Author(s): Caetano, Jorge; Santos, Pedro; Justino, Paulo; Goix, Laurent Walter; Renditore, Paola; Demartini, Matteo; Falcarin, Paolo; Martín, Raúl; Martínez, Alvaro; Fernández, Rosario; Baladrón, Carlos; Aguiar, Javier; Carro, Belén.
Article title: Introducing the user to the service creation world: concepts for user centric service creation, personalization and notification
Year: 2007
Citation: Caetano, J. et al. (2007) 'Introducing the user to the service creation world: concepts for user centric service creation, personalization and notification.' In: *16th IST Mobile & Wireless Communications Summit*, Budapest, Hungary, July 2007.
Link to conference website: http://www.mobilesummit2007.org/

Introducing the user to the service creation world: concepts for user centric service creation, personalization and notification

Jorge Caetano (jorge.caetano@aveiro.nec.pt), Pedro Santos (pedro.santos@aveiro.nec.pt), Paulo Justino
(paulo.justino@aveiro.nec.pt),
NEC Portugal SA, Rua Mário Sacramento nº 177 3800-106, Aveiro, Portugal;
Laurent Walter Goix (laurentwalter.goix@telecomitalia.it),Paola Renditore (paola.renditore@telecomitalia.it),
Telecom Italia Lab, Via G. Reiss Romoli, 274 – 10148, Torino, Italy;
Matteo Demartini (matteo.demartini@m3s.it),
M3S srl, via Molo Cagni 16128 - Genova, Italy;
Paolo Falcarin (paolo.falcarin@polito.it),
Politecnico di Torino, Corso Duca Degli Abruzzi 24, Torino, Italy;
Raúl Martín (rmm375@tid.es), Alvaro Martínez (amr@tid.es), Rosario Fernández (rofg360@tid.es)
Telefonica I+D, C/. Emilio Vargas 6, Madrid, Spain;
Carlos Baladrón (cbalzor@ribera.tel.uva.es), Javier Aguiar (javagu@tel.uva.es), Belén Carro (belcar@tel.uva.es),
Universidad de Valladolid, Campus Miguel Delibes 47011, Valladolid, Spain;

Abstract—The "Web 2.0" feature that most permeates the nowadays web is "user-centricity". Now users are not only consumers of items (software, information, etc.), but also creators of those items. This paper intends to push this paradigm further, targeting mashups of telco and web services in a unique service environment where personalised services will be dynamically created and provisioned by end-users themselves, regardless of ambiance and location. The paper explains how user-centricity can be applied to the service creation world and in general to the overall service lifecycle process. It also describes the platform being implemented in the OPUCE project that captures this philosophy and will be submitted to end-user validation. Whilst focusing on intuitive editors for end-users to compose services, additional hints are provided about personalization and notification approaches to improve user centricity.

Index Terms—User-centric lifecycle, Service Creation, Context adaptation, Implicit Personalization

I. INTRODUCTION

O NE of the most impacting emerging trends in the Communication and Information Technologies (CIT) world is the "Web 2.0"[1]. This new paradigm for the World Wide Web promotes the usage of a bunch of innovative technologies and philosophies, such as semantics or peer-to-peer computing. But probably the "Web 2.0" feature that most permeates the nowadays web is "user-centricity": in the CIT world of the past, everything followed the developer-user approach. An expert (developer) created an item and a consumer (user) made use of it. In the Web 1.0 for instance, a limited group of individuals developed sites to be viewed by the rest of the users.

But "Web 2.0" paradigm has trashed out this asymmetrical approach and replaced it by a two sided, peer-to-peer approach. Now users are not only consumers of items (software, information, etc.), but also creators of those items. Sites as *wikipedia*, *flickr*, *youtube*, or the blog phenomenon show how the end user is now assuming the content creator role too, providing all kinds of multimedia content from expert knowledge and information to video and images.

The OPUCE (Open Platform for User-centric service Creation and Execution) project [2] was born in this framework to push this user-centricity paradigm further. Its aim is to bridge advances in networking, communication and information technologies services towards a unique service environment, where personalised services will be dynamically created and provisioned by end-users themselves, regardless of ambiance and location [3]. In other words, to build a platform allowing creation and execution of telco services by end-users themselves.

This further step in user-centricity, to allow the user to create not only static content, but also applications and services, seems to be one of the most interesting aspects in R&D nowadays, as proven by the increasing popularity of mashups (little applications that combine information from various web sites) and the birth of various environments for the intuitive non-expert creation of web-based information services, driven by the biggest and most successful companies of the CIT world, such as *Yahoo! Pipes* [4] or *Microsoft Popfly* [5]. These environments present graphical tools based on drag-and-drop interfaces which allow the user to create this

little information services/applications even without any computing knowledge.

The OPUCE platform aims to port this philosophy to the telco world. Where *Yahoo! Pipes* for instance allow only creation of simple information-based services, OPUCE will offer support for building a wide range of telco services. In addition, platform's tools for creation, deployment and execution automation allow operators and content providers to create their own high-quality premium services and deliver them in a short time with the aid of the built-in advertising/discovery system and community support tools.

The challenge for OPUCE is obviously to put a non-expert in the center of a Service Creation Environment (SCE). For traditional environments the service creator is an individual with a deep knowledge of information technologies, but for OPUCE the first assumption is that creators do not have any background at all either in programming languages or computing in general. Therefore the OPUCE platform should include a complete set of intuitive tools to allow easy composition and automated deployment, advertising, management and execution of services.

