Specific Targeted Research Projects (STReP)

SocIoTal

Creating a socially aware citizen-centric Internet of Things

FP7 Contract Number: 609112

WP4 – Citizen Empowerment

Deliverable report

Contractual date of delivery:**31 May 2014**

Actual submission date: **6 June 2014**

Deliverable ID:	**D4.1**
Deliverable Title:	**Feature specification of intuitive user and developer environment**
Responsible beneficiary:	**CRS4**
Contributing beneficiaries:	**CRS4**, **CEA**, **DNET**, **UNIS**
Estimated Indicative Person Months:	

Start Date of the Project: 1 September 2013 Duration: 36 Months

Revision: 1.0

Dissemination Level: Public

Document Information

Document ID: SOCIOTAL_D4.1_V1.0
Version: 1
Version Date: 04/06/2014 17.31
Authors: Davide Carboni, Alberto Serra, Antonio Pintus (CRS4), Srdjan Kro (DNET), Michele Nati (UNIS), Levent Gurgen, Yazid Benazzouz (CEA-LETI)
Security: Public

Approvals

	Name	Organization	Date	Visa
Project Management Team	Klaus Moessner	UNIS	06/06/2014	
Internal reviewer	Antonio Skarmeta	UMU	06/06/2014	
Internal reviewer	Ignacio Elicegui Maestro	UC	06/06/2014	
Internal reviewer	Rob Van Kranenburg	RD	06/06/2014	

Document history

Revision	Date	Modification	Authors
0.1	16 Jan. 14	TOC and initial assignments	CRS4
0.2	21 Jan 14	Added a proposal for feature spec and mockup	CRS4
0.3	1 Feb 14	Minor changes	CRS4
0.4	7 Feb 14	Some features listed,one section from DNET	CRS4 + DNET
0.5	25 Feb 14	More features for User Environement, first contribution of features and mockups for the development environment, section from UNIS on security guidelines	CRS4 + CEA + UNIS
0.6	6 Mar 14	Architecture of DevelEnv plus other minor changes in other sections	CEA + CRS4
0.7	7 Apr 14	Consolidate content in all sections	all
0.8	21 May 14	All missing contents	
0.9	23 May 2014	Ready for internal review	
1.0	6 Jun 14	Ready	

Content

List of tables

List of figures

Description of the deliverable content and purpose

This deliverable contains feature specifications of the two SocIoTal tools called: **SocIoTal End-user environment** and **SocIoTal Developer environment**. As the names suggest, these components are targeted to different groups of people, but both contribute towards lowering the barrier for bottom-up adoption of IoT for citizens.

The deliverable is structured as follows:
Section 1 - provides a general overview of the main approach for the whole work and describes the relations with other work packages. In particular, it makes a first analysis of security and reputation issues and mentions the main dependency with the on-going work in SocIoTal work packages WP2 and WP3.

Section 2 - is focused on the user environment (UserEnv). This section starts with some challenging bottom-up use cases, then depicts corresponding mockups of functionalities, and makes a comparative analysis of the existing tools that may be relevant to understand the current state-of-the-art and the proposed advancements. Finally, the section describes in more details the features and the architectural view of the UserEnv.

Section 3 - has a structure similar to Section 2, but is focused on the Service Developer Environment (DevelEnv). It contextualizes the tool in its relevant state-of-the-art first and then describes more detail the feature specification and relevant architectural views.

Contribution of the Deliverable to the overall project objectives

The main SocIoTal objective is to unleash the full potential of IoT, going beyond the enterprise centric systems and moving towards a citizen inclusive IoT in which IoT devices and contributed information flows provided by people are encouraged. To lower the barrier of IoT bottom-up development and adoption, some key aspects must be tackled: security, control, transparency and simplicity. These aspects are addressed in different blocks of the SocIoTal architecture and are also integrated and made available by means of tools. This work package is about the tools, and the features of such tools contributing to the general objectives of the SocIoTal project. In particular, below are listed the sub-objectives that are directly addressed in this work package:

O4.1: To define and develop tools that will provide intuitive mechanisms to the user for expressing the way in which they want their environment behave.
This objective will be mainly addressed by the *Task T4.1: Intuitive end user environment (M4-M36)* that will define and develop tools that will provide intuitive mechanisms to the user for expressing the way in which they want their environment behave. We refer to these tools as the SocIoTal User Environment or shortly **UserEnv** further on. Among the components of the SocIoTal dashboard is the personal workspace in which users can compose data coming from city sensors with data coming from the circle of his/her own devices and from those of his/her contacts (according to sharing policies). Integrating IoT and mainstream social networks is also a key aspect.

O4.2: **To define and develop tools aimed at service developers**. This objective will be mainly addressed by the task *T4.2: Service creation environment (M4-M36).*
This task will tackle this issue by providing a novel model-driven approach for rapid dependable creation of software for service developers (called **DevelEnv** further on). The goal of this task is therefore twofold:

- To provide service component models, APIs and guidelines to service developers to transform the IoT devices into service producers
- To provide tools to compose IoT services based on a model-driven approach in order to guarantee the correctness of the created services. Besides verification at design-time, the behaviour of composed services will also be verified at run-time.

O4.3 To provide integration of trust/reputation and privacy-preserving communication mechanisms inside the tools. This objective is only partially addressed in this deliverable (which reports about task 4.2 and task 4.1) and will be addressed in the task 4.3 which will start in M19. In particular, the task 4.3 will be focused on the integration with protocols for securing communications, tracking the use of data and devices by social circles, policies and mechanisms to grant trust and to manage reputation. In this document are introduced, with a coarse and immature design, some of the future integrations with the work packages WP2 and WP3. To not wait until M19, the output from WP2 and WP3, as soon as they are consolidated, will be taken as input for the refined design and development of WP4 tools.

Section 1 - Definitions, Dependencies, and Guidelines

1.1 Dependencies on other SocIoTal deliverables and tasks

The main inputs to Task 4.1 and 4.2 and in particular deliverable D4.1 (as shown in Figure 1) are coming from D1.1 where scenarios, use cases and services are presented together with the main requirements. These inputs are driving the work in T4.1, not only in terms of the terminology used (IoT-A [1]), but also in terms of the functionality expected from the end user environment.

In addition to this formal input, further input is gathered from WP6 activities. WP6 is organizing several events, workshops and the so called Meetups, involving citizens and developers' communities throughout project duration. The aims of these activities will be not only to present the project, but to truly interact with all participants and gather feedback regarding the potential usage of the project outputs as well as wishes, potential functions and features that the SocIoTal solution should provide. These inputs are mainly gathered through internal reports, summarizing outcomes from different events.

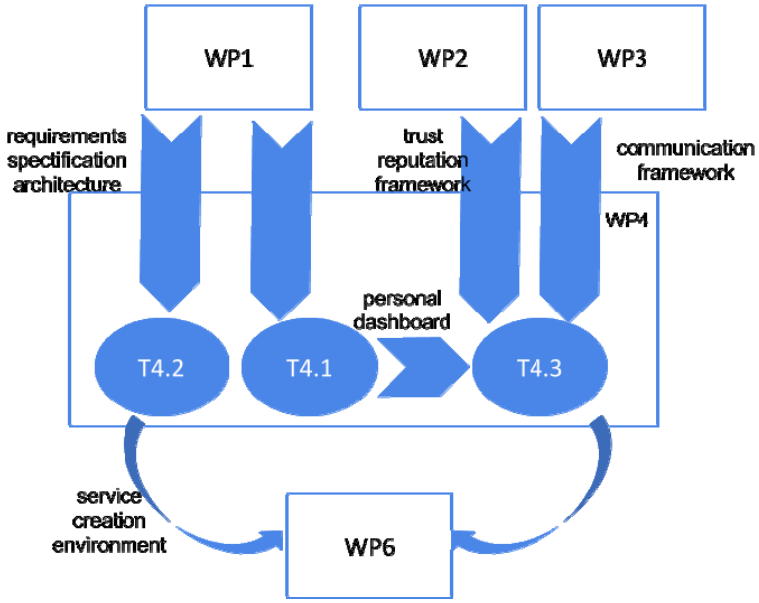

Figure 1 Collaboration flow among SocIoTal workpackages. View centred on WP4

1.2 Assumptions and definitions

It is important to understand and consolidate the level of abstraction where the WP4 tools are going to operate. The proposal here is to design the UserEnv as an API broker that consumes data from some other API and push data to another API. These APIs are provided by platforms or services actually storing the data collected from the devices. This approach assumes that physical devices connect to IoT gateway services that virtualize these, making them accessible via API calls. This design allows paving the ground for the high level features of the WP4 tools without facing device level details that are still undetermined and dependent on the first version of SocIoTal architecture.

In a similar way, the DevelEnv is assumed to be decoupled as much as possible from external entities, providing connectors to IoT platforms as a way to interact with SocIoTal, 3rd party components and platforms are not detailed in this document.

For instance, one IoT service may be another component of SocIoTal like a Device/Users/Data directory where data are actually stored and the user environment is an

application built on top of the API provided by such component. The assumption is that UserEnv and DevelEnv will base their operations on another platform (or set fo components), able to perform basic CRUD (Create-Read-Update-Delete) commands on concepts like User, Device or Message/Event.

Another cloud service could provide access to a social network like Facebook or Twitter. Existing social network services can be leveraged or mapped to SocIoTal requirements in order to reuse or develop functionalities on top of the social networks' API.

1.3 Methodology for Feature specification

In this section we define the main methodology (as illustrated in Figure 2) to identify and document the feature specifications for both tools: the SocIoTal User environment and the SocIoTal Developer Environment. We first describe the main use cases that highlight relevant challenges and innovative key features. Where applicable each use case is followed by a mockup user interface to better explain how the features are envisioned. Once the key challenges are identified, the section provides an analysis of related tools using comparative criteria that are related to the key features here identified. The end of the section provides a more detailed enumeration and description of features.

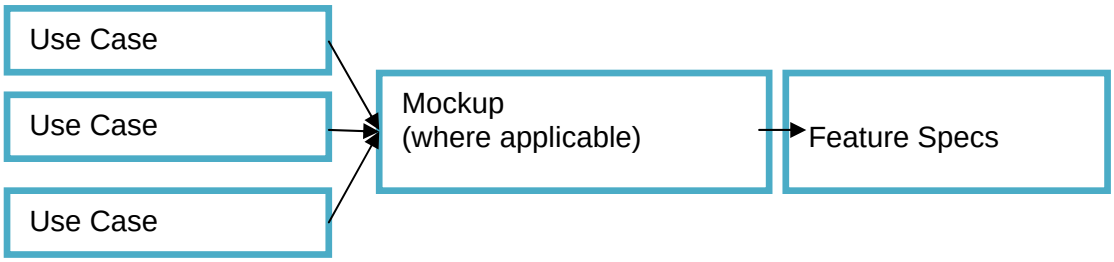

Figure 2 Simple specification process from use cases to feature listing

1.4 Privacy and reputation guidelines

Aim of the SocIoTal project is to foster and support users' devices and data sharing, thus achieving a machine-to-machine collaboration, the realization of which requires to solve various challenges in terms of privacy and security of the generated data and associated users. For this reason, one of the main research challenges in the SocIoTal project is the creation of a set of tools supporting such functionalities that will be integrated in existing Internet of Things architecture. Through well-defined APIs, the access to the same tools and their related functionalities will be made available also to the UserEnv and DevEnv designed and implemented by SocIoTal within the WP4 activities. This will empower two different communities of users and developers respectively with the tools to design and benefit of new IoT applications and services.
With this respect, some of the current WP2 and WP3 work is here reported and placed in the scope of WP4 tools, by showing how UserEnv and DevEnv could be supported in terms of privacy-preservation and reputation models. However, the provided view is still partial, and will be further enriched by the next output of WP4.
To manage a high number of users and their objects, in dynamic and changing context, an Identity Management module will be provided with the capability to manage identity in a privacy-preserving manner and being able to ensure anonymity, by assigning pseudonyms and resolving them. In order to control the access to different objects, the concept of relations among objects and users will be properly specified. Alongside with the definition of such relations, proper mechanisms to support authentication of users and objects, as well as authorization to perform or delegate actions on objects, depending on the context and other roles will be developed and exposed as tools to the different SocIoTal environments. Figure

3 shows an example of how the envisioned SocIoTal tools will allow to develop services and applications that make use of the authorization capability to gain access to information and services (i.e., car location and car's engine status services) handled by Bob's car electronic control unit from a third party user (Bob's wife). Besides, relations among objects are established, and, in consequence the associated accessibility to the required device is inferred.

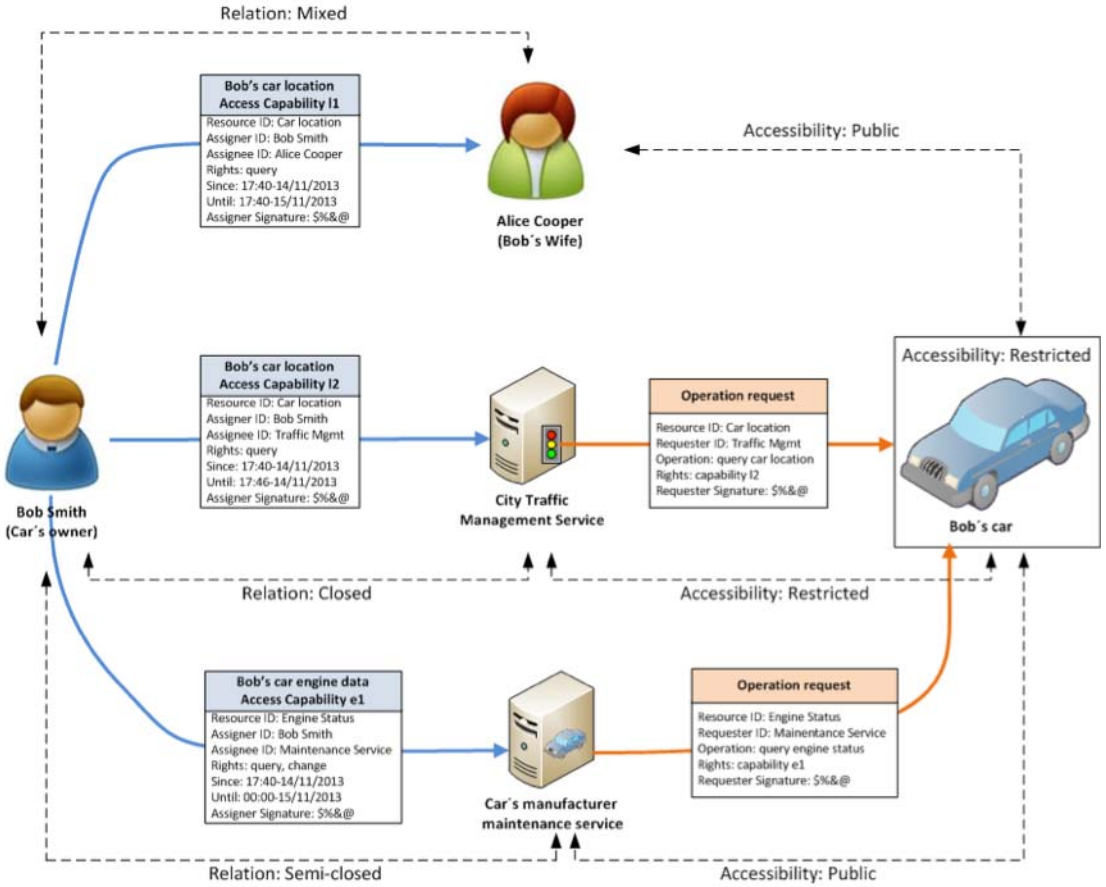

Figure 3 Capability based authorization

Among the other innovations envisioned by SocIoTal in terms of privacy and reputation, is the ability to control the access from selected *Data Consumers (DC)* (i.e., devices and services owned by other users) to the flow of information her/his devices will generate as *Data Producers (DP)*. To achieve this, the concept of *Circle (or Bubble) of Trust* has been defined alongside with a set of enablers that will allow to control creation, updates, and modification of new and existing circles of trust and associated data generation and sharing of policies. Such functionalities are particularly suited for an IoT scenario, where the very dynamic nature of the existing relations requires tools to quickly adapt the flow of information. For this reason, user data protection and control, won't reside only on static policies manually set by the user, but it should also reside on dynamic ones, able to evolve over time and to adjust to the user context, as automatically detected by the envisioned SocIoTal tools and according to the user expected benefit for a given context. For this reason, the notion of context and context-based security and privacy is of paramount importance in SocIoTal and tools for the context inference are envisioned and will be provided.
Figure 4 shows the *circle of trust* concept, exemplifying the described concepts in case where different people/users own different devices that can produce data through a set of data providers (i.e., microphones, video, location, etc.). While a given user can have full access and control of the devices belonging to him/her, SocIoTal envisions and fosters the

possibility to share devices and their capabilities (sensing and actuation) among different users, thus allowing the creation of enhanced IoT services.

