A user-centric approach to service creation and delivery over next generation networks

Juan C. Yelmo, José M. del Álamo, Rubén Trapero, Yod-Samuel Martín *

Universidad Politécnica de Madrid, E.T.S.I. Telecomunicación, Av. Complutense 30, Ciudad Universitaria, Madrid 28040, Spain

ARTICLE INFO

Keywords:
User-generated services
Next generation networks (NGN)
Service-oriented architecture (SOA)
Mobile applications
Web 2.0

ABSTRACT

Next Generation Networks (NGN) provide Telecommunications operators with the possibility to share their resources and infrastructure, facilitate the interoperability with other networks, and simplify and unify the management, operation and maintenance of service offerings, thus enabling the fast and cost-effective creation of new personal, broadband ubiquitous services. Unfortunately, service creation over NGN is far from the success of service creation in the Web, especially when it comes to Web 2.0. This paper presents a novel approach to service creation and delivery, with a platform that opens to non-technically skilled users the possibility to create, manage and share their own convergent (NGN-based and Web-based) services. To this end, the business approach to user-generated services is analyzed and the technological bases supporting the proposal are explained.

1. Introduction

Telecommunications operators (telcos for short) gave up the monopoly on service creation long ago; hoping that third-party created services could hit upon user needs and thus increase subscribers' expenses on network usage. However, network capabilities are undergoing a process of commoditization, and operators desperately seek for the new "killer application" that may revitalize the languid figures of the Average Revenue Per User (ARPU) and provide a much needed added value to their networks. Agile creation approaches have arisen that try to reduce the time to market of new services. However, even they have proved to fail at keeping up with the pace of users' demands—but what if users themselves could create the new services provided over the operator's infrastructure?

Next Generation Networks (NGN) provide open and controlled interfaces over standard technologies that allow third parties to create new services that the underlying infrastructure is as well prepared to support. Those third parties that create services unfurl completely new business models where the relation between the different agents in the service delivery chain may well depart from the usual.

Drawing from the success of user-generated content (UGC) in the context of Web 2.0, we envision a rich telecom-based service ecosystem where tools assist users themselves in creating, sharing and using countless services, some of which will get to gain wide popularity and dissemination. Supported by these ideas, we present the Open Platform for User-centric service Creation and Execution (OPUCE) [1]. OPUCE has created a Telecommunications service delivery platform over a prototype of NGN that supports the rapid creation of new services by users. Based on a set of user-friendly service creation and management software tools, OPUCE tries to bring the successful model of Web 2.0 to the Telecommunications field, bridging NGN with the concept of user-generated services (UGS). For that, OPUCE tools handle the particulars of service lifecycle and hide them from users, whereas providing them with all the power of NGN at their hands to compose new services—their creativity being the only limit.

The remaining of the article shows how user-centric service platforms relying on NGN are viable from a technical perspective and how business models exist that may support them, providing OPUCE as a proof of concept. The next section presents an overview of the evolution of service creation in the Telecommunications domain, departing from the traditional closed models to the involvement of third parties and, eventually, towards a user-centric approach that improves as well the user experience. Section 3 introduces the business approach UGS and explains how a consistent business model may build on them. Section 4 describes our user-centric service creation and delivery platform, which validates the ideas introduced along the paper. Section 5 concludes the article with some global considerations about our major achievements and the expected future work.

* Corresponding author. Address: Universidad Politécnica de Madrid, Departamento de Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, C-215, Av. Complutense 30, Ciudad Universitaria, Madrid 28040, Spain. Tel.: +34 91 336 73 66x3028; fax: +34 91 336 73 33.

E-mail addresses: jcyelmo@dit.upm.es (J.C. Yelmo), jmdela@dit.upm.es (J.M. del Álamo), rubentb@dit.upm.es (R. Trapero), samuelm@dit.upm.es (Y.-S. Martín).

2. Opening up Telecommunications networks to collaboration

The efforts to open up Telecommunications networks to collaboration started with the standardization of the Intelligent Network (IN) [2], which decoupled the service development from the network infrastructure. Later on, the IN evolved to the Customized Applications for Mobile networks Enhanced Logic (CAMEL) to solve newer features and requirements of GSM (Global System for Mobile Communications) networks [3].

However, IN was not able to fulfil some of the requirements that new converged networks impose, such as a shorter time to market of new services and the development of value-added, network-independent services. To cover this gap, the International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) is leading the work towards a convergent approach for Telecommunications networks, which has been named Next Generation Network (NGN) [4].

NGN provides network operators with the possibility to share resources and infrastructure, facilitate the interoperability among networks, and simplify and unify the management, operation and maintenance of service offerings. This allows external parties to take advantage of Telecommunications functionalities in their own services. And as for the end-users, NGN allows having services which were precluded to mobile networks due to signalling delays and latencies, low throughput, etc. Therefore, end-users can now enjoy personal, broadband mobile services at any time and anywhere and can begin to expect the flexibility, scope, and service variety they have experienced through the Web. This process is accelerating, lowering further the barriers to entry for new service providers and introducing paradigms not previously seen in the Telecommunications ground.

2.1. Towards external service creation over next generation networks

One of the most salient characteristics of NGN is the convergence of different access and transport technologies around an all-IP (Internet Protocol) core network, broadening the range of services available over this common foundation. On the one hand, NGN services can be provided over any access network and device supporting IP, thus taking advantage of the respective capabilities available at each delivery situation and, more important, allowing users to seamlessly roam among different access contexts with a uniform service experience (concept known as Virtual Home Environment). On the other hand, any existent IP-based service can be easily provided over NGN; furthermore, new NGN services can integrate functionalities from external IP services in a transparent fashion. Since virtually any service can be provided over packet technologies such as IP, NGN can leverage on a varied and ever-increasing base of existing IP services.

Moreover, this reasoning can be extended to higher, application-level NGN layers, with the use of the Session Initiation Protocol (SIP) as a multi-purpose signalling protocol that enables flexible sessions of quite different media (voice, video, text, presence, gaming actions...) and service paradigms (streaming, conference and multiconference, event subscription...) over diverse transport protocols. SIP similarity to the most widespread web protocol, the Hypertext Transfer Protocol (HTTP), being human-readable and request-response-structured, eases the development of services that rely on it.

This is actually a two-way, mutual integration process—NGN service capabilities are also exposed allowing third-party, IP-based services to reuse them. Exposure through open Application Programming Interfaces (APIs) decouples the application layer from the underlying servers and protocols implementing NGN functions. IP-based NGN architectures such as the IP Multimedia Subsystem (IMS) [5] open the service creation process to third parties collaborating through different means such as the Session Initiation Protocol (SIP) Application Servers (AS), the IP Multimedia Serving Switching Function (IM-SSF) interfaces with CAMEL application servers, or the Open Service Access Service Capability Server (OSA-SCS). Operators expose service capabilities in the form of service enablers that employ technologies long used in the Internet world such as Web Services, thus opening service creation to new, non-traditional players coming from outside the Telecommunications arena. Notwithstanding, a secured, measurable, and controlled access to NGN capabilities maintains as well the network under control of the NGN operator.

Different initiatives have developed specifications to interface with service-enabling functionalities such as the Open Mobile Alliance (OMA) [6] and the JAIN Service Logic Execution Environment (JAIN SLEE or JSLEE) [7]. They rely on well-established paradigms such as object-oriented components, accommodating Telecommunications distinct features such as event-base interactions (which imply asynchronous interfaces) and transactional models.

These interfaces allow rapid and agile service creation and deployment, thus reducing the time to market of new services and further broadening the range of service creators. Third parties find easier to program new services that leverage on existing NGN capabilities, reusing them in a Service-Oriented Architecture (SOA). Convergent service providers can thus reuse bandwidth-intensive services requiring broadband capabilities (hi-fi audio, video, browsing, file sharing, massive multiplayer online role-playing games), new NGN-specific services (presence, availability, location), those enhancing traditional Telco services (VoIP call setup and control, SMS – Short Message Service, MMS – Multimedia Message Service) or non-Telco ones (instant messaging, IPTV – Internet Protocol Television).

Other NGN facets may also contribute to boost the richness new services created by external agents:

- Providers can personalize the service experience to user and access characteristics, reusing profitable personal identity and context information provided by the operator. Moreover, complex authorization and charging patterns may be issued in an easy way. Since users are permanently bearing their terminal devices, information is readily reported to NGN, which makes it available through convenient interfaces. All the same, user privacy is preserved, decoupling the exposed information from other network functions.
- Quality of Service (QoS) mechanisms and Service Level Agreement (SLA) foundations allow creating different service classes and guarantee service performance, whereas maintaining the network under control of the NGN operator.
- Automatic service deployment and provisioning can be made available through the intensive use of Operation Support Systems (OSS). Simpler interfaces and better OSS systems allow a dynamic service ecosystem, with continuous activations and retreats. They permit creating straightforward resource provisioning mechanisms that put the service lifecycle in the hands of the creator.
- Different providers can offer new services from scratch over diverse transports and access devices with no network impact, since exposure interfaces are transport independent.
- NGN provides scalable infrastructures that support this larger number of services, using decentralized architectures (e.g. peer-to-peer protocols such as SIP).

More available services may generate more revenues for the NGN operator. Most evident benefits come from a more intensive network usage, but novel business models may also arise—Section 3 thoroughly describes our proposal in this respect.

2.2. Enhancing the user experience through user-centric service creation

The service creation process can be extended to any user by leveraging on the same principles that have allowed opening NGN to external service providers. This allows users to acquire a new role, that of service producers, thus becoming *prosumers* (producers + consumers). This new paradigm initially appeared on the Internet with UGC, defined by the Organisation for Economic Co-operation and Development (OECD) as [8]: (a) content made publicly available on the Internet; (b) which reflects a certain amount of creative effort; and (c) which is created outside of professional routines and practices. *Prosumers* and UGC play a pivotal role in the so-called Web 2.0 applications [9].

