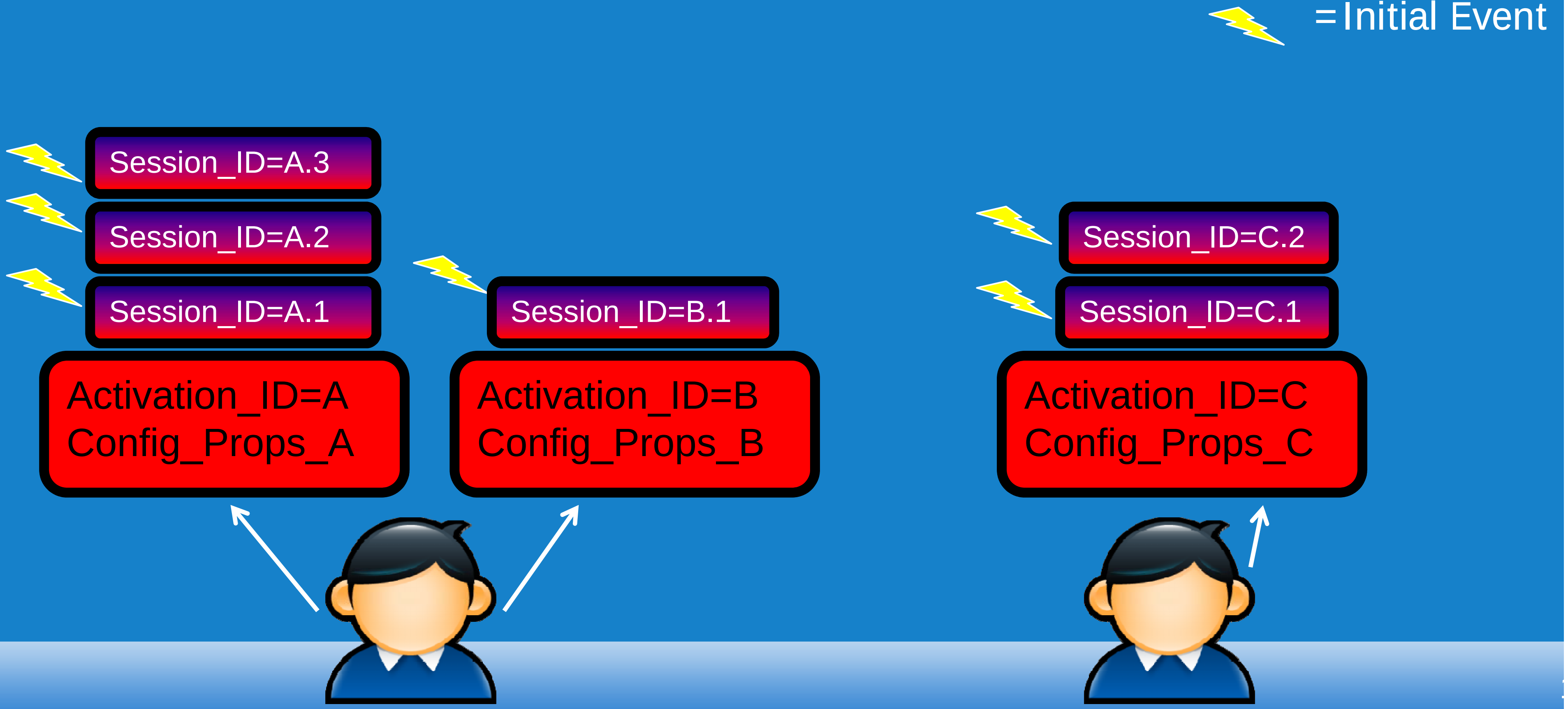
The Service Creation Platform



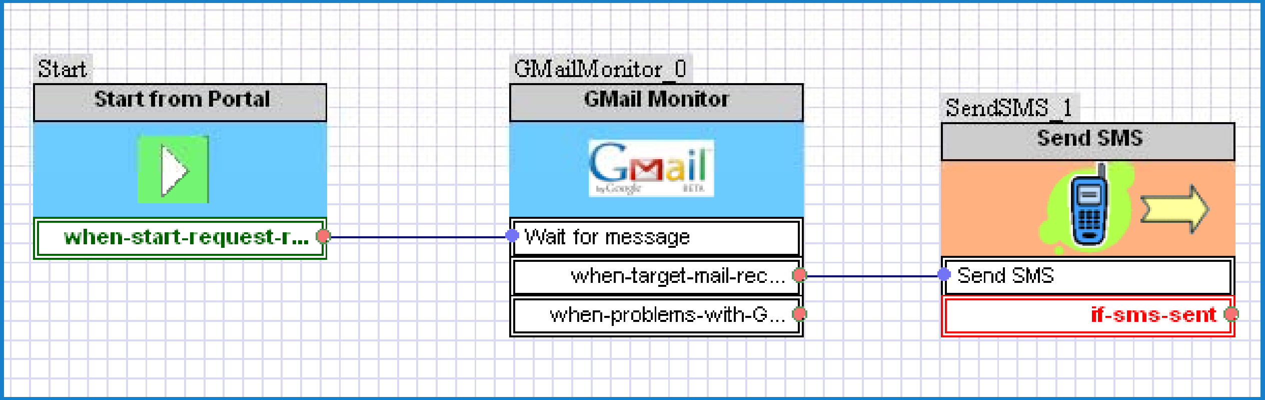
Platform overview (4/6)

Activation→ Service Composition configuration

Session → Actual Service Composition Execution



Platform overview (5/6)



Config_Props_A = <E_mail_Filter, "U2 Concert">, <PhoneNumb, "123">, …

Config_Props_B = <E_mail_Filter, "Discount">, < PhoneNumb, "456">, …

Config_Props_C = <E_mail_Filter, "Katy Perry">, < PhoneNumb, "325">, …