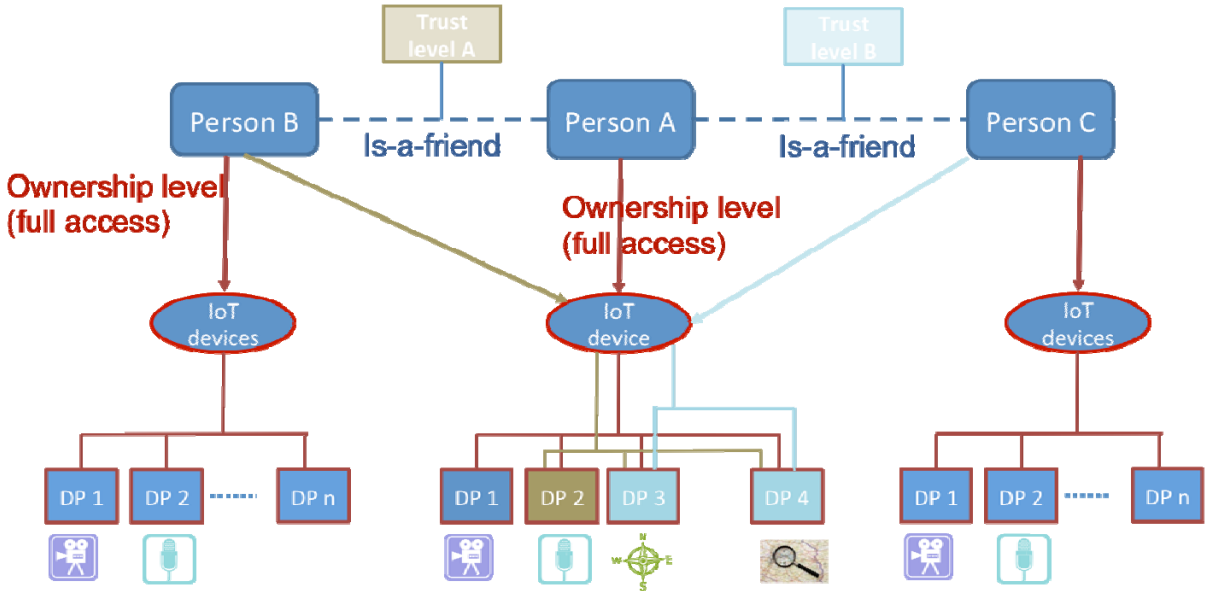

Figure 4 Circle of trust

Access to devices is assigned according to trust levels and the devices'' memberships to a given circles of trust. Users and their devices can belong to different circles and each circle allows access to a specific set of third-party devices and data producers.. Participation to a given circle is based according to relations among users and their corresponding devices, thus allowing translating additional qualifiers from devices to users and vice-versa. Examples of such qualifiers can be represented by social network relationships among users as detect by existing social networking services and thus transposed to user's devices. Additionally, qualifiers based on new enablers that SocIoTal aims to develop (WP2 and WP3) and expose to WP4 in forms of dedicated IoT tools that translate device relations to relations of users are also envisioned as one of the SocIoTal novelties. Such relationships and the participation to a particular circle of trust are seen as very dynamic and depending on the context, thus resulting in a complexity of relationships that can take place at the same time.

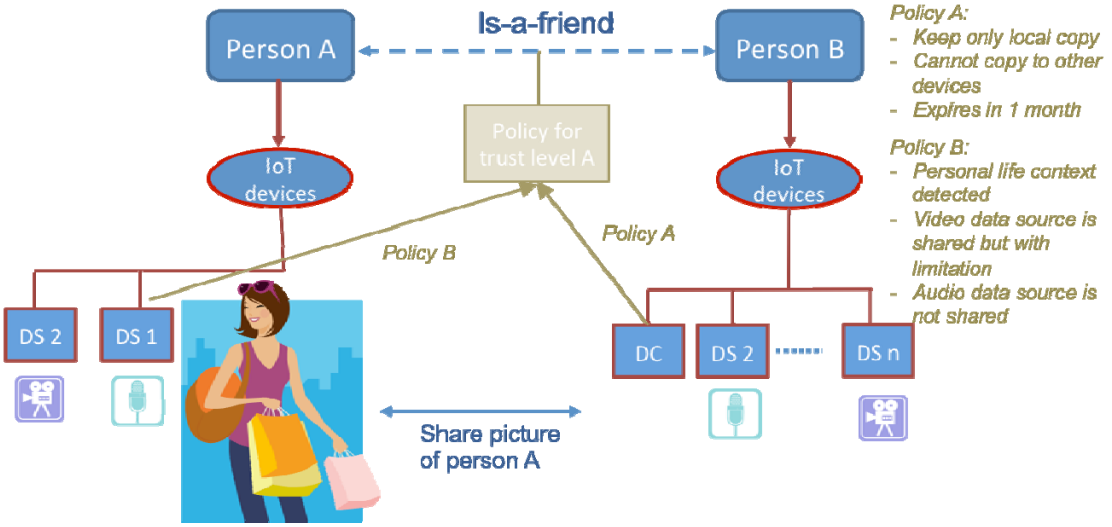

Figure 5 Data management policies

Alongside with the definition of the circle of trust, a set of data policies and policy manager need also to be defined in order to regulate not only the access and the sharing of the data but their further usage from a given data consumer. Figure 5 shows the case in which data from a given data producer are acquired from a data consumer device in the same circle of trust. However, due to the very sensitive nature of the acquired information, further management of data must follow the producer policies, taking into account the trust relationship between both. To this purpose, some of the data (e.g. audio) cannot be shared depending on the detected context.

All the required enablers, developed in WP2 and WP3, will be exposed as IoT tools/services and integrated in the User Environment and Developer Environment, developed in WP4, thus allowing the user to create and develop user-centric services to access shared resources and manage the access from other users to his/her devices.

To this purpose and in order to control this information flow, mechanisms for the creation, modification, discovery and communication within Circles of Trust and/or Communities are defined and provided by SocIoTal. Context-aware policy enforcement will then allow to dynamically adapting the policy to context, by balancing between user benefit for sharing data and privacy protection to hide some of them, according to the situation. A continuous user engagement in validating and rating effectiveness of policies application will be constantly incorporated in order to feedback and evolve the system with user preferences.
This way the UserEnv and DevEnv can be seen as front-ends that the citizen and developers can use to configure devices to adhere to such low-level functionalities by composing the required services and applying the desired policies and flow of information. To extend the SocIoTal vision aiming to protect user information by allowing the user to control the devices consuming his/her data, not only will be derived relations about the users from existing social media (i.e., social graph), but a set of new enablers will be also created and developed in order to extract user's social relationships from the daily life communications involving user devices. This way, a transposition of devices' relations to users' relations and vice-versa will be made available as a tool for enabling discovery of new circles of trust and for the modification and composition of them as well as to identify and recognize specific context situation in which different policies could be enabled, and data generated, shared and consumed.
Finally, a trust and reputation framework, to assess the reliability not only of the users/devices producing the data, but also of the users/devices consuming them, thus guaranteeing a preservation of the user privacy and a control on the shared data will be made available. This will make the user always aware of potential threats that could affect his/her data, while the system detection and notification of the occurrence of such threats will allow the user to rate and co-participate to the reputation assessment of given Data Consumers and user devices.. As mentioned above, of particular importance is to the possibility to include in the trust and reputation evaluation the degree of social interaction between involved users, as perceived by the interactions of their devices. While similar concepts well adapt to the interaction between user-mobile smartphones and user-user and smartphone-smartphone relations, however similar concepts will be applied and exploited also for identifying and understanding the stability of relations between devices and devices and the environment. All these information will be collected and elaborated exploiting devices level communication and only the result of this elaboration will be made available to the user in the form of trust/reputation scores, that annotates the resources the user can access for his/her services composition. Among the others, the availability of the following tools and modules produced by WP2 and WP3 is envisioned:

- Authentication module ensures that an identity of a subject (user or smart object) is valid, i.e., that the subject is indeed who or what it claims to be. It allows to bind an identity to a subject;

- Access Control module is responsible for making authorization decisions based on access control policies. These polices define the permissions that a subject (objects or user) has over certain target resources (e.g., IoT service). Thus, the policies specify which particular actions the subjects or groups can perform over a target resource under certain conditions. These conditions usually refer to the context information, e.g., time or location or more advanced definition of it;
- Identity Management module is responsible for managing the identities of the smart objects and users. The module is able to take into account privacy concerns to manage subjects' credentials (from users or smart objects), in a privacy-preserving way;
- Face-to-Face tool to dynamically adjust the circle of trust participation and access according to user/device relations. F2F relations will be detected for instance by user's mobile smartphones and used as qualifiers to define participation to a given circle of trust;
- Indoor Localization tool to dynamically adjust the circle of trust participation and access according to user/device indoor position. The indoor position will be detected using user mobile smartphones embedded sensing capabilities. Policies will be enabled according to the position of data producers and consumers;
- Pseudonyms tool to limit the possibility of eavesdropping information about users and to prevent the occurrence of impersonation attacks. The creation of radiolocation enablers will allow to extract context information useful to identify devices and to replace traditional identifiers;
- Reputation tool to verify the Quality of the Information shared and the trustworthiness of the devices consuming the shared information. Localization-based reputation mechanism, where user and its device location and repetition of recurrent path are used as score for user reputation, will be exposed. Additionally prediction of user and its device location will be leveraged to assign reputation score according to the possibility for the user and its device to be exposed to privacy and security attacks when acting as data producer and consumer. In order to preserve user privacy, it is envisioned that such knowledge will be extracted by designing simple and effective solution able to run and scale at user device level, without need to build a Big Data Management infrastructure.

Some early details concerning the integration of such tools/modules with the UserEnv and the DevEnv will be introduced in Section 2 - and Section 3 - while a more consolidated design of their integration will be tackled in the future tasks of WP4. Basically, it is envisioned that a more intuitive and less flexible way to access and deploy the above tools/functionalities will be supported by the UserEnv while the DevEnv will allow more advanced usage with the possibility to define personal configuration and different level of integration.

Section 2 - SocIoTal User Environment

In this section we first illustrate the main use cases that highlight relevant challenges and innovative features. Where applicable each use case is followed by a mockup user interface to better explain how the features are envisioned. Once the key challenges are identified, the section provides an analysis of related tools using comparative criteria that are related to the key features here identified. The end of the section provides a more detailed enumeration and description of features and finally an architectural view of the tool is described

2.1 Relevant bottom-up use cases

2.1.1 Use Case: Touch Compose

Bob is at Alice's home and they are both on the balcony, Bob likes the windmeter Alice recently bought. The wind meter is already in the SocIoTal network and is tagged with a visual/radio code. Bob asks Alice if he can use the windmeter data, Alice says yes and Bob put his mobile close to the windmeter and the SocIoTal app ALLOWS Bob to register the windmeter in the Bob's workspace, then the app pop ups "Do you want to compose with this windmeter", Bob touches the "yes" button and later on he will compose the logic "if wind raises schedule me a windsurfing session".

This use case points on the support to the "creative" side of the user, and provides a way to combine different elements to follow his/her purposes. Figure 6, Figure 7 and Figure 8 show UI mockups with pointing and composition features.

Figure 6 Mockup of pointing a device and acquiring device info via barcode to register the device as an entity in the UserEnv workspace

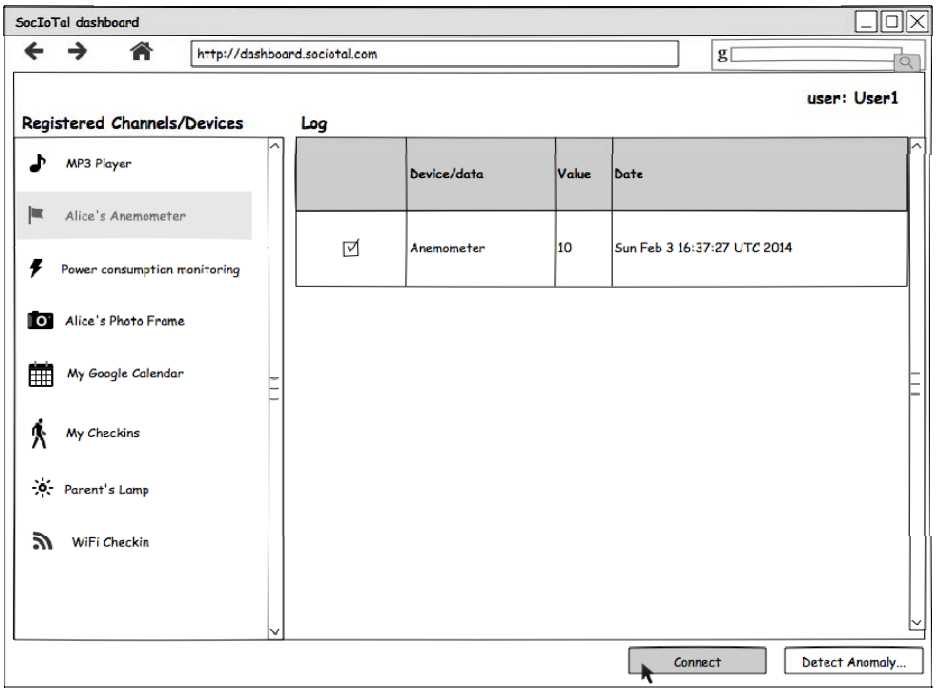

Figure 7 Mockup of a composition between an anemometer and another channel or device (part 1)

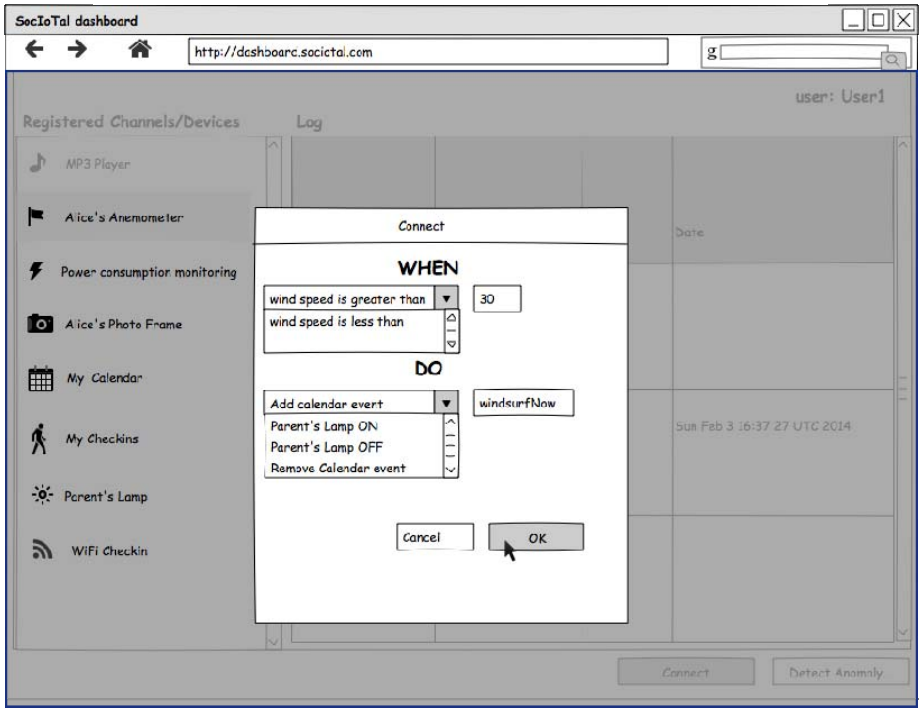

Figure 8 Mockup of a composition between an anemometer and another channel or device (part 2)

2.1.2 Use Case: Policy from Social (flexible policy)

Alice goes in the kitchen to take some drinks. Bob takes a tour in the Alice's living room and sees a nice photo-slideshow in a digital frame with a SocIoTal tag. He likes pictures and tries to acquire the frame with his mobile, but unfortunately the SocIoTal app DENIES this action showing the message "This item is private". When Alice joins Bob in the living room, Bob tells about his attempt to acquire the frame. Alice suggests re-trying, and now, given that Alice and Bob are in the same room, the SocIoTal app ALLOWS the action. Few days after, Bob is in Alice house and notice a new digital frame. He tries to acquire the picture while Alice is not home. The SocIoTal app ALLOWS the action because the system already recognized Alice and Bob to be part of the same social circle of trust.