User-centric service platforms, built around the needs and requirements of the end-users and leveraging on Web 2.0 technologies, draw the role of the *prosumers* to service creation, allowing non-expert individuals to create and share their own services. Mashup is the major concept behind the user-centric platforms: they are small applications made by the composition of two or more services and contents. Since homogeneous interfaces are available for NGN services, *prosumers* could combine in their mashups traditional Telecommunications functionalities with Web 2.0 services. In addition, service delivery can leverage on Web 2.0 supporting tools (recommendations, user profiles, user contacts and social networks) thanks to those regular interfaces. The main advantage of user-centric Telecommunications service platforms over the Internet ones is that the former possess some features that are particular to core Telecommunications networks.

User-centric service environments support the fast development and supply of innovative services enhancing the whole user experience. End-users can obtain their own, personalized, new services at their disposal, according to their needs and expectations. If it is done with an easy-to-use and intuitive service creation environment, the experience of service creation is proved to be quite pleasant and useful for *prosumers*: they can customize the services to their needs without waiting for others (traditional service providers) to do that. *Prosumers* thus create a networked service ecosystem whose value increases with each new user and service they create.

Discovery mechanisms à la 2.0, which are not limited to pull-type search, but also include information that is pushed to the users without their intervention, dramatically improve service *findability*. *Prosumers* can share and promote services they have created or simply they like, to their friends in their social network. Community generates information on services: rating them and thus making best services outstand, adding metadata at a zero-marginal cost through *folksonomies* (community generated tags), or simply sharing information and experiences through informal mechanisms (forums, wikis) that improve feedback on services. Service platforms themselves may issue recommendations to users based on their profile, context and usage history information.

When *prosumers* are allowed to share, rate and recommend services, a set of high value services will flourish and succeed from among the others. This has a positive effect to the creator's satisfaction because, eventually, the use of the platform becomes a desirable experience: Not only the creators enjoy their mashups, but also they see others doing the same, and thus their own confidence on the platform is increased, especially if they have tight social links with one another. Moreover, *prosumers* can feel useful to the community and develop a sense of prominence, especially when the services they create achieve wide success.

2.3. Related work

In this work we focus on telecom-oriented, IP-based, and user-centric service environments that enable new business models for NGN. These environments have recently begun to be introduced, and thus Telecommunications mashups, unlike Web-based mashups, are rather limited. Yahoo! Pipes [10] or Google App Engine [11] are examples of Web-based mashup tools. British Telecom (BT) Web21C was one of the earliest telecom-based examples.

BT Web21C allowed external developers and businesses to build their own applications using a small set of Telecommunications services, e.g. messaging, voice call and conference call. However, in this case the creators must be technically skilled, because the means chosen to compose services was through Java and .NET APIs. Following BT platform launch, other telecom operators began to provide developers with community portals and open APIs for network access e.g. Telefónica's Open movilforum [12], Vodafone's Betavine [13], Orange Partner [14] and AOL (America OnLine) Developer Network [15].

Microsoft and BT launched together the Connected Services Sandbox, which merged the Microsoft Connected Service Framework (focused on aggregating Microsoft services) and the aforementioned Web21C. The result was a Telco 2.0 solution where Telecommunications services can be part of a service mashup. However, this solution still lacked the variety of telco services required to achieve real value-added services, especially since third-party providers could not introduce their own services for composition. In our opinion, another drawback was the lack of an event-oriented model for service triggering, which prevented creating real telecom-oriented mashups that could be initiated by an SMS or a voice call. Recently, both the Microsoft Connected

Table 1
Service mashup platforms comparison.

Feature	Platform							
	Yahoo Pipes	Google App Engine	Ribbit (former BT Web21C)	Betavine	Microsoft Connected Services Sandbox	Open Movil Forum	Orange Partner	AOL Developer Network
IT (Information Technology) Services	YES	YES	YES	YES	YES	YES	YES	YES
Telco services								
Presence	NO	NO	YES	NO	YES	NO	YES	YES
Voice calls	NO	NO	YES	NO	YES	YES	YES	YES
Call control	NO	NO	YES	NO	YES	YES	YES	YES
Audio video conference	NO	NO	YES	NO	YES	NO	YES	NO
Event oriented	NO	NO	YES	NO	YES	YES	YES	NO
SMS	NO	NO	YES	YES	YES	YES	YES	NO
Positioning	NO	NO	YES	YES	YES	YES	YES	NO
Graphical mashup editor	YES	NO	NO	YES	YES	NO	NO	NO
Open to third parties	NO	YES	NO	YES	NO	NO	NO	NO
Context adaptation	NO	NO	NO	NO	NO	NO	NO	NO
Users' community	YES	YES	YES	YES	YES	YES	YES	YES

Services Sandbox and the standalone Web21C initiative were discontinued after Ribbit [16] has been acquired by BT, which we think is a step forward to enhancing the user experience.

Table 1 summarizes the aforementioned initiatives and compares them in terms of some features: the Internet- and telecom-oriented services available and the possibility for third parties to add more, visual tools to allow non-skilled users to create their services, etc.

3. Business approach to user-generated services

Communication markets are facing a challenging situation, as Internet competitors become a constant threat to telco operators' competences. This has made them focus their efforts on seizing clear market opportunities in order to find extremely successful services. On the other hand, the Web 2.0 philosophy offers a very attractive choice for telcos, as it capitalizes on user innovation [17] and the collective intelligence of user communities to create new knowledge and value, fostering creativity and social networks. This knowledge can be incorporated into the operators' innovation process and core business by means of open innovation processes [18].

User-centric service creation and delivery platforms leverage on these concepts and provide the means to incorporate user innovation to open innovation processes, thus enabling what we have called **open user innovation** for NGN services. Integrating the new services obviously benefits the platform provider business, since new innovative services will be commercialized. But what about the motivations and benefits for the other roles involved? Following, we thoroughly explain the business model we propose to sustain user-centric service creation over NGN.

3.1. Business model description

Three main **entities** (or agents) participate in our business model: *user*, *platform owner* and *third party provider*. The roles they can play are those of *platform provider*, *service provider*, *mashup producer* (or *creator*) and *mashup consumer*. Each role represents a different type of customer, which perceives value from the platform in different ways.

The **platform provider** supplies all the features and infrastructure that enable the creation, the management, the provision, the execution and the recommendation of mashups. It is also involved in the economic flows between the different members. Moreover, as any software platform, it provides a marketplace for service providers, producers and consumers to meet, acting as an intermediary and reducing the transaction costs for the two groups [19]. The **platform owner** usually plays the role of platform provider. We envision NGN operators as platform owners.

Service providers deliver their services to the platform so that they can be reused for composition. **Third-party providers** will usually provide specialized resources useful for composition. The services can come either from the Internet or from Network Capability Services (NCS) available in an NGN. If the platform owner is an NGN operator, it may also provide some NCS (e.g. send a SMS, set up a call, retrieve presence or information status, etc.) Service providers use the platform as a distribution channel for their products, thus gaining an intensive use of their resources at a low risk.

Mashup producers (or Creators) find that some functionality is needed and useful (for themselves or for any other member of the community) and have the necessary skills and tools to create a mashup from the set of services already available within the platform. Once the mashup is created and deployed onto the platform, the creator can share it so that it becomes public. Producers perceive value in two ways: mashups that simply fit their needs (for their own use) or mashups shared to get revenue (when they sell their mashups through the platform).

Finally, **mashup consumers** use and enjoy the mashups created by producers and made available through the platform provider. Mashup consumers may use a service that has been created by themselves or by a different entity; in that case, consumers need to have tools that ease the discovery of services useful for their specific needs. These tools (e.g. recommendation tools) are supplied by the platform provider that hosts the mashups.

The roles of both a producer and a consumer may be indistinctly played by **users**, who may act as producers of one mashup at one moment, and then as consumers or another mashup (or the same). Thus, the users in our model can be deemed as *prosumers*, as defined above. *Prosumers* can be profiled based on Forrester's Research Technographics surveys [20], which classify Web customers into six overlapping levels of participation according to how they use social technologies. Forrester's groups named creator and spectator represent the extremes of users that perform most of the time just one of the roles (producer or consumer). Creators make social content go: writing blogs, publishing their own Web pages and uploading the contents they create. Spectators consume social content created by others. Between both extremes, we find intermediate user profiles (contributors, members and collectors). We declare that our *prosumers* have an additional skill since they are in the context of an NGN: they are medium/advanced mobile users able to setup phone calls, send SMS and MMS, and use the device gadgets such as the camera or the multimedia player.

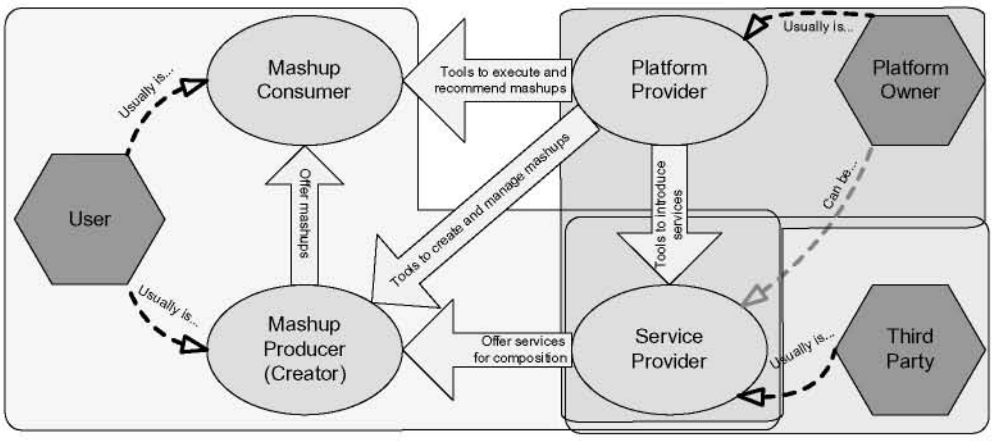

Mashup Consumer
Platform Provider
Platform Owner
User
Mashup Producer (Creator)
Service Provider
Third Party
Usually is...
Usually is...
Usually is...
Usually is...
Can be...
Tools to execute and recommend mashups
Offer mashups
Tools to create and manage mashups
Tools to introduce services
Offer services for composition

Fig. 1. Relations between entities (hexagons) and roles played by them (ovals) in the proposed business model.