This paper explains how the OPUCE project applies the user-centricity philosophy to the user-creation world and describes how the elements of the OPUCE platform implement that philosophy. In more detail, it is explained:

- the meaning of Services and Base Services from OPUCE perspective,
- how user requirements are being treated,
- what user-centric service lifecycle is (which can be understood more deeply with a use case),
- which tools OPUCE platform will provide for users to create their own services,
- the concepts regarding implicit personalisation being used,
- how users can interact with the platform in order to either discover or share new services and being notified of existent ones that match with theirs interests.

Finally, it is important to clarify the meaning of "user-centricity" inside the scope of the OPUCE project in general and in this paper in particular, because it could sometimes be an ambiguous term. For OPUCE "user centricity" generally means that users are user creators themselves, so they are the "center" of the platform in the sense that they are both creators and consumers of services and everything evolves around them This should not be confused with the more generic meaning of the expression, like in [6], according to which, users are in the center, because the product is explicitly developed towards them, and as such effort is spent at design time in order to adapt the product to user requirements. Therefore, users are in the center of the "design/development phase". OPUCE also covers this second meaning of user-centricity, through trial activities in which a sample population experiments with prototypes of the platform in order to provide feedback for further requirements, but along this paper the term "user-centricity" will refer to the first meaning explained, unless stated otherwise.

II. User Centric Services

A. Services and Base Services

OPUCE services are intended to cover the whole spectrum of next-generation, yet convergent IT-telco services. Indeed the services targeted by the platform could be mashups of IT-based services, such as web services, information services, community services, etc. capturing "Web 2.0" APIs and paradigms, combined with typical telco-based services oriented towards communication like presence, messaging, call control, etc

The fundamental concept about OPUCE services is that they are build out of collection of base services that are provided by the platform (or by 3rd parties). Base services can be connected according to suitable composition rules in order to create more complex services.

Base services are functional units deployed by the platform owner (typically the service provider) or authorized 3rd parties, which can be used to create OPUCE services. When encapsulating telco resources or network capabilities, they can for example allow for sending SMS or IM, placing a phone call, setting up an audio conference, monitoring a friend's presence, etc.

Base services wrap a single or limited set of either telco or IT capabilities. This atomicity is needed both for user friendliness and for security reasons. From the users' point of view, a base service can be represented through a single and common sense paradigm, such as a phone, a book, a letter, etc. Dealing with network resources, instead, user usually needs technical skills in order to cope with many concepts, such as protocols, error conditions etc.: OPUCE goal is to ease the users' task and present them only with concepts they are familiar with (e.g. place a call, line is busy, drop the call etc.). On the other hand, telco operators need to be ensure that such capabilities are used in a controlled and safe way, e.g. to prevent abuses, to enforce billing and accounting etc. In this sense such atomic approach provides higher control and separation when operating services, especially when dealing with 3rd party components.

Finally this base service approach can be made recursive, i.e. complex services could be wrapped up as base services to be used as atomic blocks for other compositions. Base services available to the user community can thus be easily extended, both allowing 3rd parties (possibly even users) to contribute new base services (after a suitable conformance test has been carried out) and by creating new services out of certified components.

So, base services are the functional units needed to define services, but these "bricks" are not enough to build a wall: we also need a way to put base functionalities together. For this reason OPUCE introduces the concept of Generic Building Blocks that are the "glue" that helps connecting base services according to suitable composition rules. Examples of Generic

Building Blocks are the usual conditional statements from programming languages (e.g. Switch, If-Then-Else etc), but also blocks that are more specific to the user-centric, context-aware domain, e.g. context checker (that helps defining a flow according to the specific context where the user is).

This approach is reflected in the service creation process (detailed in the next section) where base services are boxes graphically displayed to the user and connected to compose a service.

B. Considering user requirements

A key principle in designing a user-centric platform for services is the gathering and understanding of end-user requirements and wishes. As reference playground, we have been analyzing deeply the evolution of the Internet towards the user-centric "Web 2.0" paradigm and tried to extend it to the telco world, where services are more complex entities, and to networks to be operated and meshed. The scouting of different approaches proposed to web users to create, share and manage their own contents (see YouTube, Flickr, etc) and more recently services (see Yahoo! Pipes, Ning, etc) provided us with a first input on upcoming trends in end-user service creation (which seems in extensive growth), besides giving some basic hints about composition and sharing paradigms. We then captured these trends to adapt them to the telco world, targeting "*fast service lifecycle management*" and "*user-generated services*" as the main concepts of our platform, trying to go well beyond current approaches of telco feature exposure such as SDKs or remote APIs.

Focusing on the European market, also led us to some further considerations in the native support of multiple cultures and languages when targeting any citizen, essential to the wide acceptance of the project. This requirement is particularly important in the context of service creation, but also has impact on the federation across service providers and their respective platform when users share services.

Although the platform is not intended for users to design their services automatically out of requirements, but through a graphical composition paradigm, we did consider user requirements – and feedback – for the design of the platform itself. Indeed, the platform intends to be evaluated, whenever first available, by user laboratories at service provider premises to provide feedback and suggest improvements along the project lifetime. In addition, some public events are planned to share our results and invite citizens to create and manage their own services.