This use-case focus on innovative ways to allow access and enforcing policies according to context and social relationship. It is related to the property of "trustworthiness".

2.1.3 Use Case: Compose-by-example

When Bob is back home, he connects to his web SocIoTal dashboard and sees the newly registered objects (For example: a windmeter, a digital frame).
He rethinks to the failure of his refrigerator during the last week when he was abroad, and from power consumption data he realizes that an electrical problem in the house was correlated with higher consumption. Then he selects the two feeds in the data screen: power consumption and his own presence (checkin/checkout) and the system assist him to build anomaly detection and Bob associates to future anomalies an alert message for the mobile phone. The composition is performed without coding but in an intuitive way using data selections and form-filling. Additionally, using the same SocIoTal dashboard

This use case focus on the ability of the tool to be "assistive". The Figure 9 shows some mockup user interfaces to depict such features.

2.1.4 Use Case: Anonymous access to prevent tracking

Bob wants to implement a trigger to remotely turn on an IoT lamp shared in his object space. The lamp is deployed at his parents' house and it should automatically turn on when Bob is at home, to automatically provide a feedback to them. This can be done by simply composing the following rule: If Bob smartphone is detected by Bob WiFi access point, then the lamp should be turned on. To avoid to be tracked using WiFi AP connectivity eavesdrop, Bob smartphone should anonymously connect to access points when outside of his home

This use case points on the "trustworthiness" of the system.

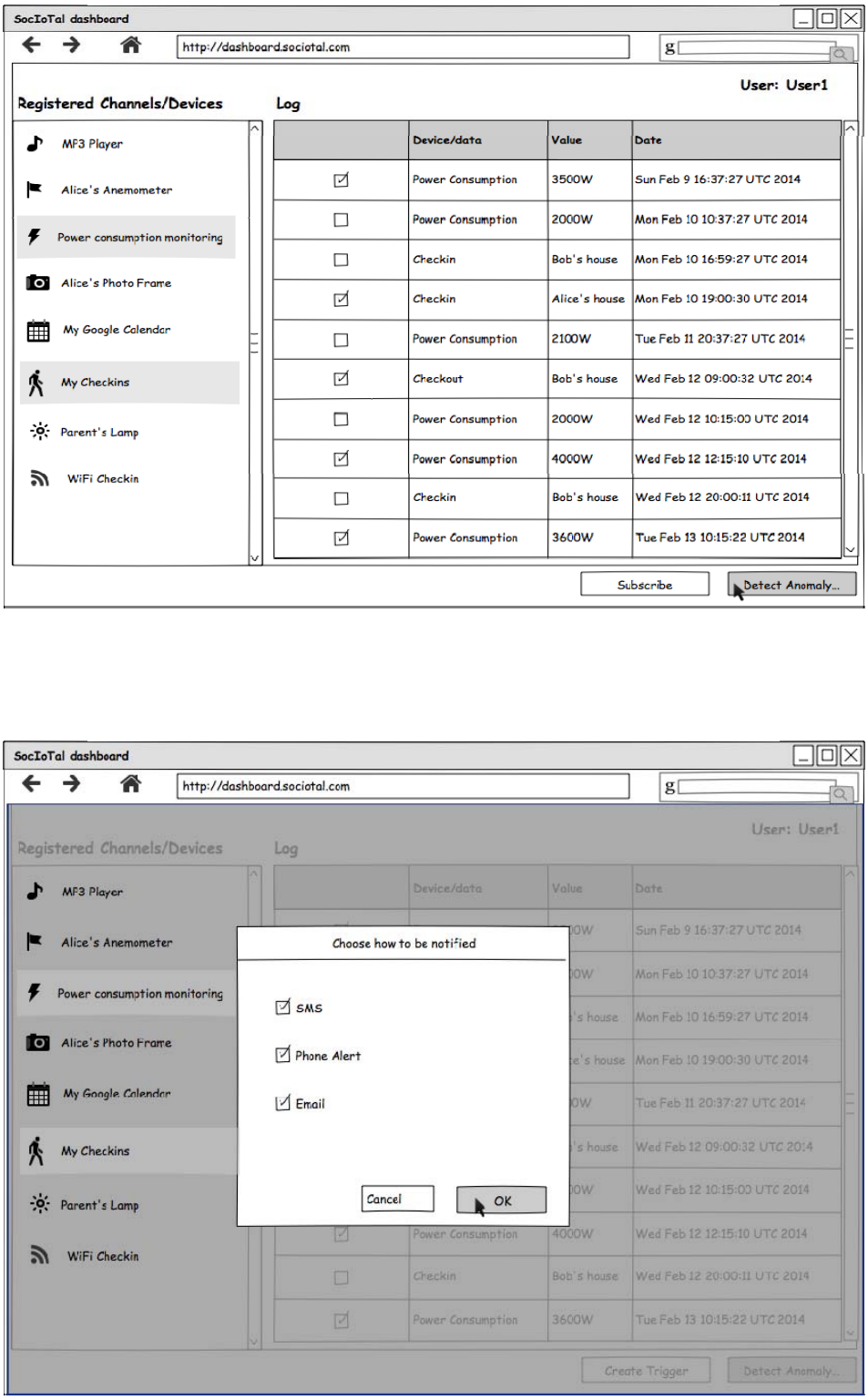

Figure 9 Mock-ups of selecting data and configuring an anomaly detection trigger. Mockups are just illustrative; the real user interface may substantially differ from the figure.

2.2 State-of-the-art of End-Users Tools in IoT

The SocIoTal citizen empowerment tools focus on the interaction between users and their devices in a privacy-aware IoT ecosystem. The UserEnv could be classified in the emerging field of End-User Development tools for which Lieberman et al. propose the following definition [2]:

"End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."

There is a debate in the scientific community about end users tools pros and cons. Main concerns are about security in a broad sense, given the scarce ability of end user to understand the security implications of their actions [3]. While others assumes that end users are not all equally naive when it comes to understanding software and that sophisticated end users are able of proficient and secure development with end-user tools[4].

SocIoTal will investigate solutions that in intuitive way allow users to compose data, devices, and services to build simple, but personalized applications. The concept of Touch-&-Compose [5][6] will be further evaluated and extended, integrating the discovery and pointing from the physical environment techniques together with a web dashboard in which those object will become components for future use and reuse.

2.2.1 Comparison Criteria

Differently from D1.1, where a whole comparison of IoT platforms [7] is presented across all possible dimensions, here we need to focus mainly on the dimensions that emerge from the citizen empowerment use cases where end users need to interact with IoT. These dimensions can be summarized in three main adjectives: creative, assistive, and trustworthy. To analyse the impact in these three dimensions we seek the following classes of features in a number of tools already available.

- Data views: the feature to visually manage data from his/her devices and from devices of paired users
- Target: is the tool aimed at end users? Which technical skills would be required from the final user? Assisted composition: the ability to assist the user to activate some actions depending on events generated by devices
- Point-discovery: the ability to discovery devices when in proximity (with barcodes, radio tag, etc.)
- Contextual-policy: having a flexible device policy access that takes count of context (eg. physical proximity, social relationships, …)
- Device classes: type and number of "things" and devices directly supported

As SocIoTal innovation is aimed at Citizens and Communities, it is also important to evaluate the ability of SocIoTal tools to be managed without the constraint of a business-to-consumer model, but rather a community-to-citizen one. Defining a tool as a piece of software delivered in form of application for one or more platforms, we can order the different levels of openness according to the following considerations:

Closed: the tool can be used only as-a-service or completely controlled by the provider, it is

not possible to plug new elements and components from the bottom. The set of features is only determined by the provider (e.g. the software author or the web company)

Partially interoperable: the tool is controlled by the provider but can interoperate with external apps providing a common format for data sharing.

Fully interoperable: the tool is able to provide a complete API to manage all relevant resources. Some functionality can be added bottom-up.

Open: the tool can be fully controlled by final users. More features can be added bottom-up.

In this respect characteristics like:

- **License**: which kind of license is available (open, EULA, other)
- **API**: It is possible to connect to the tool using an API?
- **Interoperability**: if API is not available, is the tool able to export data using a standard format?
- **Deployment**: what is the tool deployment:
    - as a service: deployed in a proprietary server and accessible as a service;
    - hosted: deployable in a user controlled server and accessible as a service;
    - standalone: could be installed in a user device or PC.

are worth to be evaluated because of their impact on lowering the barriers towards a pervasive adoption of IoT among citizens and communities, which is the main articulated objective of SocIoTal project.

2.2.2 Analysis of tools

2.2.2.1 IFTTT

IFTTT (IF This Than That)[8] is a commercial service funded with more than 8M$ only in 2012 and based on a B2C service model. Initially conceived for social networks and online services, now enables physical mashups because it is able to connect a few physical objects like the USB blink (a usb micro lamp), the NetAtmo sensor suite and Belkin WeMo (a smart electrical plugs). IFTTT has a strong and appealing wizard to compose a trigger/action recipe, it seems aimed at the above-average web user and it results quite simple to use and activate. It does not enable any particular feature to discover devices and people in proximity and the mobile app replicates more or less the desktop features. It is not possible to share a device on IFTTT, but it is possible to share recipes. This last feature it is very interesting because opens to an even more simplified way to compose trigger and actions. Another feature is the number of channels actually available in the platform which now counts about 80 among online services, device types, and social networks.
The system is provided as-a-service and the license model is EULA free of charge with data updates every 15 minutes. It does not provide any kind of API.

Pros: appealing user interfaces, easiness of use, reusable/shared recipes, large set of channels, mobile app
Cons: not aimed at sharing data and devices, proprietary and distributed as-service only, 15 minutes between trigger refresh, no API so independent apps are not possible unless setting up a bilateral agreement with IFTTT.

2.2.2.2 Paraimpu

Paraimpu [9][10] is a tool running as-a-service born at CRS4 and recently spun-off as SME. It implements the concept of user-generated physical/virtual mashups. It provides a workspace where users can register devices (among devices, Arduino boards are supported) providing a basic level of device virtualization. Objects in the workspace are classified as data producers and data consumers and such roles can be played by devices or by social sites like Twitter and Facebook. A transformation engine allows composing producers to consumers. The logic of a composition is ruled by (match, replace) paradigm where every data coming from a source, if matches one expression, is adapted to be delivered to a consumer. Paraimpu allows objects in the workspace to be shared by a mechanism of bookmarks and policies. The workspace is aimed at end-user development and it targets above-average Internet users.

Pros: Good balance between simplicity of use and flexibility, social-ability and sharing of sensors/actuators; good support of Arduino.

Cons: Closed-source and distributed as-a-service, basic level of API.

2.2.2.3 Xively

Xively [11] is a commercial (closed) service owned by LogMeIn company, formerly known as Pachube, first and Cosm, lately it is among the first IoT platform available online (if not the first). It is mainly focused to a B2B model.

Xively cloud-based services allow deploying and managing batches of products in real time. The user workbench is oriented to skilled users: it provides a set of facilities to collect data from devices through a good set of API and wrapper libraries supporting programing languages and devices like Java, JavaScript, Arduino, Android, Objective-C, etc. Very interesting is the possibility to monitoring in real time data coming from devices through configurable graphs and charts.

User workspace allows defining simple conditional triggers on connected devices data in order to automatically POST data to external services endpoints. Xively allows searching for devices already deployed in the Xively system, but not by proximity and it doesn't provide an advanced policy access: a device can be private or public. Not social features at all.

Pros: mature platform supporting REST, sockets and MQTT protocol, good set of API and libraries targeted to most used programming languages and devices. It's a B2B professional service.

Cons: proprietary and distributed as a service only, no devices inter-connection/composition facilities, workspace focused to technical, skilled users. It's not social. It's a B2B professional service, not consumer oriented.

2.2.2.4 Node-RED

Node-RED [12] is a visual, workflow-oriented, browser-based tool for the Internet of Things. Written in node.js (JavaScript), it allows to wire together services/devices and to deploy them on the related server backend. Created by IBM, is released under Apache Licence 2.0. Node-RED provides a visual workspace through which it's possible to create a workflow between nodes; nodes are picked up from a palette, which categorizes them into: inputs (like MQTT, sockets, HTTP), outputs (like serial, TCP, Multicast); and functions (like http request, switch, range), social (for social networks communication) and storage (file, database). Node-RED comes with a predefined set of basic nodes, but community is developing a growing set of additional nodes, available to install and extend the platform. Users can also write new functions and nodes. While it's quite simple for a skilled user to create flows and services and deploy them in minutes, it doesn't seems so easy-to-use for the average Web user due to workflow paradigm, nodes configuration and deploying system. Node-RED is a general-purpose environment so it doesn't provide device discovering by proximity or other advanced functionalities but it is high-extendible and embeddable in other applications and could be a valid choice for an advanced user environment or for services developers. Also, it doesn't support device pairing between users using predefined nodes.

Pros: nice visual, workflow system for the IoT. It's easy to compose and deploy workflows including Internet services and devices. It is extensible, so it is possible to add new elements and supported nodes in workflows and applications. It is open-source. Node.js based, asynchronous, event-based framework.
Cons: Maybe not so suitable for the end-user; some configuration details are not easy to understand for not skilled people.

2.2.2.5 Webinos

Webinos [13][14] is an Open Source platform that includes a set of software components for the IoT in the form of web runtime extensions. Webinos Architecture is centred on the concept of a "Personal Zone" as a mean to organise personal devices and services. Each device, whether it be a mobile, tablet, desktop, smart TV or in-car unit device, is extended to enable the device to be a part of the Personal Zone. Services are the Webinos way of exposing APIs.
Webinos provides a personal zone hub dashboard where users can manage their account or service configuration, remove connected devices and connect to other personal zones.
Also each connected device has access to a personal zone proxy "dashboard" where user can do a local discovery, edit PZP configuration and enrol, connect and share service between local/remote devices. There is also a set of links to get access to all the installed Webinos API TestPages.

Webinos architecture is extensible and exposes APIs for developers that can build their custom apps with a simple end-user environment. These Webinos apps can have a custom dashboard (https://developer.webinos.org/application-gateways) and can take advantage of the capabilities of the Webinos platform to virtualize real objects, such as sensors and actuators, as services. This means that every real object connected to a Webinos-enabled entity throws directly into the Webinos ecosystem its data stream. This Webinos application is able to interact with real objects using Webinos APIs within Webinos framework.
These apps are integrated with the Webinos dashboard. It is a common interface for managing devices and services that belongs to the user's personal zone or to a friend's user (in respect of privacy/policy rules).

Pros: extensible architecture for IoT developers that can build applications by offering a single virtual device that can consist of all devices owned by a user. It is open-source. Node.js based.
Cons: Not so suitable for the end-user directly. Developers must create applications with a final-user more friendly environment

2.2.2.6 Nimbits

Nimbits [15] is an Open Source Internet of Things platform running on a Distributed Cloud. It provides a collection of software components designed to log time series data from sensors. As that data is logged, events can be triggered.
The Nimbits ecosystem consists on users around the world who have downloaded and installed an instance of Nimbits Server on their cloud. These instances around the world, all providing highly redundant and scalable data logging, are able to share data with each other, and can be made searchable so people can find data feeds and connect to them.
The Nimbits Cloud platform is an installable server. It is a Process Data historian which means it's optimised to store data that comes in as time-stamped streams of values. It makes it easy and efficient to store high frequency changes.

Pros: Open Source. Provides REST web services for logging and retrieving time and geo stamped data. Runs on Google App Engine and can run in a Raspberry PI device.

Cons: Not so suitable for the end-user directly. Hard to understand how the platform could be extended.

2.2.3 Summary of comparison

The tables below summarize the given criteria applicable to the above tools and systems. According to the table, none of the tools analysed here provides all the features that descend from the objectives of WP4 and identified as key challenges. None of the tools analysed is at the same time creative, intuitive, assistive and trustworthy to the extent required by SocIoTal use cases. Tools like IFTTT, Paraimpu and Xively seems to privilege the easiness and are simple enough to be used without much effort. On the other hand they are cloud based and no new features can be added by the community. At the other extreme open sourced tools are more tricky to install and deploy, and even when deployed the user interface is not that creative and assistive. Nevertheless, as their code can be easily inspected and modified to fit with privacy preserving models and requirements of a Community.