The relations between the different entities and their roles in the business model are detailed in Fig. 1.

The traditional value chain for service delivery assumed a linear value flow and unidirectional relations from service providers to producers and finally to consumers, all supported by a platform. But in fact, there are many-to-many relations among the actors involved in our proposal that are not accurately described by a value chain. Therefore, we have defined our business model as a **value network** [21], which shows the complex web of direct and indirect ties between the various roles, all delivering value either to their immediate customer or to the end-consumer (Fig. 2). Here, the platform provider role is shown mediating between other roles intervening, which has been abstracted in the previous figure.

The value network helps to increase the supply of services on the providers' side and the delivery of mashups on the producers' side. Traditional network businesses base their value on the number of participants that take advantages of their features. A user-centric service creation and delivery platform is a network business as well, and thus it is important for its success to have as many members as possible. This will allow the flow of mashups being shared between producers and consumers to increase.

Nevertheless, users are the ones that play a fundamental role in our value network (playing as consumers and producers). First, consumers demand customized services, thus driving the value creation. Additionally, they can be linked together through social activities such as knowledge sharing and service recommendation, thus creating the most benefit for the people involved in the network. Knowledge can be shared among producers to create the best situations or opportunities, and to stimulate and improve open user innovation. In addition, service recommendation filters really valuable services from the huge amount of worthless (unprofitable) services, thus highlighting potential killer (profitable) services in the long tail. Services get consequently promoted at a minimum cost through mechanisms such as viral marketing, which increases product recognition through self-replicating processes that leverage on pre-existing social networks, thus cutting the marketing expenses of traditional telecom business models. On the consumer side, value networks enable ideas to flow into the market and provide the means to the people that need to hear them.

Two models are usually applied to software platforms in order to distribute the cost of software development: either the user pays or the developer pays. We think that user-centric service creation and delivery platforms will follow a similar path to PC operating systems (the user pays) with slight variations: unlike the operating system model, all consumers will be allowed to freely access the platform but they will have to pay for some (premium) services consumption (i.e. *freemium* approach [22]). Quite in the same way, service providers will be freely allowed to offer their resources as services, although they will be charged for the consumption of some specific features such as payment intermediation. Finally, producers will have to pay to use some special services in their compositions but they will be allowed to receive some revenues for the consumption of their services.

It is worth noting that the critical moment for the revenue generation mechanism, and also for the business model, is the launch of the platform: All the actors may be reluctant to invest in a new idea unless there is a substantial installed base of producers,

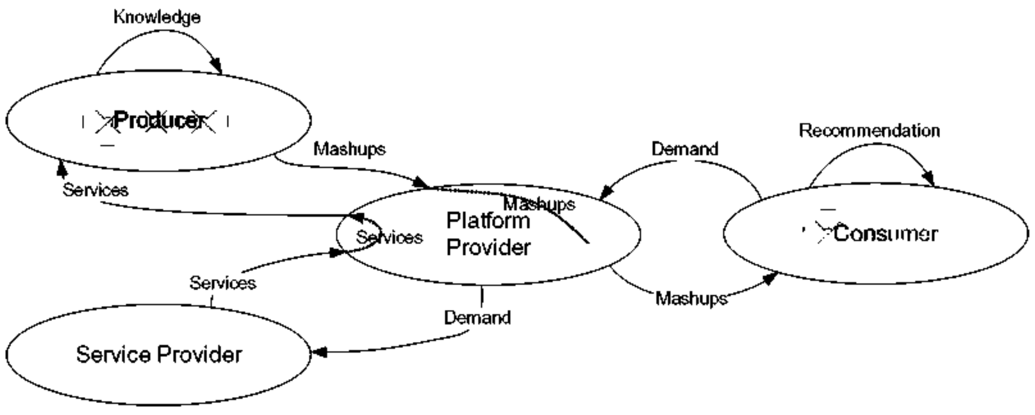


Fig. 2. User-centric platform value network.

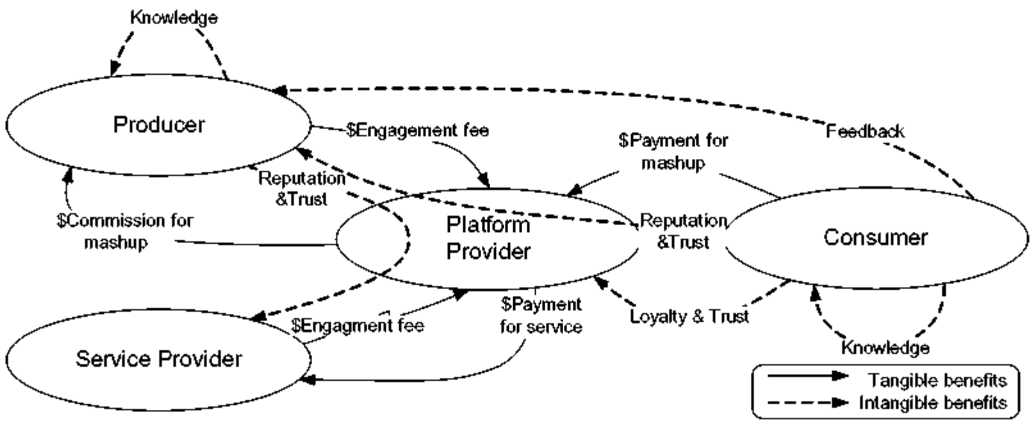


Fig. 3. Revenue flows between the different roles.

consumers and service providers. However, telecom operators have a privileged position to play the role of platform provider as they do already have this base of customers.

Fig. 3 shows the flows of tangible and intangible revenue between different roles.

Next section describes OPUCE, a user-centric service creation and delivery platform that allows validating the model we have introduced along the paper.

4. Open platform for user-centric service creation and execution

OPUCE platform is a user-centric service delivery platform that merges IT (Information Technologies) and NGN services and puts them at users' disposal for creating service mashups. OPUCE platform aims at leveraging a user-centric, dynamic ecosystem of user-created services over a NGN deployment. In OPUCE, users can create new telecom-oriented mashups by combining elementary services with an easy-to-use graphical editor. This user-friendly environment encourages users to create new services and promote them, so that others may use them. Moreover, users may easily steer the service lifecycle without dealing with the underlying intricacies of resource management. OPUCE takes advantage of the aforesaid openness that NGN features, ensuring a firm as well as seamless integration with NGN capabilities. As from the business perspective, OPUCE fits in the business model approach described above for UGS, thus benefiting users, platform and third party providers.

4.1. Designing a user-centric service platform

OPUCE platform fulfils three main requirements in the context of user-centric service creation and delivery:

(a) providing uncomplicated *user tools* for creating, managing, discovering, subscribing to and customizing mashups,
(b) starting and running the *execution flow* of the mashups users create and subscribe to, and
(c) managing *operation* and service lifecycle activities, with minimal user intervention (or none at all).

Next we present the three main architectural blocks responsible for each of these functionalities [23]. This architecture is aligned with that defined by OMA Service Provider Environment (OSPE) specification [24] –to which OPUCE has actively contributed–, especially regarding the lifecycle of services. Fig. 4 represents the platform architecture from the point of view of the integration of its main modules with NGN capabilities and Internet services.

The **Execution Environment (EE)** orchestrates the services that build up a mashup. OPUCE architecture follows an approach supported by Web Services. Services are exposed using Web Service Description Language (WSDL), whatever their implementation is (e.g. JSLEE, Parlay-X [25], etc.). Since composite services (mashups) are created by combining services, we have chosen Business Process Execution Language (BPEL) as the means of expressing their logic. The Execution Environment receives a BPEL description and invokes services accordingly, depending on the incoming events it receives, as will be explained later. It also receives information to schedule the activation and withdrawal of the mashup.