Activation_ID=A
Config_Props_A

Activation_ID=B
Config_Props_B

Activation_ID=C
Config_Props_C

Platform overview (6/6)

Session State: an array of <Property_Name, Property_Value> pairs. The *single assignment* approach is used.

Session State Management:

→ Centralized vs Distributed

Scalability: Session State management strategies affect the SEP scalability

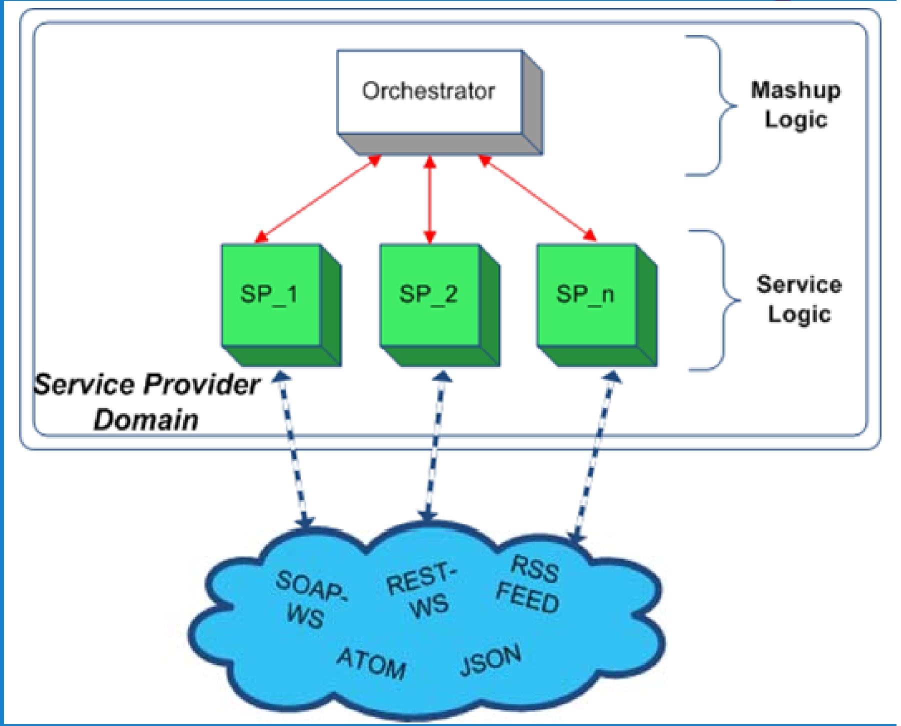
SEP Design (1/5)