Tool	Data views	Target	Assisted composition	Point discovery	Contextual policy	Device classes
IFTTT	poor	Above average	Rich	Absent	Absent	80 (including not device types)
Paraimpu	poor	Above average	Rich	Absent	Poor: people may establish social relationships inside the tool	16 classes including online services

Node-red	poor	Developers	More or less, it's a Workflow system	None	None	About 25. Simple to add new classes
Xively	Good. Graphs	Developers	Minimal. Triggers are supported, limited to HTTP POST to other web services	Absent	Absent	No classes, libraries are provided for Arduino, Android, Electric Imp. and well-know programming languages
Webinos	Rich	Developers	None	Yes	--	--
Nimbits	Rich	Developers	None	None	--	--

Table 1 Summary of key features for all the tools analysed in the domain of the SocIoTal End User Environment

Tool	License	API	Interoperability	Deployment
IFTTT	EULA	Absent	Absent	As-a-service
Paraimpu	Proprietary	Poor	Poor	As-a-service
Xively	Proprietary	Yes.	-	As-a-Service
Node-red	Apache 2.0	Only to create new things classes	-	Hosted, standalone
Webinos	Open Source	Yes. Rich	-	hosted
Nimbits	Open Source	Rich	-	Hosted

Table 2 Summary of criteria related to openness, interoperability and integration for tools analysed in the domain of SocIoTal End User Environment

2.2.4 Advancements and Success Indicators

UserEnv is not an enabler of a new class of features per se, but brings together features coming from diverse experiences. The result will be the design and the implementation of tools that in an intuitive way will allow users to compose data, devices, and services to build simple but personalized applications. To summarize, the UserEnv will move towards the achievements of three qualities or adjectives: creative, assistive and trustworthy.

As the UserEnv is dedicated to average internet users, it should be usable without being trained in computer programming or in other complex business graphical notations like BPMN or equivalent. Users are just expected to understand the metaphor of linking a condition to an action. This aspect is not new in absolute, i.e. IFTTT and Paraimpu already present this concept, but in SocIoTal we are experimenting with a tool which provides a higher level of openness and trustworthiness.

Success indicator: a success indicator is to prove the accessibility and gradual learning curve of the UserEnv against the proposed use cases. The methodology to assess this success indicator will be likely based on user testing in controlled sessions with survey and interviews.
Such evaluation may follow some of criteria available in literature for End-user development tools presented by Sutcliffe [16]. As a certain level of is always required to learn a new tool, the users' motivation depends on their confidence that it will empower their work, save time on the job and/or raise productivity. This study defines the costs as the sum of:

- Technical cost: the price of the technology and the effort to install it
- Learning cost: the time taken to understand the technology
- Development cost: the effort to develop applications using the technology
- Test and debugging cost: the time taken to verify the system.

The details of the methodology and any eventual benchmark, where applicable, will be presented in the second iteration of task 4.1 together with a first version of the UserEnv prototype.

2.3 List of SocIoTal User Environment Features

The SocIoTal User Environment (UserEnv) will be likely composed by two applications:

- A mobile app, we can call the SocIoTal Mobile UserEnv
- A web based app, the SocIoTal Web UserEnv

Both, the web app and the mobile app will need to communicate to a backend service (or set of services) which at the moment can be envisioned as a black box with a set of APIs that can be invoked remotely[1].

Authentication
Scope: mobile and web When the user opens the app, he/she is required to authenticate to access all the configurations and information he saved before in the UserEnv. The UserEnv is not expected to provide authentication services but to access to authentication using some authentication protocol (eg. Oauth2) Pre-requisite[2]: An authentication provider (eg. Oauth2 [17] or alternative) is in place to provide authentication services.
Registering a new virtual entity in the workspace
Scope: web A physical device is not directly available in the UserEnv unless it has been previously connected with a device gateway. After this operation, it is expected to have some address of the Virtual entity representing the physical device and such address can be used to register such entity in the workspace Pre-requisite: A device gateway able to virtualize physical entities into virtual entities is deployed and its address is known to the UserEnv
Enforcing a policy
Scope: mobile and web When a user tries to acquire/connect to a device/feed of another user the web app/mobile app asks the backend if the policy allows this action. If the authorization fails no actions are possible and the object is not acquired. Different privacy policies will be defined and made available, according also to the different attributes to be exposed. Pre-requisites: Users and devices info are stored and managed by an external entity and the policy or access/control/lists are available upon request.
Flexible policy
Scope: mobile/web Policies will be adapted to the context. The UserEnv will provide functionalities to link policy to context. As example of context, people connected by a face-to-face relation or part of the same circle of trust or in the same location, may have a more permissive policy able to

[1] The first draft specification of SocIoTal API will be available after M12.
[2] A pre-requisite highlights a dependency to another component external to the UserEnv, i.e. a library, a remote service or another SOCIOTAL enabler.

adapt to this context.

Pre-requisites:
The activity in WP3 will produce the required enablers to compute such context, exposed as *Face-2-Face IoT service* and Indoor localization service, to be used along with a Context Manager and Policy Manager provided as architecture tool by the SocIoTal project.

Secure communications
Scope: mobile This feature is to avoid leakage of user information and to prevent impersonation attacks Pre-requisites: The activity in WP3 will produce the required radiolocation enablers, exposed to the UserEnv in the form of *Pseudonyms IoT service*. The service will allow generating pseudonyms to be used in communication between devices, by making use of specific devices characteristics and their position and relation with surrounding environment and other devices. The use of the service will make very unlikely the possibility for an attacker to impersonate a trustworthy device. The notion of context will be enriched in a way that makes for its attacker always more difficult to mimic it. Users' and devices' identities will be protected by making use of pseudonyms.
Data Producer and Consumer Reputation
Scope: mobile/web Pre-requisites: The activity in WP2 will develop the required reputation mechanism, exposed to the UserEnv in the form of *Reputation IoT service*. The service will allow to verify the Quality of Information produced and shared by a Data Producer as well as verifying and measuring the trustworthiness of the Data Consumer, consuming the shared data. Flexible metric computation will be defined in order to adapt to the given context and by taking into account and rating information produced also by other existing services and enablers.
Discovery of devices using a mobile as pointer
Scope: mobile The user can put his mobile close to a physical device (eg. a Windmeter) and the SocIoTal mobile app registers the newly discovered thing in the user workspace, given the security policy authorizes this operation. Pre-requisites: The mobile phone is able to acquire a device_id from the barcode/rftag Device/Sensor are physical tagged with a radio/visual tag Device/Sensor info (like type, name, owner, id, …) are stored and available upon request
Simple Composition
Scope: web The UserEnv will assist the user providing the ability to create a trigger related to a condition on a feed (eg. Wind > 20 kmh) and to associate the trigger to an action already available on the workspace (eg. Add an event to a calendar) Pre-requisites: Devices, feeds and actions should be available in the UserEnv as API connectors
Data/event visualization and selection

Scope: web The user can visualize data/event and logs from the feeds/devices and actions registered in his workspace. Pre-requisite: Data should be available to be retrieved by UserEnv via remote API or other equivalent technique.
Trigger on anomaly
Scope: web The user can select feeds in the data visualization and build an anomaly trigger. The anomaly is based on statistical tools (eg. If a value is over or under certain quantile) the trigger generates a new anomaly event. If not specified, the anomaly can be automatically composed by the system with a push notification action or other alerting action. For such reason, this feature is also relevant for “assisted or semi-automatic composition feature” described above. Pre-requisites: Feed must provide numerical data (eg. a sensor of power consumption) A push notification or another alert action is already registered in the workspace (for automatic composition with an alarm)
Receive a push notification
Scope: mobile Here the mobile app can receive alerts or messages from other devices/services in the personal IoT. Examples, in case of anomaly in energy consumption the user receive a push notification. Pre-requisites: An IoT service to provide subscription to a device and/or service (producing notifications) and an IoT Broker to perform publishing (push) data (notification) (Publish/Subscribe enabler)
Create/Register to a community/circle of trust
Scope: mobile/web The mobile and the web apps should be able to create a circle of trust and/or community, including all the currently available desired devices and/or read the available existing communities and perform a request to be included in selected ones. Once the user has been accepted, the user (through the app) should be able to share, from his/her owned devices/resources, those which wants to share within the selected communities. Pre-requisites: A set of IoT Services that provide community discovering, and management, allowing add and remove devices/resources and access to their data.

Table 3 List of features of the User Environment

2.4 Architecture of the SocIoTal User Environment tool

The main components of the user environment architecture are a user web and mobile application as well as an API-to-API broker. User environment is built on top of the general SocIoTal architecture by leveraging envisioned SocIoTal API that provides predefined set of generic operations enabled by the API-to-API broker, which also enables communication with third party applications.

2.4.1 Overview

The main sub-components of the User Environment are described in the following section.

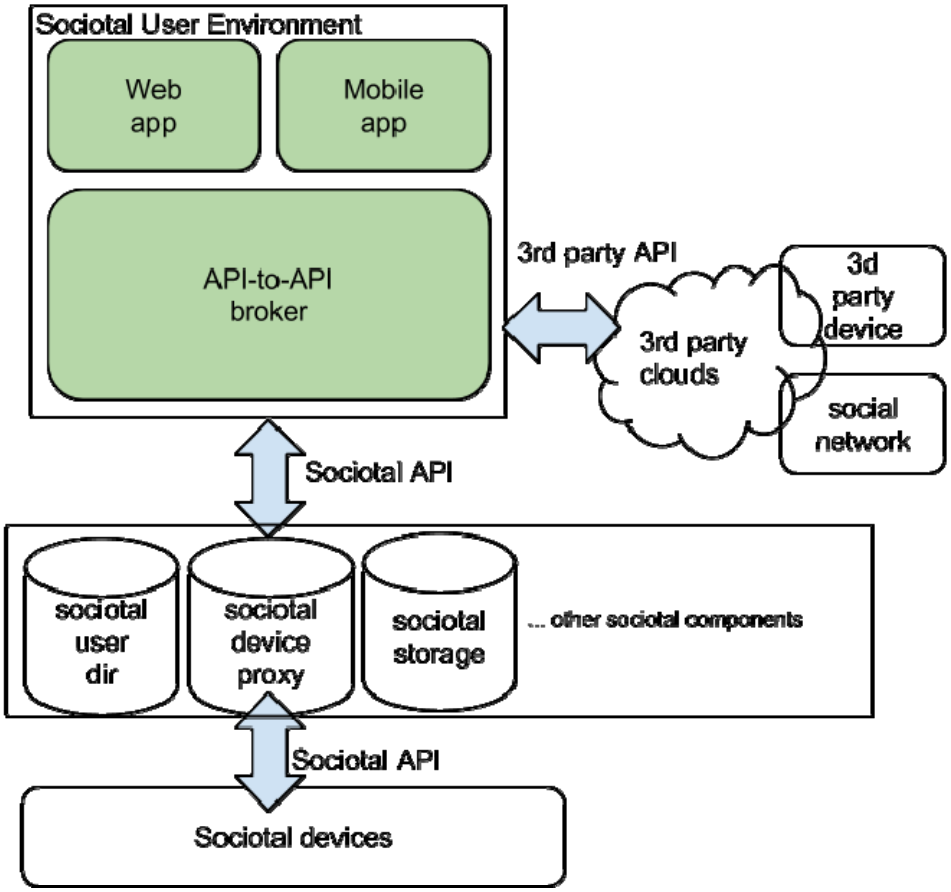

Figure 10 High level view of components and subcomponents of UserEnv in the context of SocIoTal and 3rd party systems

Components and subcomponents

SocIoTal UserEnv Web app: will be a component of the **UserEnv**, able to communicate with the API-to-API Broker and with other SocIoTal Components and third party cloud services. This component is expected to provide the following functionalities:
Authentication module client against SocIoTal User Directory or equivalent authentication service.Connector to Social Networks,

- Client to subscribe a trigger/action on the Event Mapper
- Data views

SocIoTal User Environment Mobile app: will be a component of the User Environment able to communicate with the Message Broker and with other SocIoTal Components and third party clouds. This component is expected to provide the following functionalities:

- Acquisition of device profile via barcode (or to NFC)
- Views to manage operations on the mobile
- Push collector: a module that receives Push notification from API-to-API broker.
- Authentication client to **SocIoTal User Directory Service** or equivalent authentication service

SocIoTal User Environment API-to-API Broker: will be a server-side component able to route messages from an endpoint to another (eg. From a REST API of a SocIoTal device to a REST API from a third party cloud). This component is expected to provide the following functionalities:

- **API Connector:** API connector is a module that allows interfacing the UserEnv to a SocIoTal API (eg. A SocIoTal Device gateway) or to a 3rd party device API (eg. Xively), or a social/online service (eg. Facebook). Each category of device or services to be managed in the UserEnv will have a corresponding API Connector.
- **Event Mapper**: it is a submodule that collects events from event sources and looks up actions that must be invoked according to trigger/action subscriptions

As described in Section 1 the components that are **not part** of the UserEnv, but on which the UserEnv may depend are (thus to be identified as pre-requisites):

- **SocIoTal AAA (Authentication, Authorization and Accounting) -** This service should provide the credentials for the UserEnv to operate with other services and devices.
- **SocIoTal Device Gateway/Proxy -** virtualizes real devices and adapts their low level protocols to HTTP stack. It should enable the UserEnv to access virtual entities that represents physical devices and physical entities
- **SocIoTal Data manager and storage** – data and message platform storage, if applicable. The UserEnv is not expected to deal with storage of end-user data collected from sensors and devices. However some of such data will be stored and provided as part of the SocIoTal architecture.
- **SocIoTal Context Manager**: defines the detected context that will regulate the participation to circle of trust. It hides details of F2F IoT service and Indoor Localization IoT service to dynamically adjust circle of trust participation and access according to user/device indoor position and detected social relations;
- **SocIoTal Trust/Reputation/Pseudonyms** to limit the possibility to eavesdrop information about users and to prevent the occurrence of impersonation attacks and to verify the Quality of the Information shared and the trustworthiness of the devices consuming the shared information;
- **SocIoTal Publish/Subscribe module** to perform the automatic notifications pushing to those registered applications

Other third party services (eg. Xively, Smart-Citizen.me [18]) may represent loose dependencies. The components and enablers assumed as pre-requisites will be defined in the SocIoTal architecture (T1.2) while the interface among the internal and external components to the User Environment will be defined by the SocIoTal Open API (T1.3)

2.4.2 Domain Model for SocIoTal UserEnv

Here are described the main concepts/entities to model the UserEnv domain. In bold are terms that may directly refer to such entities.

User

A **user** should authenticate before accessing the **UserEnv**. The authentication may be performed against another SocIoTal service, but a User entity could eventually be created in the UserEnv backend.

User may be part of different circles of trust and the participation to one or the other circles can be dynamic and adapted to the user context (see Section 1.4).

In a first implementation we may use a social id as authentication server As soon Sociotal defines an alternative authentication service thisthese dependencies could be removed.

API Plug

An **API Plug** is in the **UserEnv** a bundle element which handles the communication abstraction to a **Device** or to a **Service.** In this respect the **Plug** can be seen as univocally related to a **Virtual Entity** in IOT-A terminology. The name plug suggests it is a self-contained bunch of functionalities that can be easily isolated and decoupled for the rest of the architecture. It is mainly composed by a **Task (**request/response or subscribe/notify**)** with an **API consumer** that may require some sort of **User** authentication when instantiated. It is likely for the **User**, to have in the **Workspace (**a subcomponent of the **UserEnv)** specialized **Plugs** for each category of **Device**/**Service**. For instance a **XivelyPlug** is a subclass of **Plug** for devices connected to Xively while a **NetatmoPlug** for Netatmo devices. SocIoTal may define one or more SocIoTal devices and an API for those devices with one or more corresponding SocIoTalPlug(s). In the early prototypes and design, we will often refer to some existing **device virtualization services (or Device Gateways)**, to let emerge this as a requirement for the core SocIoTal architecture. **Social accounts** and online services concepts may be captured by **Plugs** generalization. Each API **Plug** class is a modular component which bundle also one or more **Trigger** classes and one or more **Actions** classes for that Plug. Each **Plug** may have one or more specific **Configuration** attributes. A **Plug** starts also a **Task** and may be parameterized with some scheduling information (eg. get data every 30 minutes) for those Plugs related to device/services that does not autonomously push data (see Figure 11). Thus a **Scheduler** is needed to manage the executions of such tasks. A **Plug Task** can be put in execution (checked) also in an interactive way (eg. following **User** click on the UI), or by an external event (eg. an unsolicited push coming from an external entity). When scheduled, the Plug task connects to the device/sensor API, retrieves the current value and propagates to all its instantiated triggers.