The **Portal** provides an interface between the users and the platform (see Fig. 5). Producers are provided with a set of tools that make up an environment to create new mashups and then share them. Creation in a desktop computer is supported by an AJAX (Asynchronous Javascript And XML) based editor which provides a friendly and intuitive interface. In mobile devices, creation is supported by a locally installed application. Consumers can use the portal to discover, subscribe to and customize new mashups, manage their personal information and share mashup-related informa-

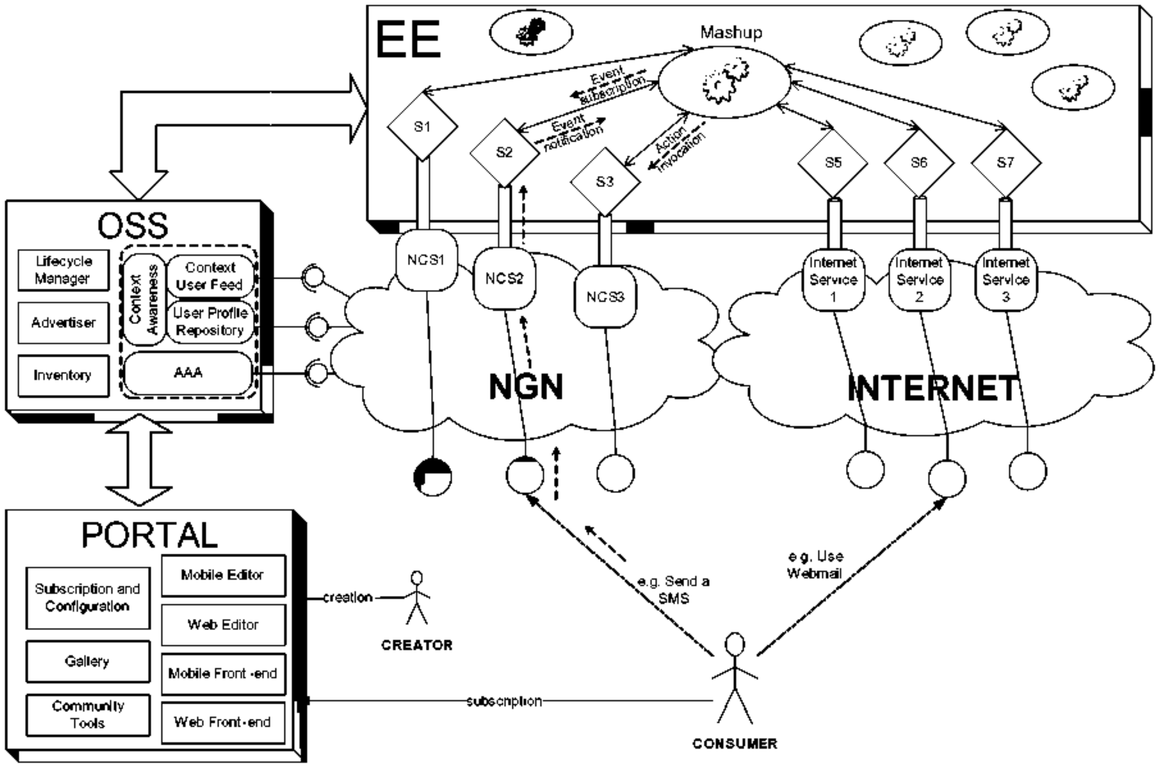


Fig. 4. High-level OPUCE platform architecture.

Welcome to OPUCE!

OPUCE is a new platform to compose, share and enjoy your own services.

You will be able to create mashups including not only Internet content, but also communication capabilities like calls, messaging, SMS, etc.

Try our tutorial video for more information or you can directly browse the online help for concrete doubts.

Fig. 5. OPUCE portal.

tion with their social network through the use of different community tools.

As a user-centric platform, OPUCE pays attention to several aspects that ensure the best user experience. Users are not usually skilled programmers, so they encounter straightforward user interface mechanisms that hide the complexities of underlying services. Besides, users are given a limited set of options (supported with help and tips) that reduce the possibility of errors and ease recovering from them, while still allowing users unroll the potential of their creativity in new mashups. Well-established metaphors and tips allow those users who seldom use the system to remind its operation between scattered sessions without needing to memorize it. Friendly user interfaces are provided to make the platform appealing; they also take into account the variety of user devices that are expected to access the platform in the context of NGN, paying attention both to desktop and mobile terminals. Finally, a good information architecture and interaction design ensures that users will succeed in achieving what they want and doing it fast, that is, in the efficacy and efficiency of the platform.

The **Operation Support Systems (OSS)** bridge the Portal, the EE and the underlying NGN implementation. The *Lifecycle Manager* is in charge of processes such as mashup deployment, scheduling and provisioning. The *Advertiser* manages mashups recommendations. An *Inventory* stores the descriptors of mashups and services available for composition. An *Authentication, Authorization and Accounting* (AAA) server manages security features. The *User Profile Repository* manages information about subscribers and the *Context User Feed* compiles users' context information. Finally, the *Context Awareness* module uses this information to dynamically adapt the mashup execution and personalize the presentation according to user context and preferences.

4.2. Mashups and services inside OPUCE

It is well known that events rule the behavior of a service in the Telco world. During a phone call, the communication is established with an event generated in the destination when the phone is picked up. And as for SMS, the mobile device keeps waiting for the event of an incoming message. In OPUCE, the EE handles events received from different modules: Lifecycle Manager (mashup lifecycle events), User Information Management (users profile events) and Context Awareness (context adaptation events). They are triggered anytime during the mashup execution and drive the mashup logic and thus the invocation of services.

Mashup triggering is also done on an event basis. When a consumer subscribes to a mashup, the OSS registers that subscription at the EE, so that the mashup execution is triggered when conditions apply. Then, when a subscriber accesses an NGN service interface (e.g., sending a premium SMS), the event generated during the interaction is forwarded through the NGN and the EE to the mashup that is listening for it. That very same event is used to determine the service execution path.

OPUCE has captured this event-oriented nature by declaring that all services can perform actions and may be affected by incoming events. As a result, a service composition is a directed graph that can be represented as sentences of type "*WHEN eventName THEN DO actionName*". We should reflect at this point on the nature of the execution process of OPUCE mashups and services. An OPUCE service neither represents a state nor an activity; it is rather an atomic component that offers a set of actions to be executed and defines a set of events it may fire. Consequently, an OPUCE mashup does not represent either a flow-chart, a workflow or a set of states through which execution moves; it is just *a mesh of services* that specifies the actions associated to each event that may be triggered. The appearance that a mashup is the consecutive execution of several services is just an illusion that only applies in specific cases. In general, a running instance of an OPUCE service may fire many events during its lifetime, and it may be running in parallel to other services appearing in the same mashup. From a global perspective, it would not make sense to denote a specific execution point at the mashup at each moment, since several services may be simultaneously running, may fire events at any time, and may be waiting for action invocations.

Each elemental service may also have its own internal state, which is exposed as a set of service properties. It is also possible to personalize some properties in the services with both static and dynamic values (evaluated at runtime). During the creation process it is possible to include context and user information. The user sphere concept (represented as $*me*) supports this. The set of properties that are visible through $*me* are managed by the editors by using virtual services during the composition. For instance, in an *EstablishPhoneCall* service the destination phone number can be obtained by different means:

- Directly obtained at the time of creation because the creator inserts a fixed value for this property.
- Obtained from the property of another service previously executed in the composition. For instance, it could be taken from the property *From* of a *ReceiveSMS* service.
- Extracted from a reference to an attribute of the user profile or context (i.e., $*me.privatePhoneNumber*). The creator does not know the specific value of this attribute, but it will be dynamically obtained by the platform during the execution time.
- Filled by the subscriber at subscription time (if the creator left if blank), when the mashup configuration is being personalized.

Finally, a mashup may also include control structures, presented to the creator as pseudo-service blocks, which enrich the possible links established between events and actions. Control structures include conditional links (*if*-alike), *fork* points (one event to many actions), blocking and non-blocking *join* points (many events to one action) and *termination* points (which stop the mashup).

Fig. 6 shows an NGN enabler available in the platform as a service, *GetLocation*, which retrieves a user's position. As any other service in OPUCE, it contains three types of elements to be set:

- Events:
    - *when-user-located* is triggered when the user's position has been correctly obtained.
    - *when-connection-fails* is triggered when there has been a problem while retrieving user's position.
- Properties:
    - *UserID* contains the reference to the user that is going to be located. In the figure below, the property has the value $*me.username*, which means that the final value will be taken from the subscriber profile through the $*me* parameters.

- Actions:
  - *get-location* fires the locating process of the concerned user.

Fig. 7 shows a composition that has been created using the OPUCE Editor. This mashup monitors a Gmail account; when an incoming mail arrives, it is converted to a voice message and a phone call is automatically issued to the receiver, who can listen to the incoming mail. In case the user is not available on the phone, the text of the mail content is sent by SMS. Same as any mashup, this one is defined by:

- a set of services (a *Gmail monitor*, a *Text-to-Speech converter* and an *SMS messaging* service, each one with its own parameters to be configured at creation time),
- a set of links between services, which map events to actions, and
- an initial event (the explicit start of the mashup from the Portal by the user).

As we have seen above, service composition in OPUCE involves several aspects that work together: services from vendors, their interfaces and many other aspects that are closely interrelated. OPUCE has created a model for service composition that describes all the aspects that characterize a mashup, both from the most common ones (*"What is the name of the mashup?", "Who is the creator?", "What is this mashup for?"*), to the more specific ones (*"Which services does this mashup use?", "How does it use them?", "What tasks are required at deployment time?"*). The services themselves are also described by using the OPUCE model for service composition, thus answering to the questions *"How can this component be combined with others?" "What operations can be done with this component?" or "How to configure this component to do what I expect it to do?"*

The OPUCE service composition model is based on a complete service specification that describes all the aspects that might be specified for a mashup or a service. The OPUCE service specification is generic and can be mapped to a specific mashup. When it happens, a linked set of XML (Extensible Markup Language) documents is generated, which represent the mashup or service in all its aspects. Each aspect is called a ***facet*** and is described in a different document. We formally define a facet as an abstraction over one of more service properties that provide a partial description of a service. Thus, each facet describes a focused task of service consumer, describing functional, non-functional, or management properties.

This faceted approach for mashup description allows adding new facets whenever it is necessary. Furthermore, as each facet can be described using a different language, it turns out to be a very modular and flexible way to describe services in general. As an example, in OPUCE we have included facets to describe, among others:

- the logic of the mashup, using BPEL – Business Process Execution Language;
- the interface of the mashup, using WSDL – Web Services Description Language;
- the provisioning to be done in the platform for each mashup, using SPML – Service Provisioning Markup Language;
- the semantic information related to the mashup or service, using OWL – Ontology Web Language; etc.