Requirements for the SEP

1. Automatic **Scalability**, to support the simultaneous execution of a very large numbers of Mashup instances on variable size hardware systems.
2. **Fault Tolerance**, to guarantee the levels of reliability and availability typical of the Telecoms Operators (99.999%).
3. **Low latency**, to effectively support also the execution of long sequences short lived Telco services.
4. **Authentication, Authorization and Accounting**, to support the seamless integration of the Mashup with the AAA modules already existing in the owner's platform environment.
5. **Management,** to have the complete control of the platform resources, to control Mashup activation/execution as well as to allow the platform administrator to perform the appropriate management actions (e.g., enforcing SLA rules).

SEP Design (2/5)

SEP Architecture



The Orchestrator knows the logic of the deployed Service Compositions

Service Proxies (SPs):

- wrap only one internal/external resource
- decouple the SC logic from the specific external implementation

Different technologies for the SEP implementation (BPEL, WS, JMS, Compact)

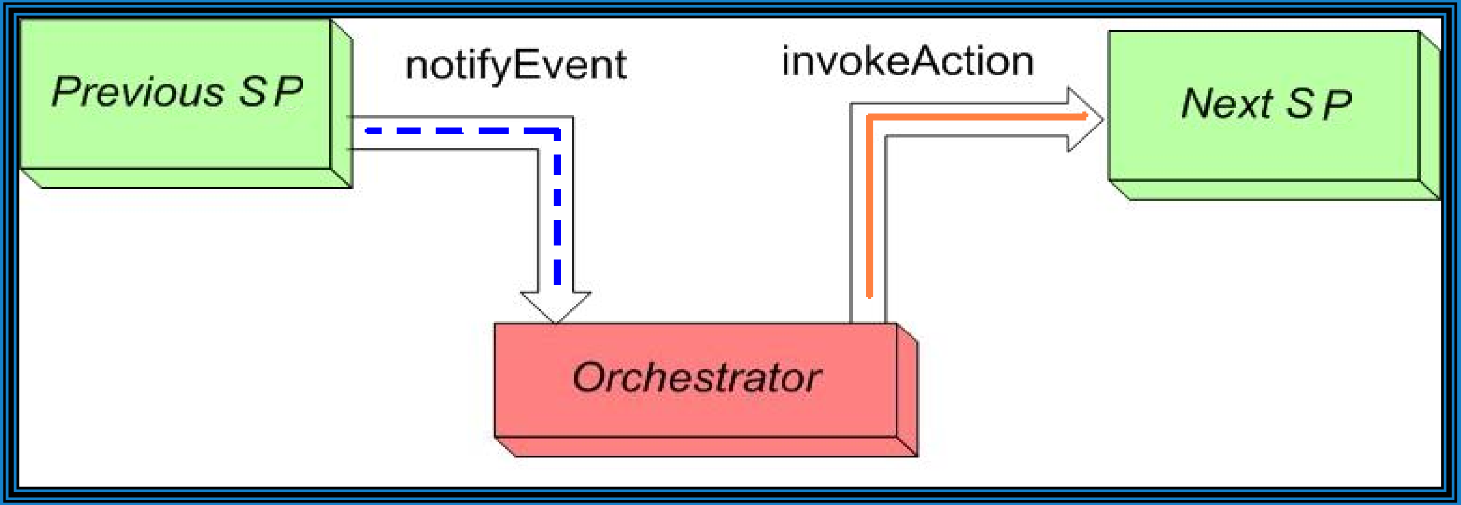
SEP Design (3/5)

The runtime model

Each edge that links two basic blocks in the graphical service description is implemented by means of two primitives:

→ a *notifyEvent* call from the SP to the Orchestrator;

→ an *invokeAction* call from the Orchestrator to the next SP(s)