Once a **Plug** is instantiated in the Workspace, it becomes a feed of **Events** and such events are collected in a **Inbox** with a limited storage size (the **UserEnv** will not provide a storage service).

Trigger

A **Trigger** is a class of conditions predefined for a Plug (the container Plug). A trigger can be parameterized at run time with a threshold or other parameter. **Events** are checked against the triggers instantiated for a Plug. The broker may route events from a trigger to another endpoint registered in the broker (eg. an action). Here follows an example:

*A plug for a windmeter sensor may expose a **highPassTrigger** with a check equal to 20 (eg. windspeed in kmh). It propagates an **event** only **If** the current value is greater than 20. The following operations may take place:*

- *the container Plug generates a new data **event**,*
- *if the trigger condition is fulfilled the trigger propagate the event to the **Event Broker**, if one or more actions are registered, such actions are activated.*

Action

An **Action** is a class of actions predefined for a **Plug** and eventually can be instantiated with one or more parameters. Like triggers, actions belong to some parent Plug, have a configuration, and are executable. Actions are not scheduled for execution, but they are executed immediately when the Event mapper receives an event checked by a trigger the action is listening to. Here follows an example:

*A **CalendarPlug** may export an action **CalendarAddAction** that needs only a title as parameter (eg. windsurf today at 10pm, date and time are parsed by the calendar service)*

Mapping

The **Event mapper** is a process or thread able to receive an **event** from a **trigger** and to route such event to an **action**, implementing always the generic **logic** "***if event then action"***. To operate, it may need to map a data event to an **action** call with its parameters. For instance, in the case of routing a wind speed event to a calendar schedule the transformation will be described in Table 4.

Event	Action
Trigger: highPassTrigger, check: 22.5	action: CalendarAddAction, params: ["windsurf now"]

Table 4 Example of Event to Action mapping

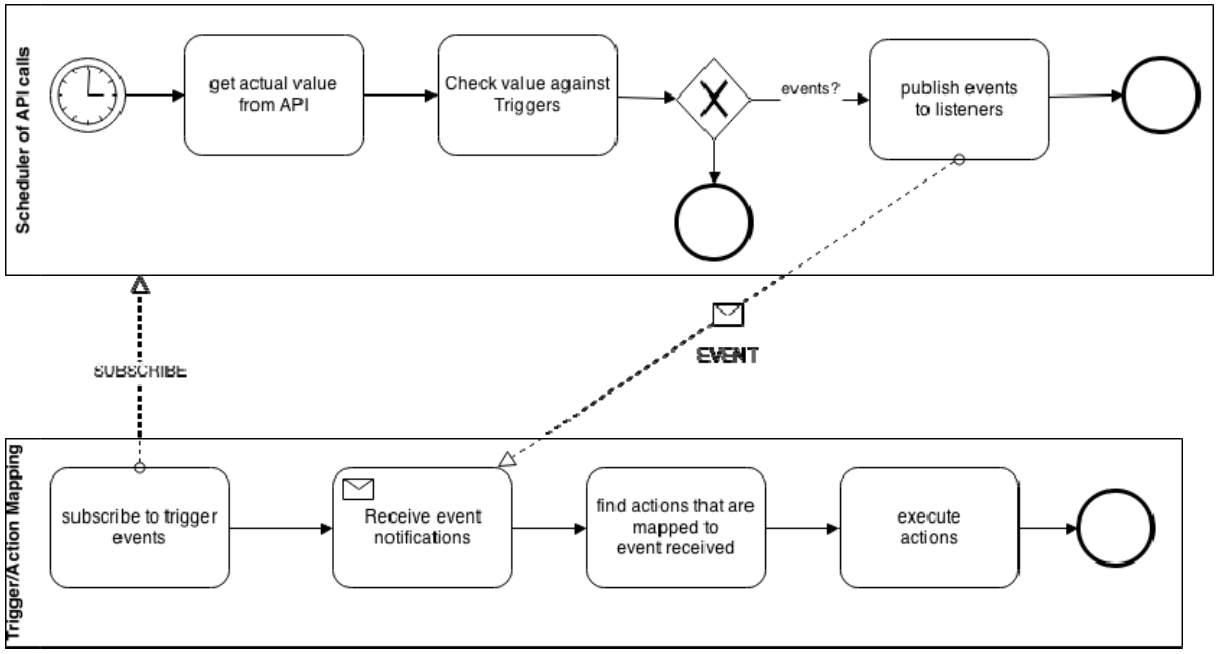

Figure 11 Business modeling when external sources of data are queried by a scheduled and periodic task.

Section 3 - SocIoTal Developer Environment

In this section we first illustrate the main use cases that highlight relevant service development features. Where applicable, each use case is followed by a mockup user interface to better explain how the features are envisioned. Once the key challenges are identified, the section provides an analysis of related tools using comparative criteria that are related to the key features here identified. The use cases presented in this section are about the general idea of the Developer Tool in SocIoTal. The integration with the core SocIoTal innovations (trust model, bubbles concept, etc.) will refine the design of the tools as soon as such concept are defined in work packages WP2/3 and consolidated in the project. In the use cases presented in Section 2 - , related to the User Environment, already emerge some requirements related to privacy policies that are, at some extent, also applicable to the Developer Environment even if the way to interact with the two tools will be different.

3.1 Relevant bottom-up use cases

3.1.1 Use Case: DIY smart home application development

Bob is a technophile person; he likes developing his own applications by using the devices he has at home. He has just bought new presence detectors for his house and deployed those in its different rooms. He turns them on and they are automatically discovered by the SocIoTal system. Bob launches the ***SocIoTal development environment*** *and starts to create his own application using these presence detectors, such as: if you detect someone at this room and if the luminosity level is below a certain threshold you turn on the light of that room with different dim values according to the time of the day. The development tool proposes many features that facilitate the application development such as repository of available devices and their capabilities, auto-completion while writing the program in a dedicated language, detection of conflicts with other applications, detection of deadlocks, etc.*

3.1.2 Use Case: City automation

Bob is a software developer working in the city municipality. The municipality requested him to optimize the energy consumption in the city by using all available devices in the city infrastructure. He uses SocIoTal tool to first discover all available devices in the city and identify the ones that he can use for this purpose (thanks to the descriptions of the active devices and services). He decides to use presence detection services that can give the presence information as well as the number of present people in the streets. By having this information he can then regulate the luminosity level of the street lamps (e.g., no light if there is no one) by using the regulation service exposed by the lamp actuators. He can use many features of the SocIoTal tool such as auto-completion, detection of incoherent or conflicting rules, etc. Once he writes the application by the dedicated language, he compiles checks and deploys it into the SocIoTal platform.

3.1.3 Use Case: Smart home application deployment

Daisy has bought some nice designed communicating lamp for her house. Once turned on, it is discovered by the SocIoTal system which communicates with the server of the lamp provider to get some more information about its capabilities. Once it has enough information, SocIoTal system proposes to Daisy a list of applications that can make use of the lamp features, and which are already developed and available in SocIoTal app store. Some

examples are: changing the color based on the weather outside; turn on when presence detected; blink when her mother sends a message indicating that she wants to talk with her; etc.

3.1.4 Use Case: Healthcare

Donald is diabetic so he needs to take his medicines daily. He is unfortunately forgetting to take them regularly. His caregivers have decided to build a reminder application by using different devices at his house that can be used to give some reminder messages (TV, speakers, PC screen, etc.). For instance every day at a given time, if the TV is on, it is paused and a message is displayed, if there is a music playing, it is stopped and a reminder message is sent instead, etc. To build this application, the caregiver uses the SocIoTal tool to explore the devices and configure an application that realizes the reminder application by using the APIs provided by the devices

3.1.5 Mockup/UIs Related to Service Developer Use Cases

An enhanced graphical interface would facilitate the development of IoT applications. A mockup of such interface is given in Figure 12. The navigator view at left shows the available IoT devices ready to be used. This view should be automatically updated when devices join or leave the environment. Moreover, in the navigator, the developer should find services provided by a device and for any service, and list the resources exposed by this latter.

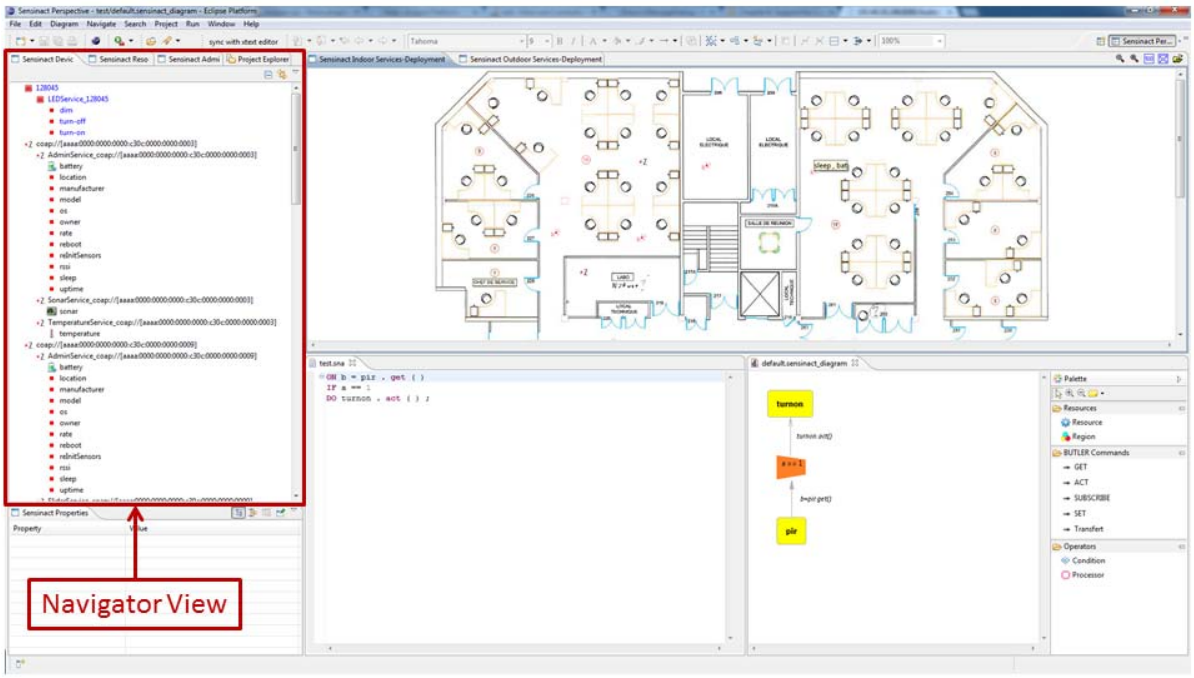

Figure 12 Mockup: Navigator view in Service Developer Environment

Resources may represent information (properties, state variables, sensed data) or functionalities (actions) exposed by services, for example if we consider a temperature service, it will provide access to a temperature resource that maps a thermometer device and through which we can get the temperature measurements, on demand, periodically or if certain conditions are met. The Figure 13 shows an example of a list of resources:

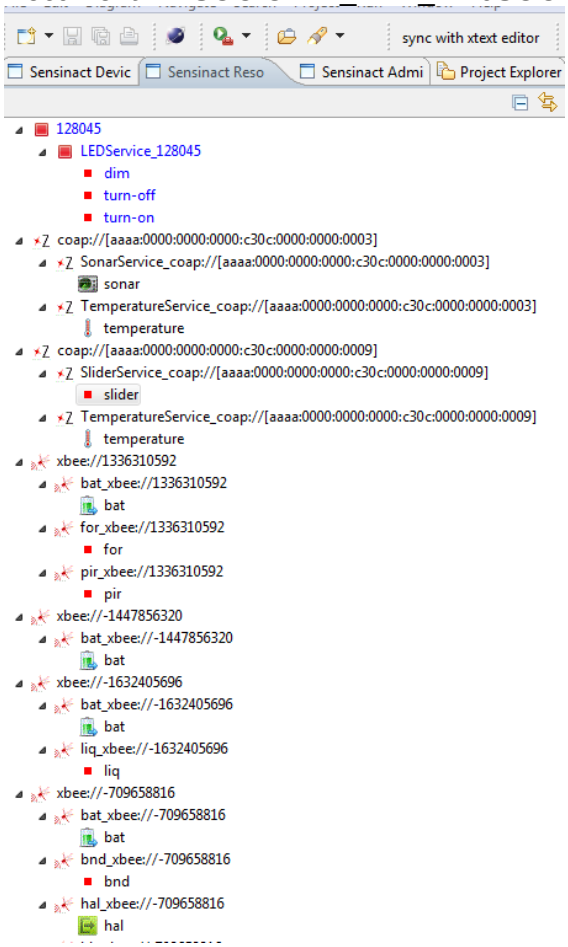

Figure 13 Mockup: List of resources in the Service Developer Environment

The obtained measurements from the sensor devices can be graphically visualized or actions can be performed remotely on actuators.

A deployment view (top of the Figure 14) can show the location of indoor and outdoor devices. This can help the user to visualize all the available IoT devices in the system and use them for the development of their applications. The locations of the devices can be updated by the user.

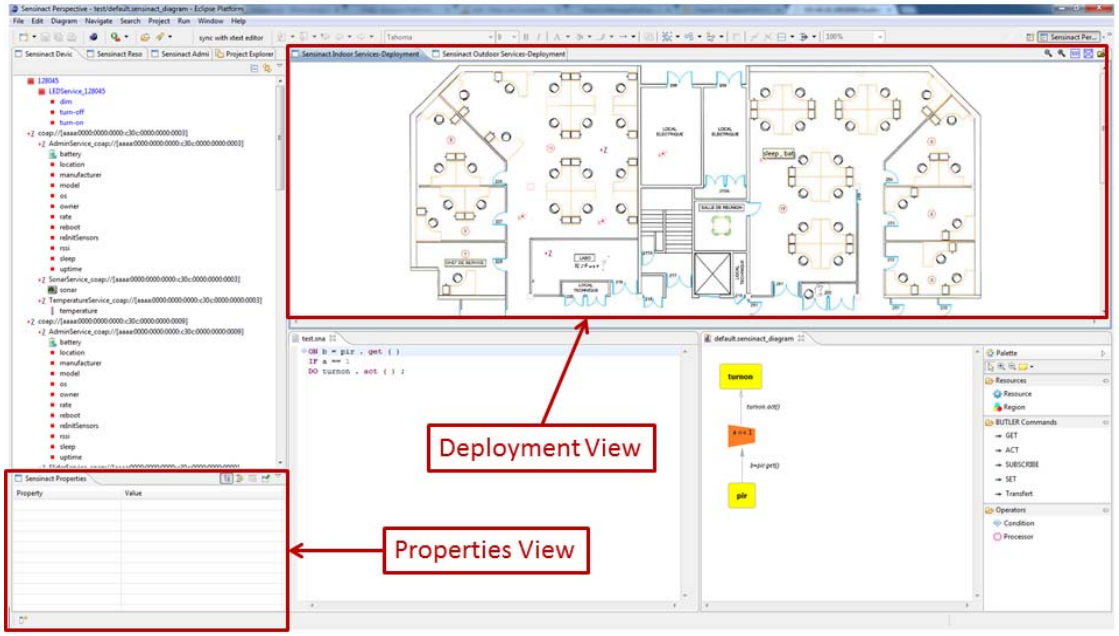

Figure 14 Mockup: Properties and Deployment view in the Service Developer Environment

Properties view (bottom left of the Figure 14) gives more information about manipulated entities in the GUI and allows to modify them.

The GUI includes two other views, namely domain specific language, DSL, (at the middle bottom) editor and Graphical Description Language, GDL (at the bottom right of Figure 15).