The generation of the facets is different if we refer to facets that describe a mashup or a service. If we refer to a mashup, facets are generated automatically at creation time when creators compose it using the graphical creation environment. Each facet is automatically generated according to the information introduced by creators when configuring and linking the modules that represent the services available. The creation environment interprets the information obtained from the composition (what modules are combined, what properties the creator has configured, etc.) and automatically generates each facet (for instance, the service logic facet can be generated according to the modules combined and how they are linked). When the mashup is inserted in the platform the facets are stored in a service inventory, awaiting to be extracted by the rest of the modules of the platform as long as they are needed (i.e., the EE will use the service logic facet when the mashup is executed).

Finally, if we refer to a service, the facets need to be created by the external vendors (service providers) when they are introduced to the platform. This process is not so automatic as with mashups, but OPUCE provides third parties with a service manager, which includes wizards and intuitive graphical menus that makes these tasks easier. This service manager, presented on Fig. 8, provides a set of graphical wizards that allow service developers from third parties edit the contents of the different facets of a service, load its implementation and make it ready at OPUCE platform (providing that the third party has been granted with sufficient privileges to do so). This service manager also uses typical IDE (Integrated Development Environment) metaphors so that developers find it friendly.

4.3. Executing mashups inside OPUCE

OPUCE EE is based on an orchestration layer that decouples the mashup logic from the heterogeneous runtimes where service implementations may be running. Three main elements are

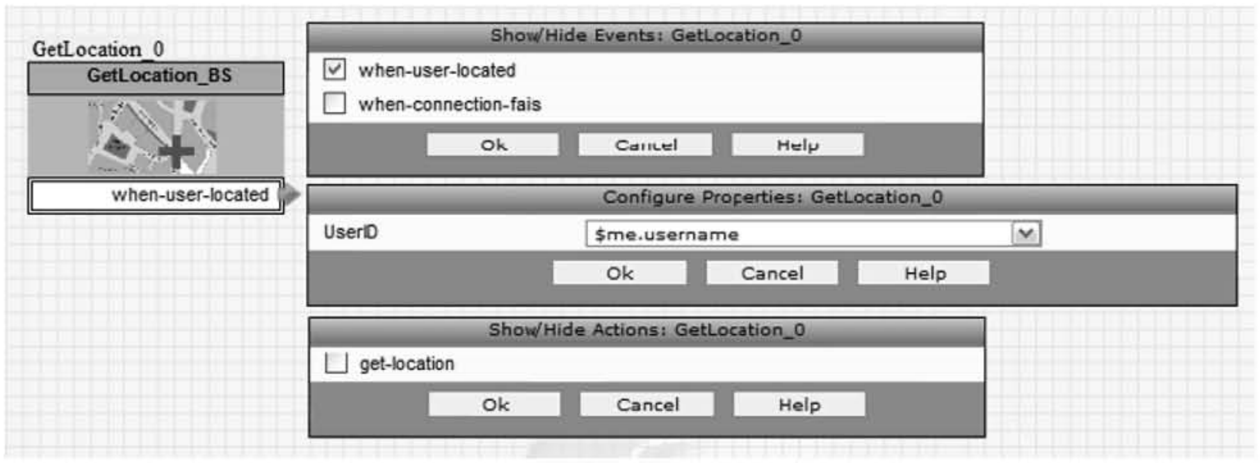


Fig. 6. GetLocation service.

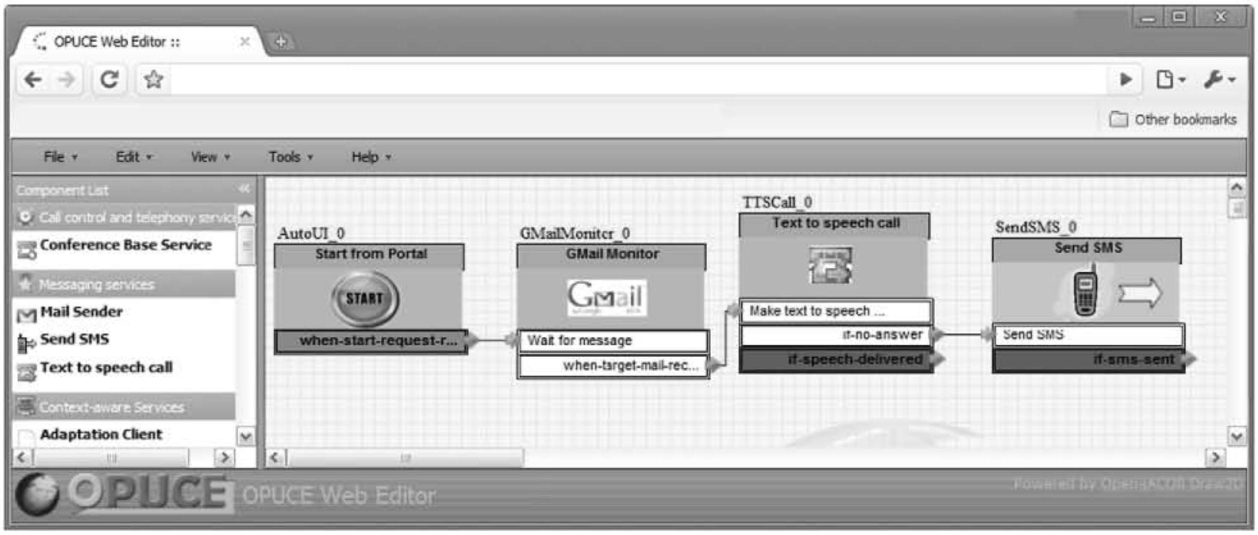


Fig. 7. Mashup example composed in the OPUCE editor.

distinguished inside the EE, as presented in Fig. 9: the *Service Runtimes*, the *Event Gateway* and the *Mashup Logic Engine*.

As above explained, each mashup is built of a set of services –basic building blocks that wrap and provide access to underlying telco and Internet capabilities, and encapsulate their functionalities behind a standardized interface. Each service stores its state as a set of properties, offers a set of actions (whose exact runtime behavior may depend on the service state) and may trigger asynchronous event notifications (which may also involve a change of the service state).

A **Service Runtime (SR)** is a container where these elemental services are running. An SR may contain several runtime instances of the same service, executing at the same time (one for each appearance of the service at each mashup session where it takes part). An SR exposes the services running on top of it, through Web Service interfaces that allow invoking service actions and accessing service state. Apart from that, no other constraints are placed on the specific SR technology used –in OPUCE, this is illustrated with two SRs developed: a JSLEE platform for telco services a Java EE container for Internet services.

The **Event Gateway (EG)** intermediates between the underlying services and the Mashup Logic Engine. EG is in charge of routing event notifications, retrieving consumer properties, and starting new mashups upon initial events. When a service triggers an event, it is notified to the EG. Then, the EG seeks the mashup instance corresponding to the source service instance, dynamically queries updated user context and profile information from OSS modules (since it may be necessary to fill action parameters); and sends the notification, the service state and the user information to the endpoint that is associated with the mashup instance at the Mashup Logic Engine (MLE).

When no mashup instance is associated to an event notification, this may be the case of an initial event that starts a mashup session. If the EG finds a mashup whose declared initial event matches the one received, then, it instantiates a new local mashup session and instructs the MLE to start a new mashup instance. Two sources of initial events may exist:

- in the case of one-shot mashups, its activation directly triggers the initial event;
- in continuous mashups (those that are repeatedly executed every time a trigger event occurs), a configured initial service fires a notification of an initial event each time a new mashup session must start.

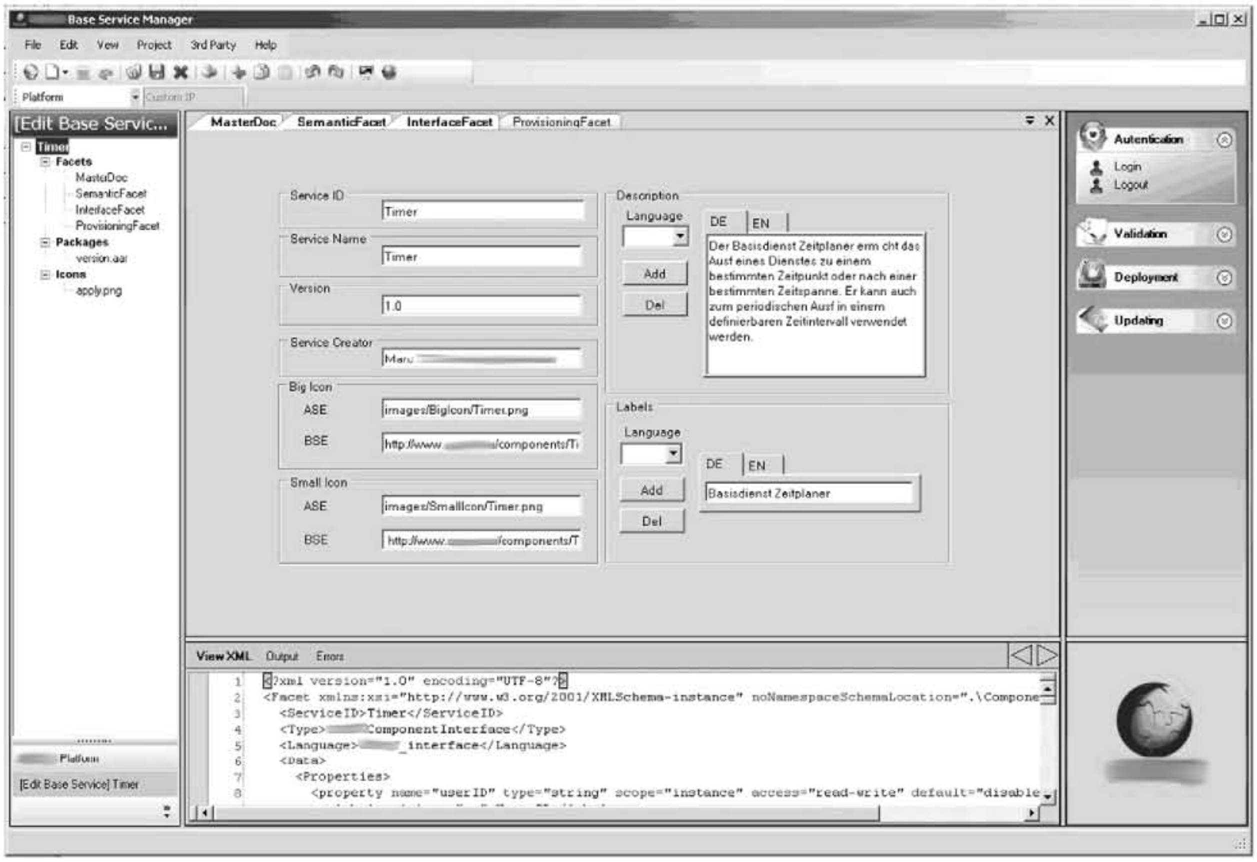
The **Mashup Logic Environment (MLE)** consists of a BPEL orchestrator that invokes service actions (through their Web Service interfaces) depending on the events received from the EG. As it has been explained, a mashup may be basically regarded as a mesh of service instances, together with a set of associations that link events to service actions. Accordingly, the function of the MLE is not to keep track of an execution workflow; it just rather applies those associations and invokes the actions corresponding to an event received.