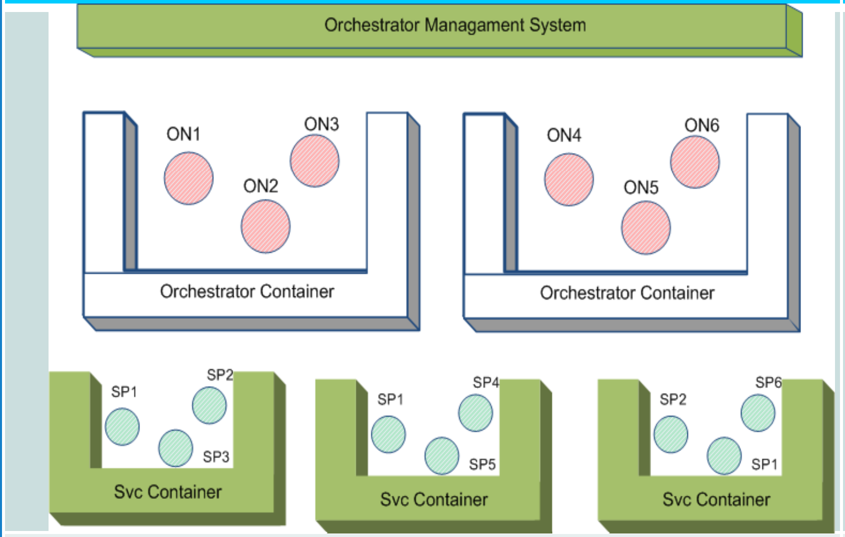
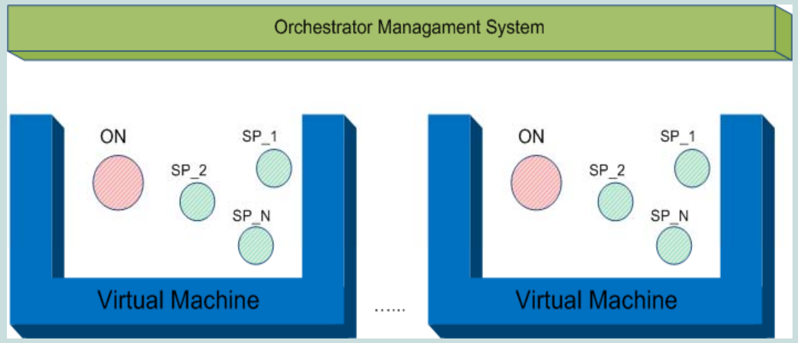
SEP Design (4/5)

Different SEP implementations

- ❖ Based on the Business Process Execution Language - BPEL
- ❖ Based on an ad-hoc lightweight Orchestrator
- ❖ Based on the Java Messaging Service (JMS) technology
- ❖ Based on POJOs (a.k.a., the "Compact SEP")
  - – Ongoing activities: definition of the most suitable concurrency model (e.g., Non-blocking I/O?)
  - – Deployment of the SEP in Virtualized Environments to take advantage of autoscaling mechanisms and fault tolerance mechanisms (e.g., VMware Fault Tolerance)

SEP Design (5/5)

Comparison between the two approaches

Framework-based version	*Compact version*
 Orchestrator Managament System ON1 ON2 ON3 Orchestrator Container ON4 ON5 ON6 Orchestrator Container SP1 SP2 SP3 Svc Container SP1 SP4 SP5 Svc Container SP2 SP6 SP1 Svc Container	 Orchestrator Managament System ON SP_1 SP_2 SP_N Virtual Machine …… ON SP_1 SP_2 SP_N Virtual Machine
• 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

The SEP & the IoT (1/5)

Scenario

- Smart objects, Sensors, Actuators available for composition
- Pachube repository
- Real-time Open Data (e.g., traffic sensors, parking sensors)
- Ongoing Projects about Service Composition in the IoT domain
  - Web Of Things (SAP/ETH Zurich, Switzerland)
  - Casena (Assisted Living) (Alcatel-Lucent Leuven, Belgium)
  - Twitter of Things (Deutsche Telekom Berlin, Germany)
  - Paraimpu (CRS4 Cagliari, Italy)
  - Others…