C. User centric lifecycle

User-centric service lifecycle is intended as the process of enabling end-users (not technically skilled) at creating their own services and managing their lifecycle autonomously within the service provider's platform. Such process allows users to share their own services with a community, thus creating a powerful and self-increasing ecosystem.
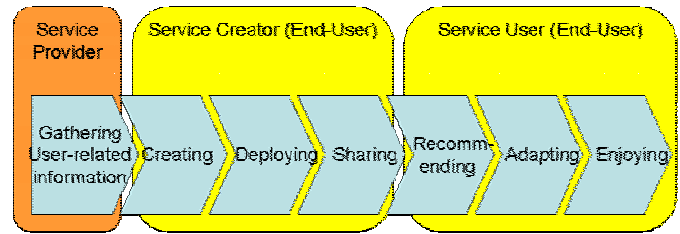

Fig. 1. Overview of the OPUCE approach to User Centric Services

Fig. 1 illustrates the approach we are considering for User Centric services. Three major roles are identified:

- the *service provider*, focused on learning information about end-users (both creators and final users), including context information, service usage statistics, preferences, etc. The service provider is typically also providing some base services to be composed and network resources to operate the platform and run services created by end-users,
- the *service creator*, an end-user creating her/his own composed services out of base services, also in charge of managing their lifecycle, such as deploying them or sharing and notifying them to her/his social network or interest groups,
- the *final service user*, an end-user of the platform interested in services of some kind that receives updates about newly available services and can enjoy executing them after some context-aware adaptation step.

Each step along the overall lifecycle is directly involving the end-user, as an entity either releasing some personal information (such as context, preferences, etc) to the service provider, or being a fundamental actor of the community, designing, managing and sharing services autonomously. This approach also intends to satisfy more passive users, who ultimately benefit of a wide range of services designed by other users that *per se* could better fit their needs, besides taking advantage of their context information for adapting at best their experience of such services in any situation.

To support this separation of roles still guaranteeing user centricity, we introduced the concept of "User Sphere", which refers to the set of information that describe the user as an individual having his own reality, such as her/his relationships, interests, well-known places, phone numbers, e-mail addresses, etc. Reference to the *user sphere* at each step of this lifecycle is essential to address relative information, whose actual value may vary across users. We use *$Me* as a generic notation referring to the final user's *user sphere*, e.g. *$Me.phone_number* references the final user's phone number.

To achieve such a lifecycle we are focusing our work on the service creation and service lifecycle management tools, relying upon existing service execution platforms that however need to be compatible with user-created services, for example for security or reliability reasons.

The service creation sub-system is responsible for providing the tools, both on the end-user device and at the service provider premises, for creating, managing and sharing

services, whilst the service lifecycle management sub-system aims at providing the deployment, monitoring and notification tools to quickly react and adapt services to the community's requirements and needs.

Such tools are essential in the field of user-centric platforms, which target demanding and ubiquitous users on the move, having their own *sphere,* together with numerous heterogeneous services, sometimes for a very short time.

As cross-cutting concerns, personalization and security are essential in enabling a heterogeneous community to design, share and operate services autonomously at each step of the overall lifecycle. In that sense, we are taking extra care of gathering and protecting user information (such as context, profile, preferences, etc) and validating user identities and permissions, especially when sharing services or executing another user's service.

III. USING OPUCE: A SIMPLE USE CASE

In this section it is presented a simple scenario where an OPUCE user has a communication need and decides to solve it using the OPUCE infrastructure. A short introduction to implicit personalization is also included in an attempt to explain the role of personalization in the user-centricity of the platform and its mapping to the proposed use case.

Personalization is the way to provide users with services tailored to their needs, preferences, interests and expertise. Implicit personalization is achieved by a process of adaptation that relies on the user context. The exact definition of context is a matter of discussion for authors. In OPUCE, context is considered as the situation of the user [7] and it is characterized by widely agreed primary components: location, identity, time and activity. A context, as complete as the available data, can be built from these basic variables. Modern techniques on collaborative filtering take care of complex mechanisms that model the user preferences in a way that contribute to the context abstraction (see section V). In the use case described here, the personalization aspects in OPUCE are based on the aforementioned primary components, showing the platform as a user centric one for the creation and execution of services. The implicit personalization is supported here from the very first step in the service life cycle: the creation. The use case describes the capability of the platform to enable a service creator to configure a service with her/his own context, to execute the service according to final user's context, (what is called in the project the user sphere, see section II) and finally to publish it so that other users can benefit from it.

OPUCE implements context aware service composition. This kind of composition is based on service components that are assembled by the service creator to build a user centric workflow, whose execution is modified during runtime according to context. These components are fed by the user's data in the way designed by the creator of the composition. OPUCE services are context aware applications supporting features [8] like:

- • *Presentation of information and services to the user*. Context in OPUCE is presented to the user as another type of information. This is implemented as the $Me concept explained in section II.
- • *Automatic execution of the service based on the user's context*. The context acquisition at execution time effectively determines the service performance towards the user.
- • *Resource discovery*. This is a twofold characteristic in OPUCE. Generally speaking, this feature is understood as the ability to inform users about services that may be of interest to them, according to the abstracted context. This particular feature is accomplished by the service advertising system of the platform. The second aspect commonly understood for resource discovery is the location and exploitation of resources relevant to the user's context. OPUCE platform also implements such system in its architecture.