The logic of a mashup is represented at the MLE by a BPEL process. When the EG receives an initial event, it instructs the MLE to create and start a new instance of the BPEL process associated to the corresponding mashup. This BPEL process will then represent the mashup instance at the MLE. The MLE exposes an endpoint for each mashup instance, where it will receive the associated events during its lifetime. Together with the event, the EG also supplies the updated values of the properties of the service instance that fired the event. The MLE implementation maintains a local copy of the state of each service instance, which it updates when an event is received, thus reducing the need to remotely access a service each time its properties might be needed.

The MLE implements a straightforward algorithm for each mashup instance. The MLE is waiting for events coming from the EG during most of the time (nonetheless, the mashup can be deemed to be active since one or more service instances are running meanwhile). At some point, a service instance fires an event, which is routed by the EG to the associated mashup endpoint at the MLE. Upon receiving an event, the MLE looks at the mashup process description and identifies the corresponding service actions (if any) associated to that specific event. After filling the action parameters, the MLE invokes the actions. Unless the event is terminal (which implies the termination of the BPEL process instance and the mashup instance), the MLE starts waiting again for events.

The EE just described corresponds to the initial implementation used in OPUCE project. However, several **improved standalone EE versions** have been also developed to enhance EE performance.

EEv1 is the EE above described. It is based on a BPEL engine that orchestrates service invocations and an Event Gateway (implemented as a JBoss Application Server service) that routes events received from services to the endpoint associated with each mashup at the BPEL engine. BPEL was chosen since it defines a WSDL-compatible orchestration engine that supports both synchronous and



Service ID: Timer
Service Name: Timer
Version: 1.0
Service Creator: Maru [redacted]
Big Icon
ASE: images/BigIcon/Timer.png
BSE: http://www.[redacted]/components/Ti
Small Icon
ASE: images/SmallIcon/Timer.png
BSE: http://www.[redacted]/components/T

Description
Language
Add
Del
DE EN
Der Basisdienst Zeitplaner erm cht das Ausf eines Dienstes zu einem bestimmten Zeitpunkt oder nach einer bestimmten Zeitspanne. Er kann auch zum periodischen Ausf in einem definierbaren Zeitintervall verwendet werden.

Labels
Language
Add
Del
DE EN
Basisdienst Zeitplaner

```
<?xml version="1.0" encoding="UTF-8"?>
<Facet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" noNamespaceSchemaLocation=".\Compone
  <ServiceID>Timer</ServiceID>
  <Type>[redacted]ComponentInterface</Type>
  <Language>[redacted]_interface</Language>
  <Data>
    <Properties>
      <property name="userID" type="string" scope="instance" access="read-write" default="disable
```

Fig. 8. User interface of the OPUCE service manager.

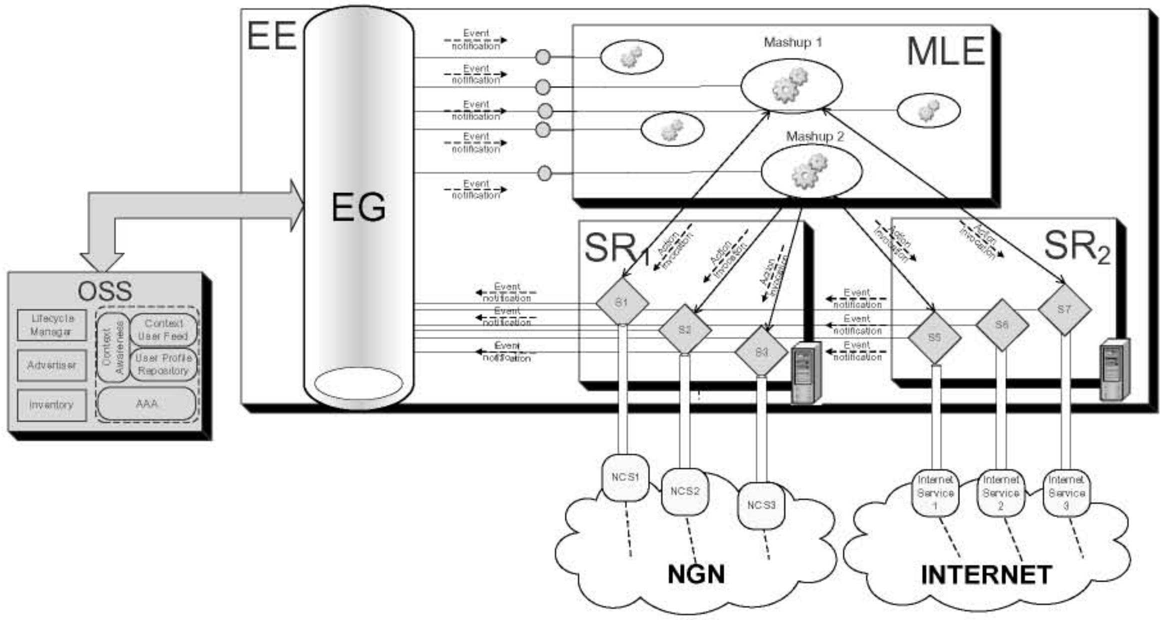


Fig. 9. Detailed architecture of the OPUCE execution environment.

asynchronous process definitions and includes tools to manage incoming events. EEv1 provides a centralized entity through which all events must pass, thus providing a point to enforce control policies. However, performance tests (presented below) showed poor results for this EE, which has led to develop alternative implementations of the EE, whose operation dramatically improved that of the initial version.

EEv2 replaces the EG and the BPEL engine with a single, much lighter, ad hoc orchestrator. As it collapses both elements into a single component, remote calls are reduced and thus latency is lowered –all this helped by the use of non-blocking calls. For each running mashup, EEv2 orchestrator keeps an in-memory, static routing table (instead of the BPEL process of EEv1) that associates service actions to events, in order to invoke the respective actions as needed. EEv2 orchestrator is completely stateless (all the session information travels along with each event, which might nonetheless increase the traffic), thus allowing several orchestrator instances run in parallel, for delegation and load balance. In addition, EEv2 orchestrator retrieves just the user information that is required at each moment to fill the action parameters. Notwithstanding, a centralized, orchestration management component must be kept separate in order to start/stop services.

EEv3 goes one step further and gets virtually rid of the orchestrator, substituting it for a choreography-oriented framework. Event routing information is not stored by a centralized orchestrator; it is instead split and distributed among the services. At deployment time, each service instance is provided with the rules to access the necessary parameters during execution and to route actions to its follow-up service. Then, when a service fires an event, the service itself uses those rules to map the event to another service action and invoke it (thus, notifications do not need to flow through a centralized module). Same as in EEv2, a centralized, choreography-management component is yet needed for service initiation and termination. EEv3 dramatically reduces the amount of remote calls that go back and forth (especially when different services run inside the same SR), and eliminates the bottleneck originated by the centralized orchestrator.

EEv4 is an integrated solution implemented on a JSLEE application framework, which provides an environment optimized for asynchronous processing. The mashup engine is implemented as a JSLEE Service Building Block (SBB) which acts as the orchestrator (to receive notifications and route events), together with a set of children SBBs to interact with services. Performance is improved because of the event-oriented nature of JSLEE, together with the use of a pool of objects to interact with services. In addition, a JSLEE Resource Adaptor supports communications with services and decouples the specific interface technology. This endows EEv4 with a versatile, loosely coupled design that enhances compatibility by allowing different service interfacing technologies, apart from Web Services –such as native JSLEE Services, which turn out to be faster as well.