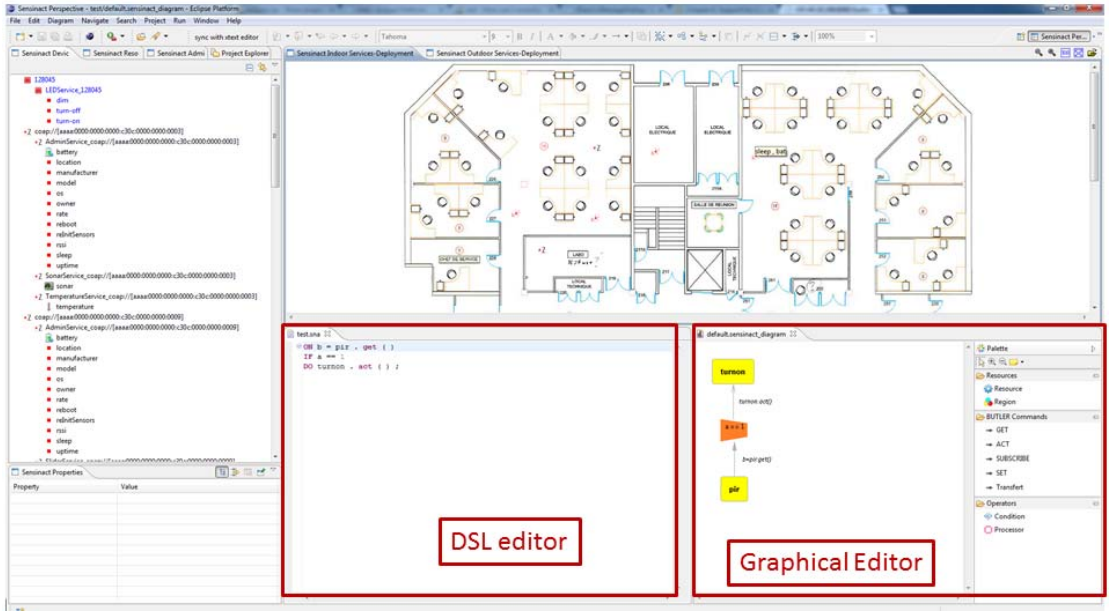

Figure 15 Mockup: The GDL and DSL views should be auto-synchronized

The DSL is used by developers/end-users to help them writing their IoT applications without requiring technical details of the devices, execution platforms, etc. The DSL can implement a language (such as an ECA language) that should be simple to use. The idea behind the simplified language is to create applications in terms of rules that are triggered when

interesting events occur. After verification of a condition a planned action can be performed. The current mockup is based on the XText environment[19](Figure 16) that offers a complete environment to implement DSLs and interesting functionalities such as syntax checking and autocompleting when writing programs.

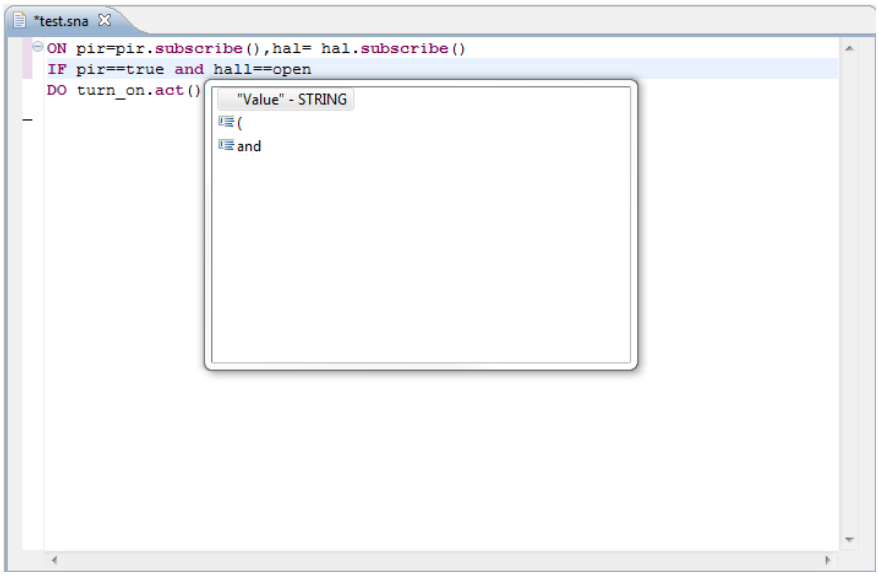

Figure 16 XText component inside the Service Developer Environment

On the other hand, a graphical description corresponding to the ECA language can follow the same logic of device description to facilitate application development. The user can find resource, access methods and condition entities in the tools bar of the GDL (at right of Figure 17) in order to link the resources between them with conditions. The DSL and the GDL editors are synchronized automatically in order to help the developer to choose both ways of creating his applications.

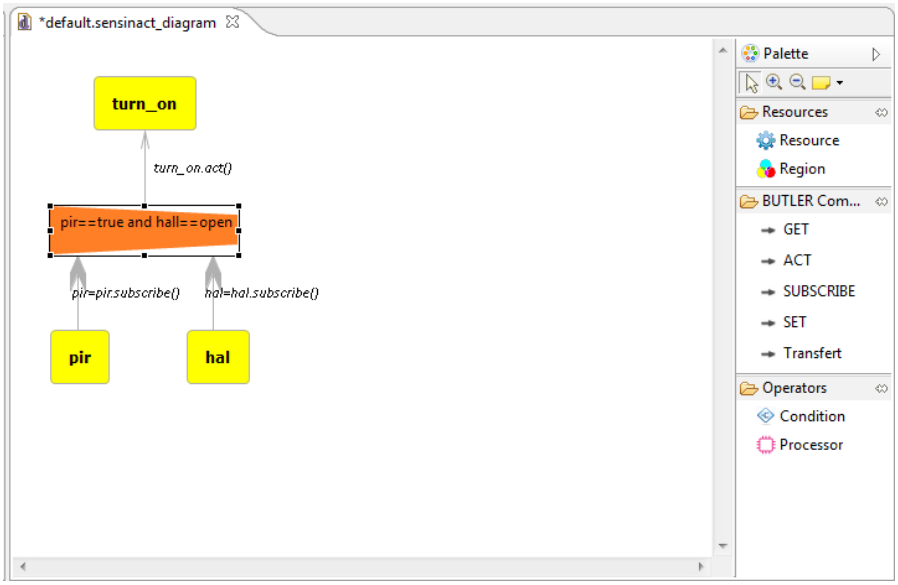

Figure 17 The GDL editor inside the Service Development Environment

To develop an ECA with the GDL, the user can use Drag and Drop functionalities from the deployment view. That allows user deciding which devices to use to develop his application with the ECA rules. The user also can use the GDL tool bar to create a device resource, a condition and connections between them. The property view can be used to assist the user defining a condition details (this functionality use also Xtext capability in the property view).

3.2 State-of-the-art of Service Developer Tools in IoT

This section gives an overview of some state of the art tools that aim at helping developers (experienced developers, non-technical users, etc.) to build applications by service composing or mashup in various domains. The components presented in this section include solutions from two main domains: data mashup tools (WireCloud, Mycocktail, OpenSocial, and Apache Shinding) and service/business process composition solutions (EMML, BPMN and BPEL).
The aim of this section is provide an evaluation of the above tools according to some defined criteria. The benchmarks are a quick and effective way to make an assessment. These criteria are in accordance with the objective of highlighting some lacking aspects that are required by Sociotal developer environment.

3.2.1 Comparison Criteria.

The criteria we used are divided in two parts: one part is about key features and another part of criteria is about openness and interoperability. In next subsections, data mashup tools (like WireCloud, Mycocktail, OpenSocial, and Apache Shinding) and service/business process composition solutions (like EMML , BPMN and BPEL) are analyzed. Here follows the set of key features criteria used for comparison:

- Target:Based on the recent technological advancements, the boundary between the software developer and end-user is being increasingly blurred. For such reason, among the tools analyzed in this section some are targeted to end-users and some are targeted to both developers and end-users.
- Principal capabilities: to know if the software covers project task requirements.
- Development time cost: depending on target users, evaluate time cost.
- Known limitations: if some limitations are instead mandatory for the project task.

Here follows a set of criteria related to openness, interoperability and integration issues:

- License: open sources are particularly encouraged,
- API: should be easy to use and extend,
- Adopted standard: the use of standards foster interoperability,
- Type of Implementation: highlights integration efforts.

3.2.2 Analysis of tools

3.2.2.1 Wirecloud

The Wirecloud Mashup Platform[20] (see Figure 18) is a framework that allows end-users to create specific mashups, composing widgets that are the building blocks to create new applications. The widgets (also called gadgets) can be downloaded from a catalogue and can be provided by 3rd parties. In particular, the Wirecloud Mashup Platform is composed by three different components: the Application Mashup Editor, the Mashup Execution Engine and the Catalogue.

The Application Mashup Editor is a web-based composition editor used by the end users to compose their own applications. The editor is a workspace where the users can graphically connect the widgets, downloaded from the catalogue, performing an input-output mapping. The widgets can be connected to back-end services or data sources through an extendable set of operators, including filters, aggregators and adapters.

The Mashup Execution Engine is the component that provides the mashup functionalities, coordinating the gadgets execution and communication, mashup, state persistence, and cross-domain proxy. The engine functionalities are exposed to the editor through a set of API: new external modules can be added in order to extend the functionalities.
The gadgets used in the editor can be found and downloaded from the Catalogue, which is an independent component with respect to the previous two: the gadgets published in the catalogue are described in a standard description language. In particular the gadgets can be described through an XML Widget description language and with RDF widget description language. The mashups created by the users can be published in the same catalogue: additionally, the catalogue allows to tag and rate widgets to foster discoverability and share ability.

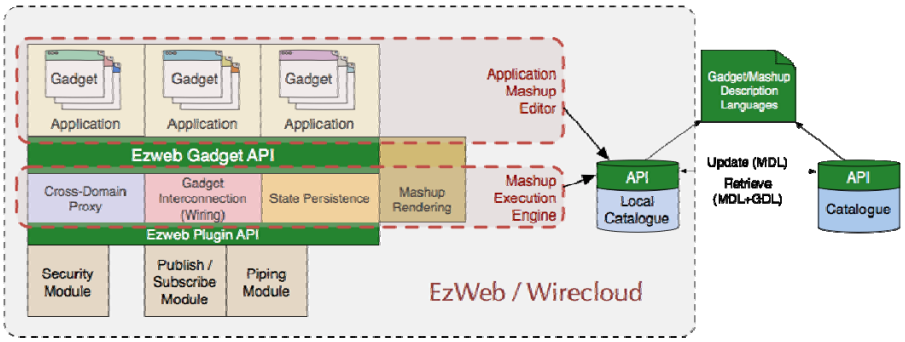

Figure 18 WIRECLOUD Architecture

The Wirecloud Mashup Platform[20] has been developed in the FI-WARE project as Generic Enabler reference implementation: it supports linked-USDL and it is integrated with FI-WARE's Marketplace, Stores and Repositories. See the tables below for a summary of the analysis.

3.2.2.2 MyCocktail

MyCocktail is a web application that provides a graphical user interface to easily build mashup easily. MyCocktail provides two different web tools(Figure 19 and Figure 20): the mashup builder and the page editor. The mashup builder is a graphical environment that allows combining and modifying data, using specific filters and information coming from REST services, and connecting these data with web gadgets, creating mashups. The gadgets can be exported as Google or Netvibes gadgets being compliant with the open social specification. In particular there are three different types of elements that can be combined in the builder:

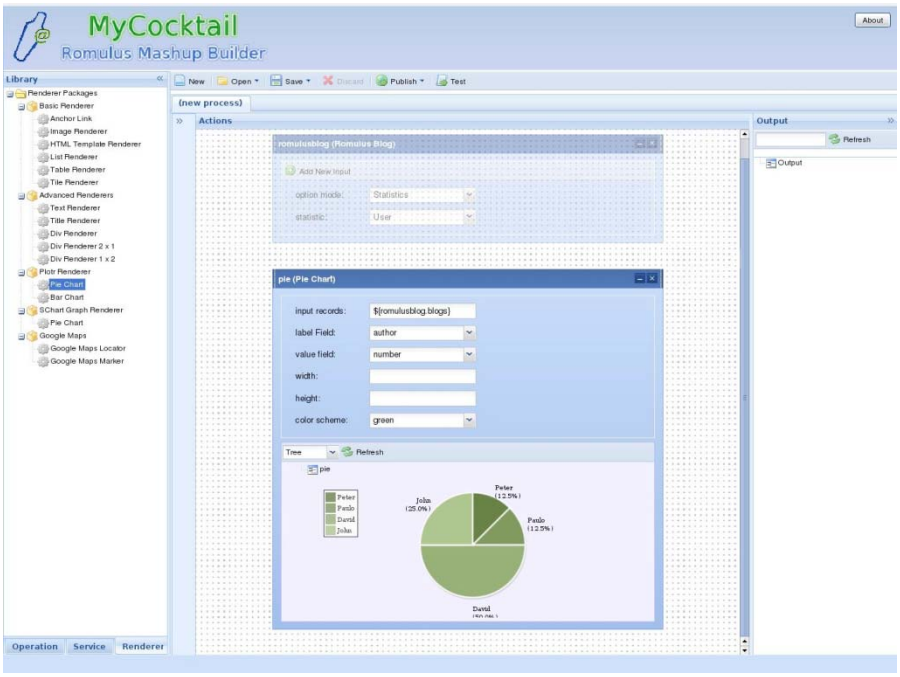

Figure 19 MYCocktail web tool

- Services: it is possible to use in the mashup information obtained by specific services. MyCocktail provides a set of predefined services to get specific information from several popular services or social networks such as Amazon, Delicious, Flickr, Google, Twitter etc. For example, it is possible to perform a specific search on Google or get information about the Twitter followers. Also it is possible to import in the editor new custom REST services using the WADL descriptions (Web Application Description Language).
- Operators: the data objects obtained from the services invocation can be manipulated using a set of operators. These operators can work on objects, arrays or strings performing series of functions such as sorting, parsing, splitting etc.
- Renderers: are graphical elements and widgets that allow rendering the information obtained and manipulated using services and operators. The final graphical output can be exported in different format such as Google or Netvibes gadgets.

The Page Editor allows designing a web page through a GUI allowing integrating mashups created with MyCocktail with other HTML elements. It is based on FCKEditor, a web text editor. The page editor has a complete toolbar in which the elements and the properties to be included in the design area that shows the result page can be selected.

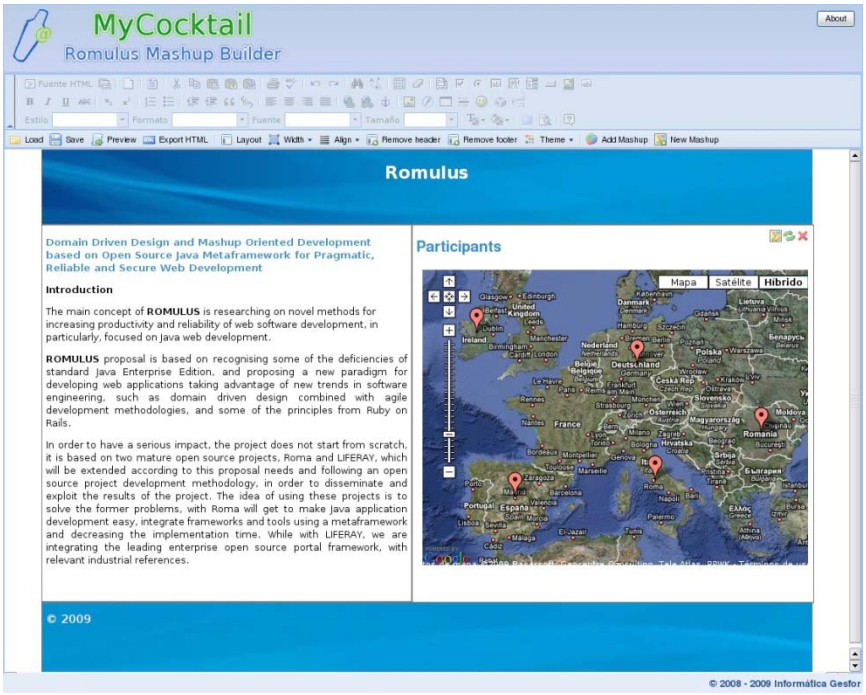

Figure 20 Open MyCocktail

3.2.2.3 OpenSocial

OpenSocial[21] is a set of common APIs, initially developed by Google, in order to create a single programming model for social applications such as gadgets that can operate on any social network that uses this standard.
Based on HTML and JavaScript, as well as the Google Gadgets framework, OpenSocial includes multiple APIs for social software applications to access data and core functions on participating social networks. Each API addresses a different aspect. It also includes APIs for contacting arbitrary third party services on the web using a proxy system and OAuth for security.
In particular, OpenSocial provides four different categories of APIs that can be used to:

- develop Social Application, such as gadgets, that run in specified OpenSocial containers
- implement Web Container: host social networking environments in which the social applications can be executed
- allow interaction between web sites and social networks that support the Open Social standard. In particular, existing sites can have access to several information, for instance:
    - the user profile (user data);
    - information about user's friends (social graph);
    - profile activities (posts, photos, video, news feeds etc.)