Following it is explained a simple scenario covering the main OPUCE capabilities as a user centric platform, i.e. service creation, personalization, execution, and notification of services, we might call this use case "e-mail advanced reachability service":

"Bob is an executive that is waiting for an urgent e-mail. In fact, it is so urgent that he needs to read it at the moment it arrives. Bob cannot be the whole day long waiting for this e-mail in front of the computer, since he is sometimes at home, sometimes travelling and so on, so he would like to have a "system" that notifies him about his new e-mails everywhere he is, and that takes into account what he is doing: when he's driving, he can only attend phone calls, at work he can know about his new e-mails via a mail client and, when he is not working or driving, he would like to receive an SMS with the content of the e-mail he is waiting for.

Bob is registered as an OPUCE user, so he is capable to know whether there are already services that fit his need. He logs into the OPUCE portal and performs a service search with the intention of finding such a service. Maybe there is one, developed and shared by another OPUCE user that fulfils his requirements. In case there is none it is a minor problem, since the OPUCE platform offers him the possibility of creating one of his own. As he cannot find anything suitable, he decides to create a service from scratch. Another possibility could have been to retrieve a similar (and public) service and adapt it to his needs.

The first task he has to do for creating the service is to open the OPUCE Service Creation Environment (SCE) and create a new service. He searches the component repository trying to find some modules available for the service composition. Fortunately, he finds several base services and building blocks that seem to be useful:

- • E-Mail Base Service: this block allows sending e-mails. Moreover, it is able to detect that an email, which is compliant

with a preset criteria, has just been received.

• **SMS Base Service**: this block allows sending SMS messages. Moreover, it is able to detect when a new SMS, compliant with a preset criteria, has just been received.

• **Text-To-Speech Call Base Service**: this base service is able to call a specified phone number and, when the call is accepted, a text (a parameter for this component) is read out.

• **Generic building blocks**: as mentioned in II.A, Building blocks are the modules that, when used in the service composition, are able to drive the execution flow. In order to create this service, Bob needs, for example, switch blocks controlled by a context By using them, Bob can define different "execution paths" for his service depending on his actual context.

Once Bob has detected the "pieces" necessary for creating his service, there is only one thing left to do, connecting them. Next figure shows how the service composition, using the available blocks, would be.
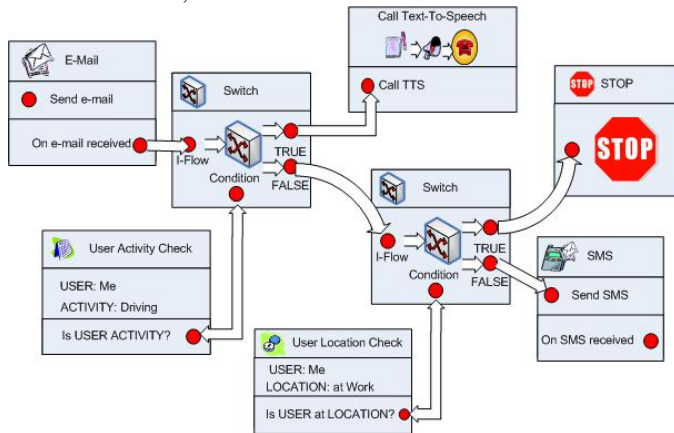

Fig. 2. Context aware service composition

After Bob has linked all the blocks in the composition, he saves it and checks its integrity.

As Bob thinks that his creation may be of interest to other users in such situation, he decides, after deploying it into the platform, to share it with every OPUCE user and especially with his colleague Anne. Therefore, he sets the service as public, marking it with some tags ("reachability", "e-mail", "SMS", "phone call"). These words will characterise the composition so that other users' lookups can locate his composition .Then, he selects explicitly Anne, an OPUCE user too, to receive a notification about this new available OPUCE service.

The notification about this new service will be delivered to Anne according to her service advertising preferences. She chose through the OPUCE user portal which notifications she is interested in, and when/how/where to receive them. In her profile, Anne set the SMS as the preferred means to receive notifications about new services when she is at home. So, when she arrives home, she will receive the notification about the service created by her colleague Bob. Other OPUCE users, subscribed for receiving notifications about new services that match their interests, would receive an advertisement too. Using the received notification, Anne, and the other notified

users, will be able to configure and execute/activate the service.

Coming back to Bob, at this moment, all he has to do is configuring and executing the service. Bob explores his services list in the OPUCE Portal and selects the one he has just created. He clicks on the link provided in the list and accesses the "GUI (Graphical User Interface)" of the service. Here, Bob, for example, configures who is the sender of the special e-mail he is waiting for. Only the e-mails sent by this sender will be processed by the service. Once the service is configured, he clicks on "Activate", and the "e-mail advanced reachability service" is running and waiting for such an important e-mail.

Finally Bob decides getting subscribed to advertisements on other users' potential reachability services, and by means of the OPUCE portal, enters his preferences. This time he had the opportunity to perform searches and compose flows, but who knows next time.

IV. USER SERVICE CREATION

A. General Concepts

Usually, service developers are highly qualified professionals with relevant skills, who could rely on suitable tools to be used to create services; they are also familiar with computer languages and technologies surrounding the creation process. With user-centric service creation, a paradigm shift occurs, because service creators are neither specialists nor professionals, but instead end-users that often have little knowledge or background in computer technologies. This role change imposes some constraints over the Service Creation Environment (SCE). Typical development environments would be very difficult to handle for average users, so a user-centered service creation process should ease the task of service creation as much as possible: suitable techniques to achieve this goal are user friendly graphical interfaces, wizards, composition helpers, guided editing etc., but others could also be leveraged, such as natural language descriptions.