Performance measurements were carried out on the different versions of EE, isolating this subsystem and preventing it from interacting with the rest of the platform. A minimal service *SleepSvc* was purposely created for performance tests: its only action (*Sleep*) simply waits for a fixed amount of time and then fires an event (*Timeout*). A mashup was defined with two instances of *SleepSvc*: at the beginning, the action *Sleep* is invoked on the first instance; later, when the event *Timeout* is eventually fired, the action *Sleep* is invoked on the second instance. Two kinds of measures were obtained:

- *Service latency*: using a zero timeout (so that all the delay is imputable to the EE), latency can be measured as the time difference between the invocation of the *Sleep* actions at both instances of *SleepSvc*. If a different timeout were used, it should be subtracted from the time measured. In practice, to reduce measurement deviations, more instances of *SleepSvc* were concatenated in the mashup and the total latency was divided by the number of steps.
- *Service scalability*: this measurement shows how the latency increases with the platform load, to test the EE behavior with many parallel running sessions. The EE is periodically penetrated with many parallel initial events that start many mashup instances. The number of running instances keeps increasing until becoming stable when the first timeouts start expiring. Then, the latency and the throughput (number of active sessions) are measured.

Tests were performed using three twin machines (one for the orchestrator and one for each service instance), each running an Ubuntu (Linux) operating system over an Intel Core Quad processor at 2.40 GHz and 4 GB of RAM. Next, the results are shown at Tables 2–4, where the latency and scalability measures are presented.

EEv1 confirmed the expected limitations, due to the amount of calls involved by the double-module (EG + MLE) solution, the poor performance of the Web Service stack and the heavy processes running at the BPEL engine. The performance values suggest that, even with hardware improvements, EEv1 would not fulfil the requirements of a production platform. Moreover, it is not probably due to the particular implementation of the BPEL engine used (ActiveBPEL), but it seems to be directly related to the choice of BPEL technology itself, which is not designed for high-throughput scenarios.

As predicted, EEv3 is the fastest solution because of the fewer remote calls and its distributed architecture (which spreads the load and consequently scales up very well).

EEv2 replaces the central orchestrator with a lightweight component with a solution in between centralization and distribution (session information is not stored at the orchestrator). It seems to be as scalable as EEv3, showing that a central orchestrator can be a employed, as long as it is lightweight.

Both EEv2 and EEv3 share a session-less behaviour, making them much lighter and requiring less memory. However, on a realistic scenario, the size of the messages could grow and require quite more network resources (since the size increases at each service, when it adds its own state). EEv3 would nonetheless be expected to perform better due to the lower number of remote calls.

EEv4 holds a much more complex internal logic at a central, stateful point of orchestration. Even putting aside the throughput, the general degradation is visibly worse than the EEv2 and EEv3. Nonetheless, when compared with EEv1, EEv4 gets the advantages inherent to JSLEE framework (better performance and flexibility).

In EEv4, native services are considerably faster, as expected, being able to fulfil telco requirements. Regarding the throughput; the internal mechanics of EEv4 for both types of services (native or external) are very similar. It should be noted that, when configured with a 3-s timeout, the scalability was even worse than with external services: other tests should be performed to understand if this is related to the specific implementation of the internal *SleepSvc* or if these results are merely due to setup reasons.

Security requirements are handled through multiple approaches: network security, access control, user privacy management and mashup sandboxing. First, the platform is executing in an operator-controlled environment where network protective measures must have been taken, thus guaranteeing that data inside the platform is not subject to counterfeit. For instance, all the modules in the portal exposing functionalities to final users are placed into a Demilitarized Zone (DMZ) so that users cannot access sensitive modules from the outside. Likewise, remote components running out of the platform boundaries (on the users' terminal or at third party premises) access the platform modules

Table 2
Results for the latency tests (ms).

	Latency (ms)
EEv1	35.0
EEv2	5.7
EEv3 (all services on the same machine)	2.4
EEv3 (services on different machines)	3.3
EEv4	8

Table 3
Results for the scalability tests: latency (ms).

Event generation rate (requests/s)	50	50	100	100	200	200
Timeout (s)	0	3	0	3	0	3
EEv1	(EEv1 can only handle a rate of about 10 requests/s before crashing.)					
EEv2	5.7	10.0	5.9	32.0	8.1	74.0
EEv3	3.3	6.0	5.1	15.0	5.2	144.0
EEv4 (Web Service-based services)	9.5	97.0	173.9	203.8	226.1	605.0
EEv4 (native JSLEE services)	0.1	10.0	0.2	329.6	0.2	3864.5

Table 4
Results for the scalability tests: throughput (session activations/s).

Event generation rate (requests/s)	50	50	100	100	200	200
Timeout (s)	0	3	0	3	0	3
EEv1	(EEv1 can only handle a rate of about 10 requests/s before crashing)					
EEv2	48.6	47.7	91.1	91.0	161.0	155.9
EEv3	48.4	48.0	91.0	88.4	166	115.8
EEv4 (Web Service-based services)	48.2	43.9	53.2	54.3	51.9	51.1
EEv4 (native JSLEE services)	47.9	45.5	92.0	71.2	166.7	107

through a Proxy Exposure Layer. Second, the AAA server is in charge of access control to the platform, preventing unauthorized users from accessing the platform at all, and deciding policies that restrict the privileges granted to each user for different actions. Authorized actions may be either platform-wide (e.g. registration), user-specific (e.g. profile access) or service- or mashup-specific (creation, deployment, activation, execution, etc.) Different user classes (basic, premium, trusted third parties, etc.) enjoy different authorization ranks. Creators themselves may specify the users authorized to use the mashup they create. AAA is also in charge of accounting platform usage, which allows monitoring access attempts. Third, user privacy is managed following some of the principles reflected at the European Data Protection Directive [26]: personal data is only collected or processed with user consent, for limited, explicit purposes and only disseminated as strictly required for the respective tasks. Fourth, the platform is virtualized at the server level and mashups are sand-boxed, allowing only access to resources according to SLAs. Finally, it should be remarked that different attacks have been emulated and succeeded at checking the validity of the security measurements implemented.

4.4. Integrating the platform with a NGN implementation

OPUCE is supported by an underlying NGN implementation based on IMS. The platform interfaces the IMS using mainly the Diameter [27] and the SIP protocols (Fig. 10).

Diameter consists of a base protocol that is complemented with so-called *Diameter Applications*, which are customizations or extensions to Diameter that suit a particular application in a given environment. The IMS uses Diameter in a number of interfaces (Sh, Cx, and Rf), although not all the interfaces use the same Diameter application. For example, the Sh interface links IMS application servers with the Home Subscriber Service (HSS), which is the master database of IMS storing user profiles, to perform operations related to subscribers data. In addition, the Cx interface links the Serving-Call Session Control Function (S-CSCF) and the Interrogating-Call Session Control Function (I-CSCF) with the HSS to perform authentication and authorization functions. Finally, the Rf interface is used by the SIP entities to send accounting information to a Charging Collection Function (CCF).

Within the OPUCE OSS, the AAA server uses the Cx interface to manage security features and the Rf interface to perform accounting operations. The *User Profile Repository* uses the Sh interface to retrieve information about subscribers. Additionally, the *Context User Feed* compiles user context information querying network enablers by means of different protocols e.g. XML Documents Management (XDM) [28] for group and list management, Mobile Location Protocol (MLP) [29] for location, SIP for Instance Messaging and Presence Leveraging Extensions (SIMPLE) [30] for presence and availability, etc. Finally, the *Context Awareness* module uses this information to support mashups adaptation.

The OPUCE EE also takes advantage of the event-oriented nature of IMS by means of the IMS application servers (Fig. 10), e.g. SIP Application Servers or OSA Service Capability Servers (OSA-SCS) [31]; and network enablers, e.g. based on OMA service enablers or JSLEE. Services enablers and services deployed in IMS application servers can be introduced to the OPUCE platform. They act as a link point to enable access to the mashup from IMS users and reach IMS users from a mashup, assigning to the OPUCE platform the role of a user-centric service delivery platform smoothly integrated with the underlying IMS. As a way of example, OPUCE has introduced services such as *SMSReceiver*, which includes SMS reception functionalities in mashups, and *Text2SpeechCall*, which allows setting up a phone call to a SIP user-agent and reproduce a locution.

4.5. Moving next-generation mashups into real action

Our proposal is supported by an underlying infrastructure where services are deployed and executed; in OPUCE, such infrastructure is an NGN implementation. We have used an open source implementation of IMS as our initial test-bed. Recently, OPUCE was launched as a beta version integrated with Open movilforum. The integration has been achieved by wrapping the APIs offered by Open movilforum as Web Services, so that they can be integrated in OPUCE as services. For example, our initial beta pilot includes APIs for locating mobile devices based on CellID information (*Localizame*), for using a network-hosted agenda that synchronizes with the mobile device local agenda (*Copiagenda*), for sending SMSs, and even for monitoring the reception of new SMSs; as well as others from the Internet world for sending email messages, monitoring e-mail accounts, reading content from an RSS (Really Simple Syndication) feed, etc. The network operator has assigned a premium number to our OPUCE test-bed, which, leveraging on the SMS monitoring service, results specially useful for creating services "triggered" by an SMS message sent by a user. For in-

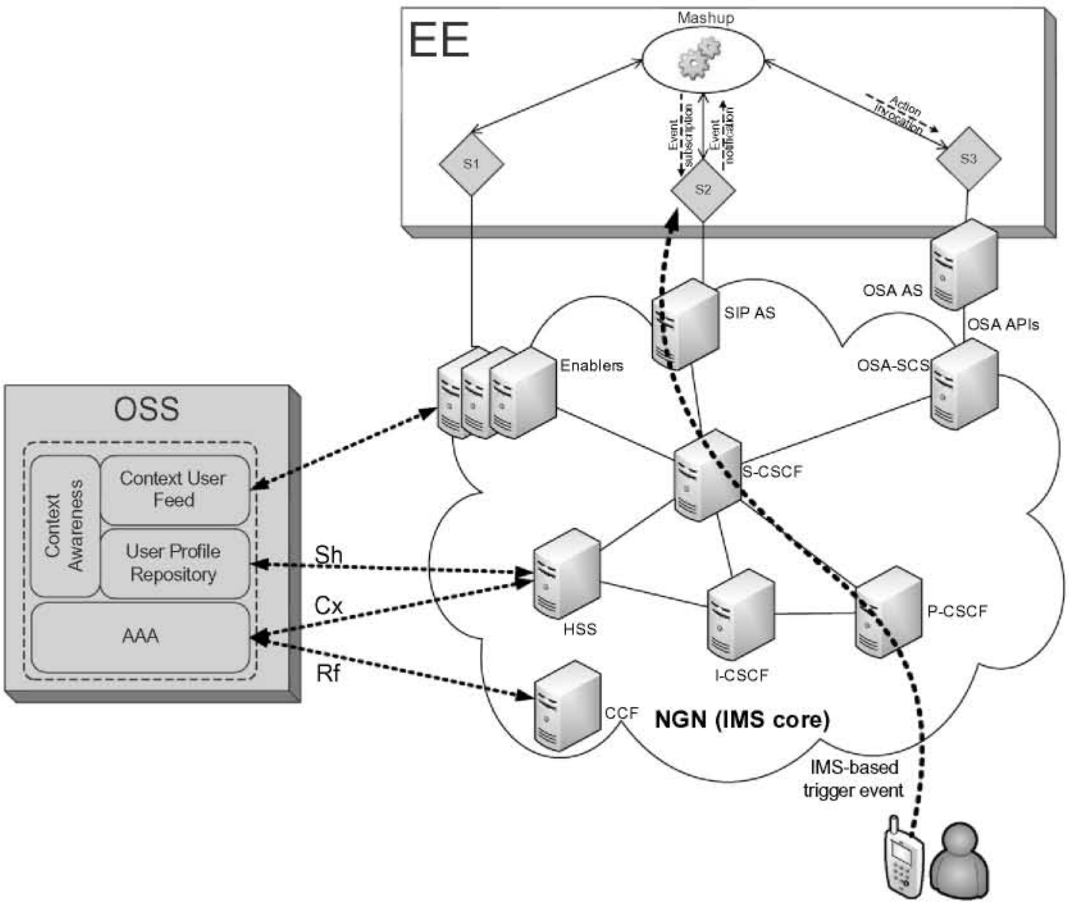


Fig. 10. OPUCE OSS and IMS interaction.

stance, the *EuroNewsSMS* mashup, which is activated when a user sends an SMS with the word "EuroNews", sends back an SMS containing the news that EuroNews (a European TV) publishes on its RSS.

The integration with the underlying NGN implementation has gone further and the authentication of users in OPUCE has been delegated to the NGN implementation, leveraging on existing telco assets and subscribers base: Mobile network subscribers are allowed to enjoy the OPUCE platform without needing either a previous registration or even an explicit login process. Besides, they are charged for the use of the premium services they use in their regular monthly bills.

Concerning the business model, free access and free use are granted for simpler services, mainly those coming from the Internet world. Premium and advanced services, normally the Telco ones, require a pay-per-use fee. In the aforementioned example, the mashup is formed by four different services: three of them are free (*SMS monitoring*, *RSS feed checker* and *RSS formatter*) and just the service for sending the SMS is charged to the subscriber's bill at the cost of a regular SMS.

5. Conclusions

In this paper we have presented the OPUCE platform, a user-centric approach for service creation and delivery over Next Generation Networks. Using OPUCE, non-technically skilled users are able to create, deploy, manage and share their own Telecommunications-based services so that others can subscribe, enjoy and recommend them. It deals with the application of user-centric service creation paradigms currently succeeding in the Internet, to the Telecommunications domain. We have described the technological and business context supporting our proposal, highlighting the benefits that the open user innovation approach provides to NGNs.

From a technical point of view, OPUCE provides an easy interface for users to combine a set of Telecommunications and Internet enablers to create new services that would fit their needs or the needs of a certain community. Advanced advertisement capabilities and dynamic context adaptation are other relevant functionalities provided in the OPUCE platform, thus benefiting from viral marketing and personalization opportunities. A suitable management of the user sphere (context and profile) allows a straightforward personalization of services.

Through the use of the introduced faceted service and mashup model, OPUCE provides a flexible framework to seamlessly cover functional and non-functional properties in the service and mashup management. In addition, OPUCE succeeds at providing an execution environment and lifecycle management functionalities that reduce human intervention during the lifetime of a mashup to a minimum (either by the operator staff, who would not be able otherwise to cope with the expected high amount of user generated mashups, or by the creators who lack the technical skills needed to operate mashups). Despite the initial approach for the Execution Environment did not cope with strict performance requirements expected, several improvements have been tested that demonstrate that a production-grade quality can be indeed achieved.

We have also put forward the role of this platform in the future approach of NGNs. A smart usage of a NGN deployment has validated all these ideas. First, OPUCE has been deployed over an IMS implementation and currently OPUCE also runs over Open movilforum. These NGN implementations provide an abstraction layer to lower network resources, and links OPUCE components with available services which are running either in the Internet or in a certain operator's domain.

Future work aims at completing the platform with more functionality, more services, and advanced operations and management capabilities. We will also deal with monetizing the relationship towards third parties in order to consolidate their role in the proposed business model. And as for users, enhanced user experience and usability are expected, which will enlarge the platform user base.

Acknowledgements

This work was supported by the IST European Integrated Project OPUCE (Open Platform for User-centric service Creation and Execution), 6th Framework Programme [Contract No. 34101]. We thank all OPUCE partners for their valuable comments and proposals aiming at improving the conceptual model.

References

[1] OPUCE Home Page, 2009, OPUCE Consortium, <http://www.opuce.eu/>.
[2] J.J. Garraham et al., Intelligent network overview, IEEE Commun. Mag. 31 (1993) 30–36.
[3] J. Zuidweg, Implementing Value-Added Telecom Services, Artech House Inc., London, 2006.
[4] ITU-T Y.2001 General Overview of NGN, International Telecommunications Union, Geneva, Switzerland, 2004.
[5] G. Camarillo, M.-A. García-Martín, The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds, Wiley, Chichester, England, 2004.
[6] The Open Mobile Alliance Home Page, 2009, <http://www.openmobilealliance.org/>.
[7] JCP, JSR 22 JAIN SLEE API Specification, The Java Community Process, 2004, <http://jcp.org/en/jsr/detail?id=22>.
[8] OECD DSTI/ICCP/IE(2006)7/FINAL Participative Web: User-Created Content, Organisation for Economic Co-operation and Development, Paris, France, 2007.
[9] T. O'Reilly, What is Web 2.0: Design patterns and business models for the next generation of software, O'Reilly Media Inc., 2005, <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
[10] Yahoo! Pipes Home Page, Yahoo, 2009, <http://pipes.yahoo.com/>.
[11] Google App Engine – Google Code, Google, 2009, <http://code.google.com/appengine/>.
[12] Open movilforum Home Page, Telefónica Internacional, 2009, <http://open.movilforum.com/>.
[13] Betavine Home Page, Vodafone Group, 2009, <http://www.betavine.net/>.
[14] Orange Partner Home Page, Orange, 2009, <http://www.orangepartner.com/>.
[15] AOL Developer Network Home Page, AOL, 2009, <http://dev.aol.com/>.
[16] Ribbit Home Page, 2009, <http://www.ribbit.com/>.
[17] W.H. Chesbrough, Open Innovation: The New Imperative for Creating And Profiting from Technology, Harvard Business School Press, Boston, MA, 2005.
[18] E. von Hippel, Democratizing Innovation, The MIT Press, Cambridge, MA, 2006.
[19] D.S. Evans, A. Hagiu, R. Schmalensee, Invisible Engines: How Software Platforms Drive Innovation and Transform Industries, The MIT Press, Cambridge, MA, 2006.
[20] Groundswell Home Page, Forrester Research, 2009, <http://www.forrester.com/Groundswell/profile_tool.html>.
[21] R.C. Basole, W.B. Rouse, Complexity of service value networks: conceptualization and empirical investigation, IBM Syst. J. 47 (1) (2008) 53–70.
[22] C. Anderson, Free: The Future of a Radical Price, Hyperion, New York, NY, 2009.
[23] J. Yu, P. Falcarin, J.M. Alamo, J. Sienel, Q.Z. Sheng, J.F. Mejia, A user-centric mobile service creation approach converging Telco and IT services, in: Proceedings of the 2009 Eighth international Conference on Mobile Business (June 27–28, 2009), ICMB, IEEE Computer Society, Washington, DC, 2009, pp. 238–242.
[24] OMA Service Provider Environment Architecture, Approved Version 1.0, Open Mobile Alliance, 22 October, 2009.
[25] Open Service Access (OSA); Parlay X web services; Part 1: Common, TS 29.199-01, version 9.0.0, 3rd Generation Partnership Project, 18-12-2009.
[26] Directive 2002/58/EC of the European Parliament and the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector, Official Journal of the European Union L 201, July 2002, pp. 37–47.
[27] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, Diameter Base Protocol, IETF RFC 3588, Internet Engineering Task Force (IETF), 2003, <http://tools.ietf.org/html/rfc3588>.
[28] XML Document Management (XDM) Specification, Approved Version 1.1, Open Mobile Alliance, 27 June, 2008.
[29] Enabler Release Definition for Location in SIP/IP Core, Candidate Version 1.0, Open Mobile Alliance, 18 August, 2009.
[30] Enabler Release Definition for OMA Presence SIMPLE, Candidate Version 2.0, Open Mobile Alliance, 17 September, 2009.
[31] Open Service Access (OSA) Application Programming Interface (API); Part 1: Overview, TS29.198-01, version 9.0.0, 3rd Generation Partnership Project, 18 December, 2009.