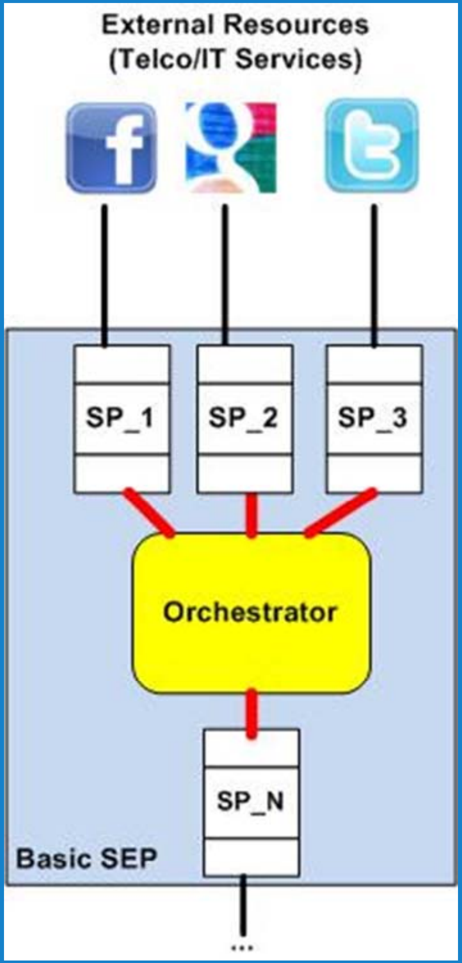
→ We designed a possible IoT-oriented SEP

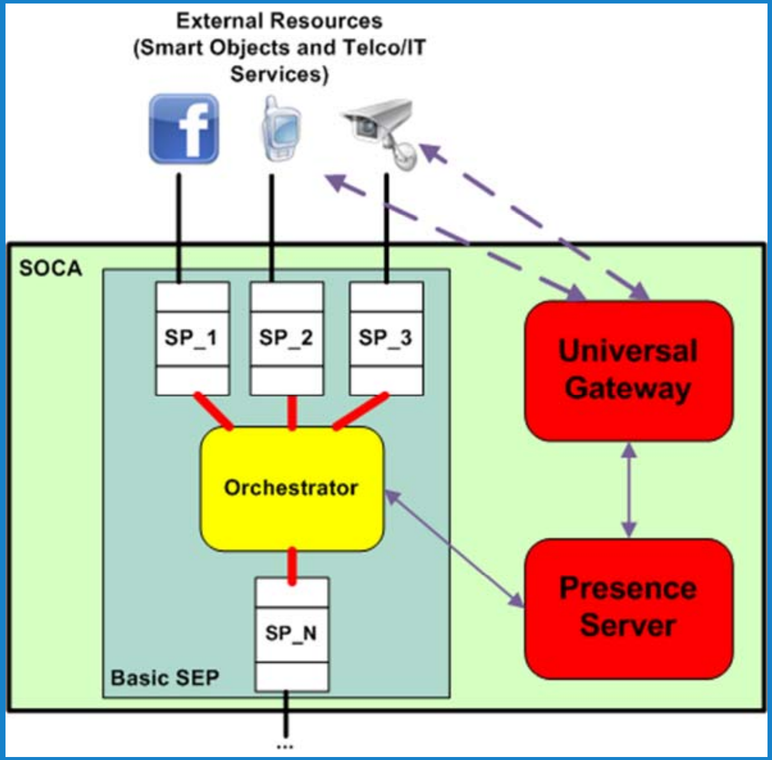
The SEP & the IoT (2/5)

Some requirements

- ***Aggregation of services*** (belonging both to the Internet and to the Telecom sphere) with smart “physical” objects;
- ***Dynamic smart object join and leave:*** mobile objects may join and leave the environment;
- ***Dynamic change of state of smart objects:*** smart objects may change their status (e.g., enter a power-saving or a stand-by mode);
- ***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);

The SEP & the IoT (3/5)





Baglietto, Maresca, Stecca, Moiso, "Smart Object Cooperation through Service Composition", ICIN 2011

The SEP & the IoT (4/5)

- 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 Universal Gateway:
  - supports signaling and media content streaming among smart objects;

The SEP & the IoT (5/5)

- 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.

Thank you!

Michele Stecca
E-mail: m.stecca@m3s.it
Twitter: @steccami