The first requirements when designing a user oriented service creation environment are thus given by users' skills and user friendliness in its many aspects (e.g. appealing interfaces, ergonomics, etc). Another important suggestion comes from "Web 2.0" concepts of social networking and user communities: in other words, such hints can be leveraged to create a comfortable and stimulating environment that pushes users to create their services so that they feel this task is both useful and funny. On the provider side, this ecosystem of users and services can be a good field to collect feedbacks from users and on users themselves: in fact a lot of data about users behaviour, preferences and so on can be elicited both explicitly (e.g. promoting forums, blogs etc.) and implicitly (e.g. observing services users make or prefer). All these data can be exploited in order to improve and to refine both the platform functionalities and its architecture, to make it more suitable to users' needs.

1) Two editors for different needs

Dealing with user-centric service creation, the OPUCE platform definitely needs suitable tools for users to describe their services. From requirements collected during the analysis phase, we came up with the idea that service editing can be successfully covered by two different service editors facing different features characterizing both users and context. A user centric Service Creation Environment (SCE) for people on the move needs to take into account, besides user's skills, also user's context (such as device capabilities). Indeed users should be enabled to enjoy their new role of service creators using every device and access at hand, namely PCs, mobile phones, PDAs etc. This extended range of target devices imposes a requirement both on the SCE and on the resulting service definitions.

Following the observations above, the approach to user service creation outlined in this paper suggests two different service creation environments: a full feature editor and a simplified editor.

- The *full feature* editor supports a description language accepting a large number of composition rules, rich graphical user interface (possibly with a large footprint), advanced validation tools etc.
- The *simplified* editor is different from the above editor with respect to many features: less demanding user interface, simpler description language, restricted set of available operations, more guided creation/customization process, etc.

This distinction can be exploited in different and possibly complementary ways; e.g. according to pricing policies (users with low budget could only use the simplified editor, while premium users could use both), according to users' skills, according to mobility constraints, device capabilities etc.

A key concept of our approach is to allow service creators to perform "cross editing", i.e. to switch between the two editing modes when conditions are applicable. For this switch to be possible, both tools have to share the same design principles and take a consistent approach to the underlying service platform, linking with a centralized service repository in the background.

The basic idea behind these two editors is that each one shows OPUCE services according to what we might call its specific "service resolution": the more advanced the editor the higher the resolution. But the important thing is that both work on the same material, i.e. the service definition.

So, the first design principle underlying this approach is that the internal service description representation for both tools has to be the same. Out of this common internal representation, each tool will present its own picture and will allow the supported operations. Although the set of operations allowed by the two tools could be different, the resulting service definition has to remain consistent across modifications performed using the two different tools.

2) Dealing with user profiles: constraints and references

Talking about a user oriented service creation environment

two important issues have to be faced: dealing with user profiles and context and enforcing constraints on service compositions.

The first issue is covered by the concept of *User Sphere* mentioned in II.B; this concept describes the user's reality and can affect service behaviour, either implicitly or explicitly. I.e. values from the User Sphere can be explicitly referred to inside the service logic to define the desired service behaviour: e.g. if I receive a call while I'm away and the number of the caller is my mother's, then redirect the call to my mobile phone number otherwise play a message; in this case "my mother's phone number" is a value from the (final user) user's sphere that is used inside the service logic definition.

This means that a user-centric service creation environment needs a way to make explicit references to *User Sphere*, when creating a service. It is important to remind that people in the OPUCE community can be both final service users and service creators. This means that, when referring to a value from the *User Sphere* inside a service definition (e.g. an email address, a phone number, etc), there has to be suitable support to tell if that reference has to be resolved in the service creator space or in the final use space. This last kind of reference is needed to make service definitions more flexible and to ease the task of enabling other users to run services inside their own *User Sphere*. This approach also helps protecting users' private data, while keeping the ability to port services to other users. In fact users cannot reference directly other users' personal data and each reference is made through an alias for values inside the user sphere: e.g. if inside a service definition the service creator needs to send an SMS to another OPUCE user in his contact list, he can refer to $Me.contactList.OPUCEUser1.mobile and that will do. Of course OPUCEUser will be asked permissions for being added to the creator contactList, and this can make his references available (if he wants to) in an anonymous way (i.e. without showing the actual number).

The issue of enforcing constraints on service definitions relates to obtaining service definitions that are both correct and safe for the platform (e.g. that use resource correctly). This issue can be statically covered both by providing a guided composition process (e.g. helpers and composition assistants) and with semantic checks on the overall composition.

B. Full feature editor

The OPUCE platform will provide a full featured editor with a rich graphical user interface, aimed at rich terminal devices. The editor will be implemented as a Web resource accessible through the OPUCE Portal and users will be able to take part in the service creation process within their favourite web browser. "Web 2.0" technologies will increase the user experience, supporting the editor visual appearance and interactive behaviour.

The service creation tool must have a user friendly and appealing interface, but one of the main challenges is to offer

an intuitive, end-user oriented model for creating service logics. The model proposed in OPUCE is based on a composition paradigm: end user will be assembling base services which will be configured and connected to define the service flow.