The goal of the project is, therefore, to:

- increase power and pervasiveness of the social Web capabilities, providing to the developers the tools needed to build applications that can be accessed by an ever increasing number of users, regardless of social networks used;
- increase interoperability between applications;
- promoting the portability of web-based components;
- stimulate the creativity of the user, in order to have new ideas "from below";
- create an open and free "framework" that is compatible with the highest number of social networks.

OpenSocial is not a product or a service distributed by Google, but an open standard supported by a large community of developers. Furthermore, it is important to highlight how the project is significantly different from what has been achieved in recent years by Facebook which, despite the huge community of developers, has focused mainly on the use of proprietary protocols and languages, maintaining a more conservative approach.

3.2.2.4 Apache Shindig

Shindig[22] is an open source project of the Apache Software Foundation that has the aim to implement an open container in compliance with the Open Social gadgets specifications. According to the Apache Foundation, the project's goal "is to allow new sites to start hosting the social apps in less than an hour's work", creating an infrastructure able to render gadgets, process proxy requests, and handle REST and RPC requests. In the applications market there are many examples of containers that are based on Shindig, such as LinkedIn, hi5, Partuza, WSO2 or Jartuza.

From the architectural point of view, Shindig consists of four basic components:

- Gadget Container: component written in JavaScript that manages gadgets functionality such as safety, communication, graphical user interface and the access to the OpenSocial API.

- Gadget Rendering Server: server used to render the gadget XML file in JavaScript and HTML for the container in order to expose the gadget through the Gadget Container.
- OpenSocial Container: the JavaScript environment that lies on the top of the Gadget Container and provides OpenSocial specific functionality, such as profiles, list of friends, activities and data storage.
- OpenSocial Date Server: is the implementation of a server interface for the access to specific information of the container, and it includes the OpenSocial REST API.

Shindig, from architectural point of view (Figure 21), has a client and a server components. In particular the client part of the system is composed by three elements:

- The container for gadgets, fully compliant with the OpenSocial gadgets specifications (GadgetsContainer).
- The OpenSocial container itself (Opensocial Container).
- The container that supports the exchange of data in the JSON and Caja format to the REST standard (RestfulContainer).

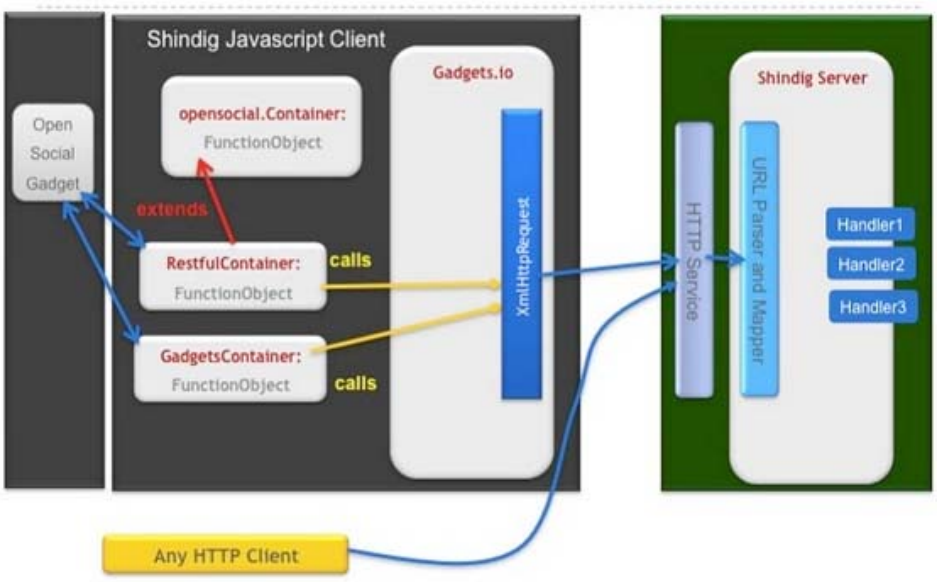

Figure 21 apache shindig overall architecture

In particular, the latter is an extension of the component OpenSocial container and deals to pass all calls to the API OpenSocial REST endpoint that will take care of both, the direct HTTP calls and those from the OpenSocial API. All calls to the server will be executed by the RestfulContainer and GadgetsContainer through the XmlHttpRequest Gadgets.io that will instantiate an object in order to send requests to HTTP Server.

The Shindig components of the server side are (see Figure 22):

- Persistent Data Access Layer: loading mechanism of the persistent data.
- Gadget Rendering Server Components: infrastructure for rendering gadgets.
- Open Social Components Server: server-side implementations of the OpenSocial API.

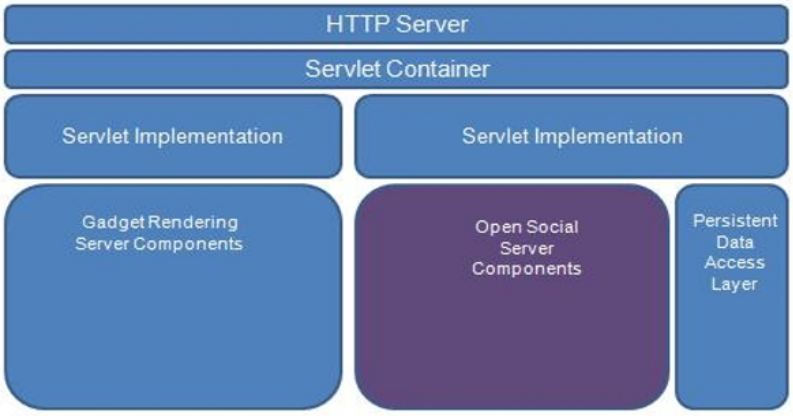

Figure 22 apache shindig server architecture

There are currently two versions of Apache Shindig written in Java and PHP, and a third, written in .NET that is external to the project.

3.2.2.5 Enterprise Mashup Markup Language

EMML (Enterprise Mashup Markup Language) [23] is an XMLmarkup language for creating enterprise mashups promoted by the Open Mashup Alliance. EMML is a declarative mashup domain-specific language (DSL) that eliminates the need for complex, time-consuming and repeatable procedural programming logic to create enterprise mashups.
EMML provides a mashup-domain vocabulary to consume different data-sources. Additionally, the EMML provides a uniform syntax to invoke heterogeneous service technologies (e.g. Web services, rest services, RSS etc.) and to combine different data formats (JSON, XML, JDBC etc).
EMML files can be considered scripts that describe the process flow for a mashup: these scripts must be processed by an EMML engine that interprets EMML statements to perform the mashup. The EMML language provides several functionalities such as:

- Data manipulation using filters
- Connection with heterogeneous services
- Semantic annotation of services
- Merge and split datasets
- Support for scripting languages (e.g. JavaScript, JRuby, Groovy, XQuery)
- Conditional statements
- Data scraping from HTML page

The Open Mashup Alliance has made the EMML schema available for download as well as an EMML reference runtime implementation that processes mashup scripts written in EMML.

3.2.2.6 Business Process Modeling Notation

The Business Process Modeling Notation (BPMN) [24] is a graphical notation that depicts the steps in a business process. A business process spans multiple participants and coordination can be complex. Moreover, well-supported standard modeling notation will reduce confusion among business and IT end-users. BPMN depicts the end to end flow of a business process. The notation has been specifically designed to coordinate the sequence of processes and the messages that flow between different process participants in a related set of activities. A Business Process Diagram (BPD) is made up of a set of graphical elements. These elements enable the easy development of simple diagrams that will look familiar to most business analysts (e.g., a flowchart diagram). The elements were chosen to be distinguishable from each other and to utilize shapes that are familiar to most modelers. For example, activities are rectangles and decisions are diamonds. It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for depicting business process models, while at the same time being able to handle the complexity inherent to business processes. The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories. This provides a small set of notation categories so that the reader of a BPD can easily recognize the basic types of elements and understand the diagram. Within the basic categories of elements, additional variation and information can be added to support the requirements for complexity without dramatically changing the basic look-and-feel of the diagram. The four basic categories of elements are: Flow Objects, Connecting Objects, Swim lanes and Artifacts.
A key goal in the effort to develop BPMN to help alleviate the modeling technical gap, was to create a bridge from the business-oriented process modeling notation to IT-oriented execution languages that will implement the processes within a business process management system. The graphical objects of BPMN, supported by a rich set of object attributes, have been mapped to the Business Process Execution Language for Web Services, the standard for process execution. For example, the core of jBPM [25] is a light-weight, extensible workflow engine written in pure Java that allows you to execute business

processes using the latest BPMN 2.0 specification. It can run in any Java environment, embedded in your application or as a service.
Other solutions to BPMN exist for example, UML Activity Diagram, UML EDOC Business Processes, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, and Event-Process Chains (EPCs). The following section illustrates the difference between UML and BPMN and thus it helps to understand more the relationships between these solutions.

BPMN & UML
The unified modeling language (UML) takes an object-oriented approach for modeling applications, while BPMN takes a process-oriented approach for modeling systems. Where BPMN has a focus on business processes, the UML has a focus on software design and therefore the two are not competing notations but are different views on systems. The BPMN and the UML are compatible with each other. A business process model does not necessarily have to be implemented as an automated business process in a process execution language. Where this is the case, business processes and participants can be mapped to constructs such as use cases and behavioral models in the UML.

BPMN to XML
The XML serialization for BPMN is provided in machine-readable form, which has the OMG Document Number dtc/2010-06-03. It is an international standard formally approved by OMG, it creates XML code that is generated when a person creates a process model.

3.2.2.7 BPEL
BPEL[26][27] is the standard for assembling a set of discrete services into an end-to-end process flow, radically reducing the cost and complexity of process integration initiatives.
BPEL is an orchestration language, and not a choreography language. The primary difference between orchestration and choreography relies on the executablility and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. Choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g., in the form of a BPEL process) for each peer involved in it.
BPEL has no graphical notation witch conducted to a variety of different notations causing ambiguities and misunderstanding. BPMN is then used to fill this gap.
The Figure 23 explains more the relationship between BPEL and BPMN using a concrete example. Details of comparison available are in [28].

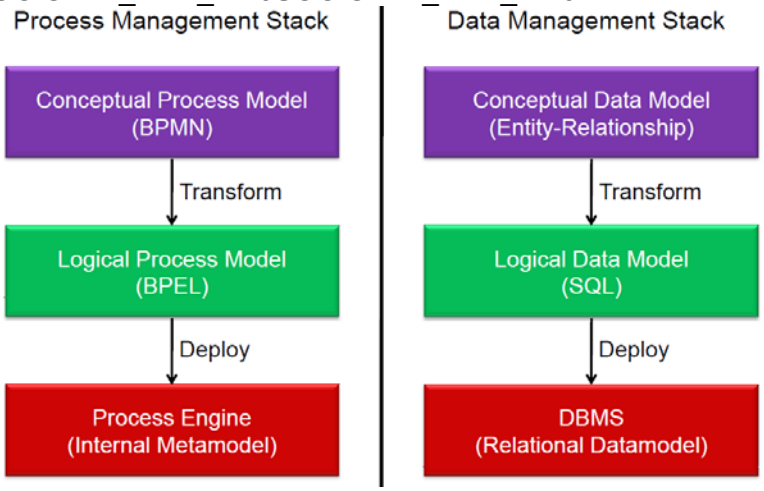

Figure 23 Relationship among BPMN and BPEL management stacks

Among the notable implementations of BPEL, Oracle BPEL process Manager is a tool for designing and running business processes. This product provides a comprehensive, standards-based and easy to use solution for creating, deploying and managing cross-application business processes with both automated and human workflow steps – all in a service-oriented architecture. Oracle BPEL Process Manager could be used in isolation to implement business processes, but the real power comes when it is used in conjunction with other SOA components. For example, you can instrument your BPEL processes using the Oracle BPEL Process Manager's sensor framework. Sensors can fire events under conditions specified by the administrators and send events to any endpoint that administrators choose. This makes it easy to monitor processes when there are hundreds or thousands running in parallel.

3.2.3 Summary of comparison

The tables below summarizes the given criteria applicable to the above tools and systems.

Tool	Target	Principal Capabilities	Development time cost	Known Limitations
Wirecloud	End users, software developers	Graphical editor: allows connecting the gadgets data. Catalogue: allows to share mashups with other stakeholder of the ecosystem fostering reuse of services in different contexts	Average	The platform doesn't allow creating or modifying the widgets using the graphical editor widgets cannot be exported or shared in different containers
Mycocktail	End users, software developers	Mashup builder: a graphical environment that allows combining and modifying data, using specific filters. The Page Editor: allows designing a web page through a GUI allowing integrating mashups created with MyCocktail with other HTML elements.	Average	It doesn't provide a clear way for the representation of the mashups, such as arrows that connect the different elements
OpenSocial	Software Developers	OpenSocial: a set of common APIs, based on HTML and JavaScript, for social software applications that allow to access data and core functions on participating social networks.	Heavy	No particular limitation has been identified at this stage
Apache Shinding	Software Developers	A container for hosting social application providing a JavaScript environment.	Average	No particular limitation has been identified at this stage
Entreprise Mashup ML	Software developers	A declarative mashup domain-specific language (DSL) that eliminates the need for complex, time-consuming, procedural programming logic.	Light but require implementation	The language can be freely reused but its license does not allow the language modification or extension.

BPMN	Software Developers	A graphical notation that depicts the steps in a business process.	Heavy	A business process spans multiple participants and coordination can be complex. But, well-supported standard modelling notation will reduce confusion among business and IT end-users. Still this language is not adapted to IoT domain
BPEL	Software Developers	A standard for assembling a set of discrete services into an end-to-end process flow.	Heavy	Mature technology with many implementations but not adapted to IoT domain

Table 5 Summary of key features for all the tools analysed in the domain of the SocIoTal Developer Environment

Tool	License	API	Adopted Standard	Type of Implementation
Wirecloud	Affero General Public License version 3 (AGPL v.3) : http://www.gnu.org/licenses/agpl-3.0.html	The Widget API is a JavaScript API that allows deployed widgets in a Mashup Execution Engine to gain access to its functionalities The Application Mashup API is a RESTful API that provides the functionality to create and modify workspaces and to manage the resources available for building these workspaces.	RESTFul, HTTP	Web application Server side: Python Client side: Javascript, HTML, CSS
Mycocktail	Apache License, Version 2.0, (http://www.apache.org/licenses/LICENSE-2.0)	None	Open Social Specifications, WADL, Rest	Web Application Server side: Java Client side: JavaScript, HTML , CSS
OpenSocial	Apache License 2.0 (http://www.apache.org	OpenSocial is a set of common APIs.	HTML, OpenSoci	Web application framework:

	/licenses/LICENSE-2.0)		al	Java, PHP, C#, JavaScript, HTML
Apache Shinding	Apache License 2.0 (http://www.apache.org/licenses/LICENSE-2.0)	OpenSocial API specifications	OpenSocial REST API	Application framework: JavaScript, PHP and Java
Entreprise Mashup ML	Creative Commons License of Attribution No Derivatives: http://creativecommons.org/licenses/by-nd/3.0/	None	XML	Language, implementation can be found.
BPMN	Open source and commercial implementations exist	Provides a complete specification.	UML, XML	Graphical Language, implementation can be found.
BPEL	OASIS standard, open source and commercial implementations exist.	A complete API from Oracle.	XML, WSDL	Language Implementations in Java and .NET.

Table 6 Summary of criteria related to openness, interoperability and integration for tools analysed in the domain of SocIoTal Developer Environment

3.2.4 Advancements and success indicators

Our proposal follows an IoT approach taking into account the different actors of the architecture: heterogeneous devices, communication protocols, data representation and the development interface. The user of the tool is able to integrate his devices, to deploy them, and to use in applications without asking a large development effort.
The DevelEnvs based on a flexible architecture and modular software so inclusion of new mechanisms can be done quite easily. Using Event-Condition-Action rules, the development tool offers the possibility to perform actions on devices anywhere at runtime, according to some conditions and events. It is a powerful and flexible tool which responds to complex situations. In the following sections, we present major capabilities of the tool.

Rules language

A domain specific language will be used by developers/end-users to help them writing their IoT applications without requiring technical details of the devices, execution platforms, etc. This DSL implements an ECA language which is designed to be simple to use. The following lines give the first version of ECA rule in Xtext format.

```
Sensinact:
   {Sensinact}
   (eca=ECA) ';'
;

ECA:
   E init=INIT? (ca+=CA)+ elca=ELCA?
;
E:
  'on' sourceevents+= EventResource ( ',' sourceevents+=EventResource)*
;
```

```
INIT:
  'init' initData+= DataResource ( ',' initData+=DataResource)*
;

CA:
   IF act=ACTION
;

IF:
   'if' ifcdt=ConditionDefinition
;

ACTION:
   'do' targetactions+=ActionResource( ',' targetactions+=ActionResource)*
;

ELCA:
  ('else') elseact=ACTION
;
```

Using ECAs, it is possible to write IoT applications like in the following example: when a presence is detected under a city lamp, the application turns on the light of, otherwise it turns it off.

```
ON presence=PIRService.PIR.subscribe()

IF presence==true

DO  LightService.lightOn.act();

ELSE DO  LightService.lightOff.act();
```

Behavior verification

Event-Condition-Action is a powerful and flexible tool to respond to complex situations. However, the behavior of a rules based system is difficult to analyze because of the ability of rules to interact with each other in social environment. Particularly, the rules that govern the relations between sensors and actuators will lead to highly distributed collaborative applications. Runtime coordination and formal analysis becomes a necessity to avoid side effects mainly when applications are critical. This document presents a case study for safe applications development in IoT. The developer tool will integrate such verification mechanism to provide safe interaction properties and support ECA distribution. The following are the main properties:

Circularity occurs when, due to the result of rule actions, rules get activated consecutively without an ending condition. This usually generates undesired and uncontrolled behaviors, making rules to execute continuously.

a. Redundancy of rules is detected when, in a couple of rules, the condition and actions of one rule represent a subset of conditions and actions of the other rule.
b. Inconsistency is detected at compilation time. It can be defined as the result of contradictory actions, provided as result of activation of different rules.

Stream events

Complex Event Processing (CEP) emerged from Stream Processing as a way to facilitate the development of “Filters/Actors” and make it “declarative”. In this context, the software development provides the user the capability to formulate ECA rules that involve patterns of events. In fact, the IoT broker that feeds the CEP engine with devices events is able to manage subscriptions to event patterns. In this case, software developers don’t have to care about patterns handling when writing ECA based applications.

Regarding the previous tables, the data mash up tools are oriented to non-technical end-users, in order to create rapidly applications from existing data sources whilst service composition tools target more experienced developers that require a minimum dependability and Quality of Service requirements. In addition, mash up technologies are mostly web based platforms while service composition tools may require more sophisticated gadgets in order to provide required properties to the services.

And even though these solutions are interesting, their implementations require abstraction of the deployment level of devices, adaptation to the requirements of used protocols and formatting of exchanged data. These imply a very significant integration effort. It can be even more substantial in an area that presents a lot of heterogeneity such as the IoT/Societal. We believe that a development solution must support the integration at all levels using flexible and simple models.

Success indicators

In terms of **success indicators** focused on the service development tool, they will come up from two different angles: the first one is related to the Quality of Service (QoS) evaluation that assesses the performance of the tool by considering key performance indicators such as time of development, time of deployment, time of response to a service call, time of reaction to arrival of new services, number of composed services and availability percentage, etc. The second one is the evaluation of developer’s Quality of Experience (QoE), which can be performed to measure their satisfaction by using indicators such as time of learning, number of services created using the tool, the lifetime of created services, number of users of the tool, etc. The QoS will firstly be evaluated in the lab environment, while the quality of user experience can only be evaluated by involving real developers/end-users.

3.3 List of SocIoTal Developer Environment Features

Discovery of IoT devices
The developer environment is automatically updated about the available IoT devices/service in the environment so that the developer can make use of the available devices in his applications
Deployment view
The developer has a view on how the devices are deployed physically in the environment so that the developer can decide which devices to use, since the geographical location of the devices has an important role on the decision of the list of devices to use.
Properties view
The developer/administrator can use the properties view to get information on the devices and possibly modify them
A programming editor
The developer has a programming editor with features such as auto-completion and syntactic checking. The editor is used to build applications based on the APIs exposed by the devices/services.
A graphical editor
The developer has a graphical editor that can help him to build applications by composing available services and defining the interactions between them (with arrows between boxes representing services). The graphical editor should synchronise with the textual editor.
Conflict detection
The development environment detects redundant, conflicting or inconsistent applications at development time in order to avoid undesired situations during the execution.
Deployment
The tool provides means to remotely deploy the developed applications on target platforms
Gathering sensor measurements
The tool provides supervision features that allow gathering sensor measurement in real time (on demand or continuously)
Graphs on sensor measurement
The tool provides graphical widgets to display sensor measurement values
Actions on devices
The tool provides ways of remotely invoking actions on the actuators

Table 7 List of features of the Service Development Enviroment

3.4 Architecture of the SocIoTal Developer Environment tool

The architecture of the development tool is composed of the following main components:

Graphical User Interface component provides all graphical elements to help, guide and assist the developer. The interface has several views, each one having particular functions. The most useful ones are: navigator view to explore existing device services; properties view to see different property values of services; deployment view showing physically where the devices are deployed (indoor or outdoor); textual editor view for programming with a domain specific language and graphical editor view for graphically composing services.

Domain Specific Language component is a dedicated programming language environment to write IoT specific applications. Using domain specific languages allows developers to focus on the application logic and omitting device and platform specific technical details in terms of system, communication protocols, etc. The DSL is agnostic to general purpose programming languages, so that it can be transformed to any other programming language necessary to communicate with the IoT platform. The DSL can provide features such as auto-completion, syntax verification, type checking, etc. to help to the developer and accelerate his work.

Graphical Description Language component provides a visual assistance to the developer. By using boxes and arrows paradigm, the services can be connected together via different types of calls between services. Usually in case of IoT, which is a data centric environment, the service calls will be transformed into data flows from sensor data services to action providing services with some data processing services in the middle. GDL should be synchronized with the textual program written in DSL, so that the developer can alternate between two when necessary. For this purpose, a common **service model** is shared between the two components.

Service Composition module is responsible of combining the services based on the program described by the DSL and/or the GDL. It creates necessary code to effectively make the services interact by service calls and set up the necessary channels for data flowing from one service to another. It generates necessary code to perform this and deploy it to the platform. Before deployment, optionally, the application can be sent to an **application validator** that can check the behavior of the created application and be sure that it does not violate some already defined properties (conflicts, constraints, deadlocks, etc.) so that it will not produce unexpected results.

Once the application is ready, the **application deployer** can finally bundle the application (e.g., into a .jar or .war file) and deploy it either to the IoT platform itself or to an external platform, for instance in the cloud. The deployer needs to have necessary authorization to perform the deployment. The installed application can have some dependencies to some external libraries that are not present on the platform. The dependency management needs to be managed by the platform itself (ideally) or the deployer needs to provide all the dependent module within the application bundle. Once deployed, **application management** component can track the execution of the application for different purposes such as accounting, supervision, updating, uninstalling, etc.

In order to interact with the IoT platform(s), a **platform connector** is necessary to make the interface with the developer environment. It allows gathering all available devices/services in the environment, as well as notifying when some changes occur (e.g., arrival/leave of services). The platform connector will mostly use REST type interfaces in order to remotely gather necessary information from the platform. The platform thus must provide such

features to the development environment. It needs to have necessary authorization to access to the IoT platform.

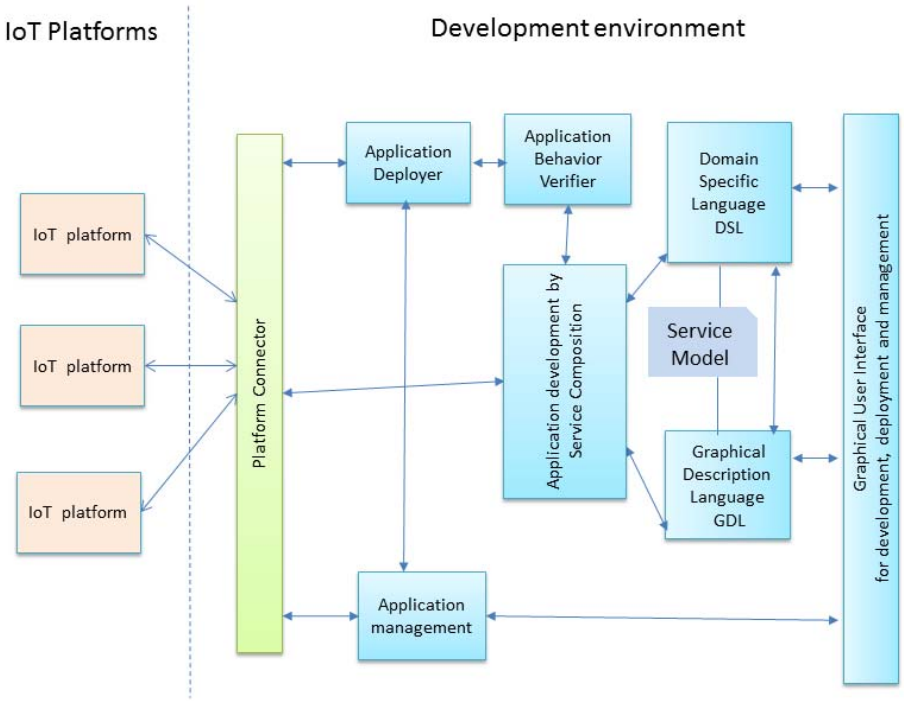

Figure 24 architectural high level view of the service developer environment

Section 4 - Conclusions

This document reports the activities of analysis and feature specifications for two tools: the SocIoTal end user environment and SocIoTal Developer environment. The two components are aimed at different targets and provide different tools but both are inspired by the main SocIoTal objective to lower the barrier of IoT adoption for citizens.
For both tools, the approach is to define features that come from some depicted use cases using, when applicable, mock-ups and pictures to show how the tools are presented to users and to developers. Given the early stage of the SocIoTal architecture at the time of writing, the whole features specification of UserEnv and DevelEnv makes some relevant assumptions. The tools are assumed to work at highest Internet levels, thus they will depend on a IoT platform or gateway that will address all the low level communication issues related to real devices (eg. when a device cannot “speak” HTTP protocol then a gateway will provide the right API to allow the communication).

As one of the core claims of SocIoTal is to provide an innovative and solid framework to manage security, reputation and trust in the context of IoT, in this deliverable are introduced, even if with a coarse and immature design, some of the future needed integration with the work packages WP2 and WP3. These WPs will provide the core enablers that will conform the core of the envisioned SocIoTal platform (or set of tools). Here, the proposed platform connector module will be boundary between WP2 and 3 and the developments initially proposed in WP4 (the UserEnv and the DevelEnv) and so, it is foreseen to concentrate the main collaborative efforts among these WPs. In Section 1, are illustrated which services will be exposed to WP4 to achieve the features of contextual policy, security, and trustworthiness. In Section 2, the UserEnv features are enriched with the above features and some early details are provided.

The two tools are described in depth inside Section 2 (UserEnv) and Section 3 (DevelEnv). For both tools is also presented a context and a state-of-the-art of tools with one or more similar characteristics. To summarize, it is highlighted that to lower the barrier of IoT adoption bottom-up, the WP4 tools will achieve the characteristics of being creative, assistive, and trustworthy. Along these lines, the details of features and some architectural specification are presented for both the tools.

Section 5 - Bibliography

[1] IoT-A terminology. Available at http://www.iot-a.eu/public/terminology/copy_of_term, [Feb. 19,2014]
[2] Lieberman, H., Paternò, F. and Wulf, V. (2006). End User Development.
[3] Harrison, Warren (July/August 2004). "The Dangers of End-User Programming" (PDF). IEEE Software 21 (4): 5.
[4] Pliskin, Nava; Shoval, Peretz (1987). "End-user prototyping: sophisticated users supporting system development". ACM SIGMIS Database 4 (4): 7–17.
[5] Davide Carboni and Pietro Zanarini. 2007. Wireless wires: let the user build the ubiquitous computer. In Proceedings of the 6th international conference on Mobile and ubiquitous multimedia (MUM '07). ACM, New York, NY, USA, 169-175
[6] Touch & compose: Physical user interface for application composition in smart environments I Sánchez, J Riekki, M Pyykkonen - Near Field Communication, 2009. NFC'09. First International Workshop on, 61-66
[7] SocIoTal D1.1 SocIoTal scenarios and requirements definition report
[8] IF THIS THEN THAT - http://ifttt.com
[9] Paraimpu, Connect Compose Share, the Social Tool for the Internet of Things. Available at https://www.paraimpu.com
[10] Paraimpu: a platform for a social web of things. A Pintus, D Carboni, A Piras. Proceedings of the 21st international conference companion on World Wide Web
[11] Xively by LogmeIn – Business solutions for the Internet of Things. Available at https://xively.com
[12] NODE-Red. Available at http://nodered.org/
[13] Christian Fuhrhop, John Lyle, and Shamal Faily. 2012. The webinos project. In Proceedings of the 21st international conference companion on World Wide Web (WWW '12 Companion). ACM, New York, NY, USA, 259-262.
[14] The Webinos foundation. Available at http://foundation.webinos.org/
[15] Nimbits. Available at http://www.nimbits.com/
[16] Sutcliffe, Alistair (July 2005). "Evaluating the costs and benefits of end-user development" (PDF). ACM SIGSOFT Software Engineering Notes (ACM) 30 (4): 1–4.
[17] The OAuth 2.0 Authorization Framework - http://tools.ietf.org/html/rfc6749.html
[18] Smart-Citizen. Available at http://www.smartcitizen.me/
[19] Xtext, language development made easy. Available at https://www.eclipse.org/Xtext/
[20] Wirecloud Mashup Platform - http://conwet.fi.upm.es/wirecloud/
[21] Open Social Foundation. Available at http://opensocial.org/
[22] Apache Shindig. Available at http://shindig.apache.org/
[23] Enterprise Mashup Markup Language. Available at http://en.wikipedia.org/wiki/Enterprise_Mashup_Markup_Language
[24] BPMN. Available at from http://www.bpmn.org/
[25] jBPM. Available at http://www.jboss.org/jbpm.
[26] BPEL. Available at http://www.oracle.com/technetwork/middleware/bpel/overview/index.html
[27] BPEL definition. Available at http://en.wikipedia.org/wiki/Business_Process_Execution_Language
[28] Leymann, Frank. "BPEL vs. BPMN 2.0: Should you care?." Business Process Modeling Notation. Springer Berlin Heidelberg, 2011. 8-13.

Section 6 - Glossary

AAA: Authentication, Authorization and Accounting
API: Application Programming Interface
CRUD: Set of operations for a resource, Create-Read-Update-Delete.
DevelEnv: The SocIoTal Service Developer Environment
DOW: Description of Work
DSL: Domain Specific Language
EC: European Commission
ECA: Event Condition Action (a rule language)
ETSI: European Telecommunications Standards Institute
EULA: End User License Agreement
F2F: Face-to-face, referred to personal interactions that occur in proximity
HTTP REST: Hypertext Transfer Protocol Representational State Transfer
IoT: Internet of Things
IOT-a: an architectural reference model for IoT achieved by the IOT-a project
JSON-RPC: JavaScript Object Notation - Remote Procedure Call
MQTT: Machine-to-Machine (M2M)/Internet of Things
NFC: Near Field Communication
P2P: Peer to peer
PaaS: Platform as a Service
PDS: Personal Data Storage
PZH: Personal Zone Hub
PZP: Personal Zone Proxy
QR: Quick Response Code
RFID: Radio Frequency IDentification
RSS: Really Simple Syndication
SaaS: Software as a Service
SLA: Service Level Agreement
SME: Small Medium Enterprise
SMS: Short Message Service
SVN: Subversion
UID: Unique Identifier
URL: Uniform Resource Locator
UserEnv: The SocIoTal End User Environment
UUID: Universally Unique Identifier
VE: see Virtual Entity
Virtual Entity: Computational or data element representing a Physical Entity in IOT-a model
WS: Web Service
WSN: Wireless Sensor Network
XACML: eXtensible Access Control Markup Language
XML: eXtensible Markup Language