The composition process can be summarized in the following three simple steps:

- Selection of base services;
- Configuration of properties;
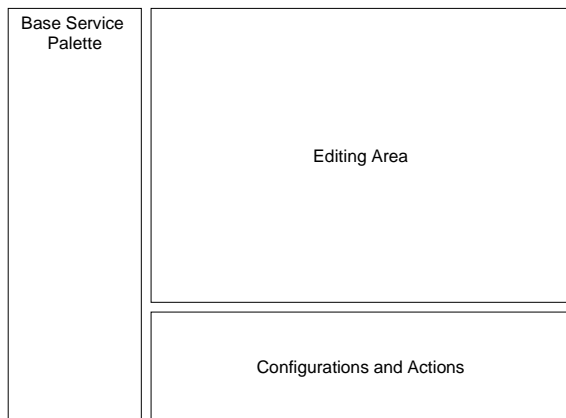- Linking of base services together.

Fig. 3.   Full Feature Editor Areas

Fig. 3 shows the different areas of the full featured editor.

The set of base services available for the service composer will be displayed in the Base Services Palette and categorized by concepts. In an end-user oriented service editor, the graphical representation of base services is very important. When users look at base services, their shape, icon and name must immediately recall the concepts the user is familiar with.

Base services are the boxes that can be selected from the Palette, dragged to the Editing Area and linked together in a workflow that represents the composed service flow (i.e. service logic). Selecting a base service in use in the composition, its properties are displayed in the configuration panel and the user can set the configurable parameters.

OPUCE specifically addresses telco services that are intrinsically event-based. Each base service can perform actions and manage events linking an event with an action, e.g. defines the action to be performed when the event occurs. Following the service flow it will be possible to "read out" services as a sequence of sentences like "*WHEN event.name THEN action.name*". Hence even the names of actions and events are important, because a proper choice may enable describing the created service compositions with meaningful sentences. For instance, supposing that a base service can manage the event "message is received" and another base service can perform the action "search telephone number", if such an event is linked with such an action, a composition which can be described with the following sentence is being defined: WHEN "message is received" THEN "search telephone number". In this way a service can be translated in a

sequence of sentences that is close to the natural language.
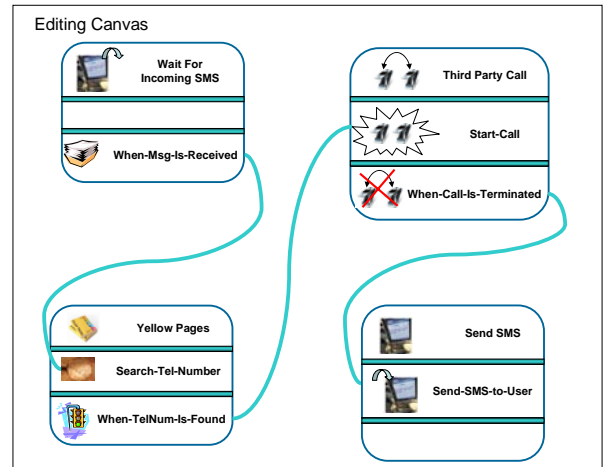

Fig. 4.   Service composition example

Fig. 4 shows an example of the "sentence composition".

The service composer doesn't need to worry about synchronous or asynchronous events: each base service is able to manage events in a common way, through specific handlers that allow drawing outgoing arcs to be linked to an action.

The full featured editor will also offer the possibility to create conditional branches in the service flow through a specific block that represents the intuitive construct "IF … THEN … ELSE" and increases the expressive power of the composition model.

Regarding the configuration of the base services taking part in the flow, users can configure properties in the following way:

- set a parameter to a specific and literal value (constant);
- link a parameter to a property of a previous base service;
- link a parameter to the user profile (or User Sphere).

Properties are global to base services and there is not an explicit binding between actions, events and properties. In this way users can define the data flow configuring base services and define the service flow linking events to actions, in two separate and pseudo-independent processes.

Furthermore, in the configuration phase the editor will help referencing user's profile data: the idea is that whenever the semantic of data needed by a base service refers to user's information, the editor will help/force to link the information to the "User Sphere". For example, if a base service has a property that requires a telephone number, the editor will suggest the *$Me.phone_number* value from the "User Sphere". Then the value will be automatically set to the telephone number of the specific user accessing the service, at run time.

C. Simplified Editor

The Simplified editor is aimed at Basic Terminals (limited graphical capabilities), typically mobile terminals, or Basic

Users (limited skills). These two targets share some common ground, which leads to four principles on which the simplified editor is based on: simplicity, minimum actions, maximum help and flexibility. All play together to achieve the goal.

For simplicity, a service composed by the basic editor is summarised to an ordered list of base services, a uni-dimensional organisation of base services in a chain, however maintaining unrestricted linking between any of the base services, that is, any base service can link to any other (one or more) base service in the chain. This approach is based on highlighting to the user the set of base services composing the service. The links between base services can optionally be showed (Fig 5) or not (Fig 6).

Fig. 5.   Chain with link display (wiring shows current linking)

Fig. 6.   Chain without link display

The simplicity of the chain concept makes it adaptable to text mode, using text lists, keeping the same base service chain concept and maintaining the same menu organisation so that it is intuitive for the user to migrate between different terminals (graphical and text based).
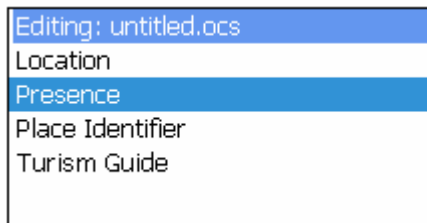

Fig. 7.   Chain in text mode becomes a list

For minimum actions, the user service creation processed is shortcutted as much as possible, by automating actions to the only available option or to the most likely available option. The automatically achieved configuration is likely (and hoped) to be the required one for a big majority of cases. This automatic configuration consists of both linking between base services and configuring some base services properties. Automatic linking of components is based on input/output types. Besides the direct linking of inputs/outputs of the same exact type, there can be also semantic adaptation which provides automatic insertion of adaptation components. For example if there is one base service that requires as input one location in "city name" format, and another that outputs one location in coordinated format, then an adaptation service is introduced that provides the adaptation between the two

formats. The adaptation services are just base regular base services that are identified (indexed) to be able to perform semantic adaptation.

Therefore most of the times the only actions required by the user are place the base services, confirm automatic achieved configuration and run. Nevertheless, there are, in some cases, some data that must always be entered by the user, e.g. the question text for a Poll base service.

For maximum help the Editor includes assisted editing to guide the user with the next steps required. An example of assisted editing is Colour Assisted Editing where colours are used to provide status indication, and to guide user to the next steps required. For example the colour of each base service in the chain can provide indication of how far is a base service from being completely connected within the service being composed:

• Red can indicate that some required linking is not complete, so user action is required;

• Yellow can indicate that all required linking is complete but some was done automatically by the editor and user confirmation is advisable;

• Green can indicate all linking is complete.

Fig. 8.   Chain using colours to indicate linking status

For flexibility, the user can always access the detailed configuration, although by default it is usually set automatically and not displayed. This includes the individual base service configuration and the linking between base services. That is, the creation detail resolution can be changed, either automatically based on user profile or manually by the user. The typical and topmost resolution is the list (chain) of base services without any configuration or linking detail.

Summarising, the aspects that distinguish the simplified editor from the full editor are: designed to run on mobile devices, simplified graphical interface, additional automation and additional assistance.

V.   TOWARDS IMPLICIT PERSONALISATION

Users' terminals (either fixed or mobile) can be used by network operators and service providers to learn more about user's data, habits and preferences and to take advantage of this information for service personalization.

While "explicit" service personalization typically requires users to configure services depending on their needs, implicit personalization relies on user information analysis and recommendation.

This information can be used to drive service personalisation in a more automated way, i.e. without user intervention, and can be divided into the following categories:

- *user profile*: typically persistent data, e.g. agenda, address book (social network), devices
- *user context*: typically more volatile/transient data, e.g. presence ,location, current device, etc
- *service usage*: which service is used by the user, in which context and with which parameters/values
- *device usage*: what is the user doing on his device. This information is linked to service usage, although it relates to the local usage of the device and its applications
- *user preferences*: about what the user likes/dislikes, based on direct feedback or input, or derived implicitly

The reference architecture used to gather exchange and provide such user information to the various tools involved in the user-centric service lifecycle is inspired from Mobilife's Context Management Framework [10]. Although main concepts are similar, the architecture has been generalized to a User Information Management Framework that could handle all types of user information, defining lightweight XML/HTTP interfaces for interactions over mobile networks.

In fact, some data are only available on the device and can be aggregated with information centrally available to mobile operators and service providers, for example to extract a higher-level description of the user context.

As perfect automation on deriving user *preferences* cannot be easily achieved from raw *context* data, recently collaborative filtering techniques have emerged as a trade-off between the mainstream personalisation approaches.

Collaborative filtering requires a user to rank a particular item, and the system can use these data to calculate similarities among groups of users, and then provide predictions on possible rank that could be given by a user to an item he has not evaluated yet. This technique assumes users have different preferences and once the system identifies that a user is similar to particular user's category, it can personalize services accordingly to this category.

The mechanism behind collaborative filtering systems requires that a large group of people's preferences are registered, in order to apply a similarity metric (like Pearson correlation coefficients) for identifying a subgroup of people whose preferences are similar.

The main problem with current collaborative filtering systems is the collection of preferences [11]. In fact, the system requires that many people could express their preferences about many items. Since the system only becomes useful after a number of opinions greater than a critical threshold have been collected, users will not be very motivated to express detailed preferences in the beginning stages, when the system is not able to recommend anything yet.

The great amount of user information hosted by Telecom operators can provide a precise picture of user behaviour and preferences, without requiring to the user any effort to set his/her own preferences.

Telecom operators can in fact avoid this start-up problem by collecting preferences that are implicit in people's actions and *usage* of services [12]. For example, users who often use a specific service implicitly express their preference for that service; *profile* information like user's contacts and social networks are also an important starting point for defining groups of similar users.

Using such techniques, implicit personalization typically happens at service execution time for adapting or personalizing services to match user needs and preferences. At service creation time, these techniques can also be used by the end-user service creation environment for identifying, among the set of services available in the OPUCE Repository, the set of most suitable services to be shown in the SCE, in order to be composed by the end user.

VI. SHARING, ADVERTISEMENT AND DISCOVERING

In an environment where items of interest (services in this case) are short lived, quickly changing and created in great numbers by users, it is of the uttermost importance to set up a system to allow their easy discovery. From the perspective of the creator, while it is possible that he wants to keep the new service completely private, usually he would like either to share it with members of his social network (a collaborative service to keep in touch with his friends, for instance) or to spread it through the entire community in order to let others use the same service or even, depending on the billing system, to get some revenues for its usage by other parties. This is *advertising*: the creator and/or the platform pushing notifications towards potentially interested end users.

From the end-user point of view, the necessity of an efficient discovery system appears either when he has a need for a service, but either he is not willing to spend time building a specific one or the service he needs is so complicated that it would represent a great effort from his side to create it from scratch.

In this case, he would prefer to take advantage of the work done by other members of the community and reuse some service picked among those already available, either "as is" or to modify it to better fit his preferences. This is *discovery*: the end user actively searching for services matching his needs.

In both cases the need for a system to put in touch the creator of a service with the potential end-users is obvious, so advertising and discovering are two different answers to that necessity.

In other environments of the telco world, the creator is usually a big entity capable of advertising campaigns using a plethora of means, but when creators are entities as small as an end-user, some other solution should be applied.

In the SOA (Service Oriented Architecture) paradigm the Universal Description Discovery and Integration (UDDI [13]) registry allows the publishing of service descriptions in order for them to be discovered. Metadata search allows a user to retrieve a list of suitable services, and this is an approach that could be reusable for user centric service creation.

The discovery system of the OPUCE platform includes a

mix of these two methods: a "push" method called advertising/sharing, in which a user sends notifications about a service to other users, and a "pull" method called browsing, in which a user searches for a specific service matching his needs using a set of metadata.

The "push" method (advertising/sharing) allows a service creator to use electronic means of advertising to notify other users of the platform potentially interested in the new service, using SMS/MMS, instant messaging, email or the internal OPUCE inbox. The advertising module of the platform takes the keywords and metadata defining a service and determines the most suitable list of potentially interested users, based on a semantic matching of service definition and user profile. Then, each of these target users is notified according to his notification preferences stated in their personal profile. These preferences let the user specify the preferred means and time of notification based on various context and presence variables, and also message filters to prevent spamming. This way, advertisements are non-intrusive, highly focused and efficient, and allow easy service sharing and advertising between small entities such as end users, but also allowing companies building complicated premium services to spread them properly to the community.

The "pull" method is on the contrary based on a query against the service repository. A semantic browser is coupled to this repository, allowing very narrow searches for very specific services if needed. Therefore, an end-user is capable of finding the exact services which matches his needs in terms of a wide list of parameters, such as bandwidth required or QoS availability, for instance, and not just using a bunch of keywords of metadata like in common browsers.

VII. CONCLUSION

In a world where technology complexity is growing, one of the big and constant challenges is too keep the technology reachable by the user while increasing the amount of features available. The key to keep an accessible environment is the filtering of both the available information and available options. That is the reason of the success of the internet search engines.

This is even more critical when the goal is to bring the user from a passive role to an active role in the system, making him an agent of the system, as is the case of OPUCE user centric creation environment.

Identified the new users centric role in the service creation and execution domain, this paper explains how OPUCE faces it and is contributing to facilitate and enhance their participation.

Tools provided by the platform to directly interact with users such as Service Creator Editors are targeted to deal with all kind of users, from the skilled ones in computer technologies to the others less skilled in that field. Associated with these tools there are others working in background to provide implicit personalisation and browsing/advertising.

The browsing/advertising system allows easy discovery of services. In an environment with items so dynamically evolving and created by small entities as end-users, this system provides non-intrusive and highly focused notification of services and semantic driven, narrow browsing, together with implicit personalization, which will derive user preferences from their usage of the platform helping them focusing on their real interests.

The OPUCE platform and particularly the concepts presented in this paper will enable the arising of new business models where 3rd parties, either individual users or small companies can play an important role.

REFERENCES

[1] Tim O'Reilly. What Is Web 2.0. O'Reilly Network. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
[2] Open Platform for User-centric service Creation and Execution, OPUCE, IST FP6 Integrated Project no. IST-034101, http://www.opuce.eu
[3] OPUCE project Web site: http://www.opuce.eu
[4] Yahoo! Pipes, http://pipes.yahoo.com
[5] Microsoft Popfly, http://www.popfly.ms
[6] W. K. Edwards, V. Bellotti, A. K. Dey, M. W. Newman "Stuck in the middle, : The challenges of user-centered design and evaluation for infrastructure", Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 297-304, 2003.
[7] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. Graphics, Visualization and Usability Center of Computing, Georgia Institute of Technology, Atlanta, GA, USA. {anind,abowd}@cc.gatech.edu}
[8] Pascoe, J. Adding Generic Contextual Capabilities to Wearable Computers. 2nd International Symposium on Wearable Computer, 1998
[9] Almeida, J.P.A., Baravaglio, A., Belaunde, M., Falcarin, P. & Kovacs, E. "Service Creation in the SPICE Service Platform." In: Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17), November 15-17, 2006, Heidelberg, Germany
[10] P. Floreen et al.., "Towards a Context Management Framework for MobiLife," Proc. 14th IST Mobile & Wireless Summit 2005, Dresden, Germany, June 2005.
[11] Shardanand U. and Maes (1995), Social information filtering: Algorithms for automating "word of mouth", Proceedings of CHI'95 -- Human Factors in Computing Systems
[12] Nichols D.M. (1998) "Implicit Rating and Filtering", Proc. DELOS Workshop on Filtering and Collaborative Filtering, Budapest, Hungary, Nov. 1997, ERCIM
[13] OASIS, UDDI Specifications, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm