Service Creation and Self-management Mechanisms for Mobile Cloud Computing

Tatiana Aubonnet[1,2] and Noëmie Simoni[2]

[1] CNAM, CEDRIC, 292 rue Saint Martin, 75003, Paris
[2] Télécom Paristech, 46, rue Barrault, 75634, Paris Cedex 13
{tatiana.aubonnet,noemie.simoni}@telecom-paristech.fr

Abstract. Today, service providers need to develop competitive applications for a quick time-to-market to attract and retain end users. To facilitate the task of developers, we introduce a reference Service Creation Environment based on service component and self-management mechanisms. This environment uses a fairly high integration level using meta-modeling techniques and exchange formats: Meta-Object Facility (MOF), Extensible Markup Language (XML), OVF ++ (Open Virtualization Format). Our approach allows developers to design the basic service components based on Quality of Service (QoS), to build the service by composition, and to manage a mobile session by ubiquitous services and the Virtual Service Community.

Keywords: service components, ubiquitous services, self-management.

1 Introduction

Today, the Next Generation Networks (NGN) [3], [14] are considered to be a "user-centric" approach in the economic and technological world. Thanks to this new concept, the user can use any access to services and any terminal. However, with this freedom of access guaranteed by NGN, we have new challenges like the demand for quickly delivering new applications. Service providers need to develop competitive applications for a quick time-to-market to attract and retain end users.

Mobile Cloud Computing (MCC) is introduced as an integration of cloud computing into the mobile environment. Mobile Cloud Computing [1] brings new types of services and facilities for mobile users to take full advantage of cloud computing. It is important for the service provider to fulfill mobile users' expectations by monitoring their preferences and providing appropriate services to each user.

With each evolution, the new actors are trying to facilitate implementation. Thus, the key concepts such as (1) OSA (Open Service Architecture) PARLAY APIs, (2) enabled services in IP Multimedia Subsystem (IMS), (3) Service Creation Environment (SCE), and Service Logic Execution Environment (SLEE) in Intelligent Networks, (4) Web Services APIs, (5) and infrastructure APIs for Cloud Computing are proposed.

V. Tsaoussidis et al. (Eds.): WWIC 2013, LNCS 7889, pp. 43–55, 2013.

But important concepts are not taken into account in these environments such as *QoS* and *continuity of service* in a mobile context.

Can we solve or at least help to solve these concepts through a service creation platform? Our motivation is based on offering the maximum number of elements through this platform. This means that we must answer the following questions: What can we offer in term of construction *(by construction)*? What are the conditions to achieve maximum agility and flexibility?

In this paper, we present a software development environment for a Service Composition and Self-Management creation in Service-Oriented Architecture (SOA) context. We discuss fundamental concepts of the service creation platform and detail its main architectural components in order to have a reference framework.

The Service Components proposed are like actual automata that are accessible after any event, because, the properties of these services are not only those of SOA, namely: reusable, interoperable and autonomous, but also mutualizable, stateless, ubiquitous and self-managed. This architecture ensures cloud users a composition of Mobile Cloud Services in a seamless and dynamic way. A fairly high integration level has been reached using meta-modeling techniques (OMG standard MOF, XML, OVF++). We recommend taking the Service Components in the Referential of Service Creation Environment.

This paper is organized as follows. The related works for service creation is described in Section 2. Section 3 is devoted to our propositions for Workbench Architecture modules of the Service Creation Environment (Service Components, Links and Referential) for MCC including service component definition, Service Composition and piloting. Our propositions for Self-Management Mechanisms are presented in Section 4. Finally, in Section 5, we exhibit the advantages of our approach in a Mobile Cloud Computing use case.

2 Related Works

Works interests in the service creation field are the existing platforms in the Intelligent Network, Web Services, active networks and also Mobile Cloud Services gateway.

The Intelligent Network introduced two concepts: Service Creation Environment (SCE) [5] and a Service Logic Execution Environment (SLEE). These two concepts have been introduced with Intelligent Networks to ease and speed up the development and deployment of services [2]. An SCE is generally a graphical user-interface for developing services using predefined components, also called building blocks. A SLEE is the environment in which the services are actually executed.

This approach has been a precursor in separating the service components and of execution logic (enchainment).

Industry shows interest in Web Services due to their potential in facilitating seamless business-to-business or enterprise application integration [11], [13]. The contributions of Web Service creation are: improving the semantics of creation and collaboration services, introducing Web Services APIs and the decentralized vision for *the composition of services (choreography).*

The active network context enables customers to install and run their own customized services on a provider's network; a framework for service creation and management for active networks in telecom environments is defined in [4]. A key problem in this context is how on the one hand, a customer, who wants to run its service on the network, and on the other hand, the provider, who owns the network, interacts for the purpose of service installation and management. This problem has been addressed by introducing a service application based on Service Components and by network *virtualization* [7].

The paradigms of Service-Oriented Architecture, user mobile centric and Cloud Computing provide service-orientation for both software and the infrastructure services. In particular, the mobile cloud computing environment distributes and allocates IT resources according to an user's request, so there should be a study on technology that manages these resources and deals with effectively. C. Chapman and al. [6] discusses the implications of the on-demand cloud provisioning and architectural definition of distributed applications deployed in the cloud. It underlines especially that such an architecture has to be dynamically configured at runtime. It proposes language elements for describing software architectures, requirements towards the execution platforms, architectural constraints (e.g. concerning placement and collocation) and rules relating to applications elasticity. Concerning the requirements for the underlying IaaS platforms, both approaches are based on OVF (Open Virtualization Format) formalism.

We note that for the mobile context there is a missing feature, the continuity of service. Our motivation is to design, self managed components, reacting throughout the life cycle and during run time to trigger the load balancing based on user localization. That is why we integrate two new aspects and we present our two proposals: a service creation environment with the QoS (non-functional aspects of the service) integration and Self-Management Mechanisms to satisfy the continuity of service in MCC.

3 First Proposal: A Service Creation Environment

In this section, we present our propositions for a Service Creation Environment. The SCE showed in this paper can reside in the PaaS for Mobile Cloud Computing. Each component of the SCE consists of a consistent functionality the developer can manipulate for its creation of services. The Service Creation Environment is given in Figure 1. The architecture is based on a double separation: on one side the separation between *Service Components* and *Links*, and on the other side the separation between *Modeling* and *Piloting* (see Figure 1). SCE eases designing and provides a guide for each stage of development. The Service Components and Links are accessible through *Referentials* that are exported in the OpenCloudware Self-Service Portal. The Referential modules constitute a space for knowledge sharing on the basis of a common representation and they are built on standards allowing model storing, reuse and activation. In the context of the SCE workbench, the developer draws on the referential for services and links suited to their needs and defines the sequence and control desired.

For MCC, we can find for example a set of basic services such as Location Basic Service, Geographic-Location Basic Service, Presence Basic Service, etc. Location Basic Service (LBS) takes out the logic address (SIP URI) of a resource from the knowledge database. Geographic-Location Basic Service (Geo-LBS) takes out the geographical position (longitude, latitude) of a resource from the knowledge database. Presence Basic Service (PBS) takes out the state of the Resource from the knowledge database. It allows us to know if a resource is "Available", "Can be activated" or "Enabled". But we can also find component security service (authentication, authorization) or management services.

The role of Referential is to assure consistent communication between modeling and piloting tools during the service creation process. The interface between the Modeling, Piloting and the Referential is assured by an exchange of models based on the OVF++/XML standards.

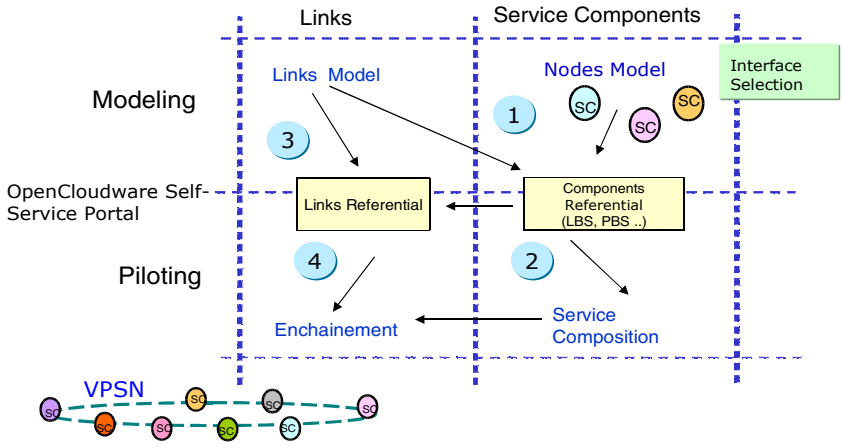

Fig. 1. Service Creation Environment (SCE)

This architecture also illustrates the separation between static (*Service Component and Link Component*) and dynamic (*Piloting*) aspects. This separation gave us a greater independence (static and dynamic) and consequently betters opportunities to evolve, as well as a more effective communication.

Compared to the SIB (Service Independent Block) approach [2], Service Component Architecture (SCA) is considered as a promising technique for service building. The main advantages of our approach are modeling the overall behavior of the system, and flexibility: our Service Components based on QoS are independent, modular, reusable and interoperable like the SOA components, but in more they are autonomous, stateless, ubiquitous and self-managed. These components form the basis for the definition of a Service Element Library called Components Referential in OpenCloudware Self-Service Portal.

The last important input for SCE is the “NLN” model (Node, Link, Network) [8], which provides an abstract image of the global system. Indeed, we divide the system into four decision views (visibility level): User, Service, Transport Network and Equipment. Each level consists of abstract nodes and links forming a subsystem called abstract network.

In this article the SCE is focalized by "Service Level" in where the nodes are the *Service Component (SC) and Links* (service interactions) and the *whole forming a Virtual Private Network Service (VPSN)*. In this section, the question is: how to define a service creation platform allowing the introduction of self-management. To reply, we present:

- Service Component (Section 3.1).
- Service Composition (Section 3.2).
- Piloting (Section 3.3)
- E2E Process (Section 3.4) (life cycle: think, build, run.)

3.1 Service Component

Service Component Properties

All the services are built according to SOA principles to allow the design of new services via a composition of service elements, a loose coupling between them and an exchange standard by standard interface. We present the set of important Properties to respect and follow in any design and deployment of SOA architecture.

Reuse: consists in designing services to make them more reusable. Reusing the same service in various business processes allows to reduce the development effort to meet new business needs.

Loose coupling: it is to limit the dependencies between services. It ensures agility that allows an SOA solution to adapt to changes in the IT world efficiently. Usually, these changes are due to external needs in the IT environment and therefore are difficult to be planned in advance. For this reason, loosely coupled relationships become an essential property for SOA. It is carried out by using the service contract, which allows the services to interact according to predefined parameters, and to share some knowledge while remaining independent from one another.

Stateless: it represents a service that does not keep state information and does not handle it. If a service maintains a state in the long-term, it will thus lose its property of loose coupling, its availability for other queries, as well as its potential to scalability. To do this, the services must be designed in a stateless way even if it means delegating state management to someone else. For a service to be stateless, its operations need to be designed to make stateless treatments, i.e. the treatment of an operation should not rely on information received during a previous invocation. It should be noted that the more intelligent the messages exchanged are, the more the services will stateless.

But specially, in addition to these SOA Properties, we introduce the *mutualization property* of the Service Component. It is designed to offer the same treatment to multiple users. This requires all component operations to be executed for all requesters. It is the contrary of the object, because the user can execute a selected method. This property will enable Ubiquitous Services Components to by make.

Among properties, we also introduce the four QoS criteria types: availability, reliability, delay, and capacity describe below in the Management Interface.

Service Component Interfaces

The Service Component is illustrated in Figure 2 with three Interfaces: Usage, Control, and Management.

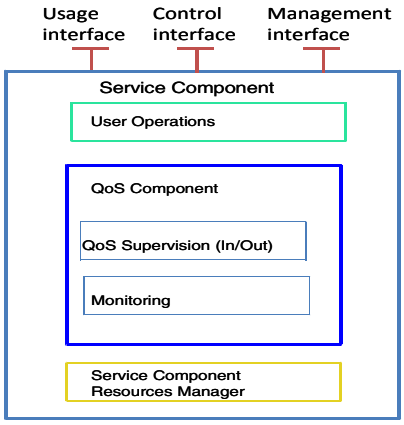

Fig. 2. Service Component

The *Usage Interface* is used to enable a basic service.
The *Control Interface* is used to reserve resources.
The *Management Interface* is represented by a QoS management interface integrated into each component composite. We use the same modeling to which we add the following models: *QoS generic model* and *interaction model*. The QoS model is generic and represents the behavior.
Four criteria have been proposed to describe a behavior:

(1) *Availability*: accessibility rate of the Service Component (accessibility rate).
(2) *Reliability*: running without alteration of information (error rate).
(3) *Delay*: time for request processing (response time).
(4) *Capacity*: maximum load of the Service Component (processing capacity).

Unlike some contexts, where just the throughput criterium as in UMTS is considered, or two criteria (throughput and fault tolerance) in the G1010, we found that these four criteria were necessary to describe the behavior of any function. We can say that these are sufficient because these criteria are independent of context. Indeed, they meet the following *transparencies*. The first is temporal and spatial transparency, i.e. to process and to transfer the information every time it is produced by the user (availability). The second is semantic transparency, i.e. the processing and the transfer is made without changing its content (reliability). The third is transparency in the distance, i.e. to process and transfer without changing the temporal relationship to the information generated (delay). Finally, transparency to the source, i.e. treat and transfer the information generated instantly (capacity).

These criteria are evaluated by Key Performance Indicators (KPI) which are measurable parameters. A KPI is a metric capturing some aspects of the performance

of one or more resources which is measured either directly, or could be defined in hierarchies. An example of KPI for the delay criteria is defining access time < 2s.

The interaction model specifies the autonomous degree of the distributed components, roles. The different roles are: *passive, active, reactive, proactive*. The developer of the service creation platform chooses to activate one of the roles. A reactive and proactive component describes the interaction behavior of an active component in which the object, which is highly-autonomous, does not simply act in response to its environment stimulus (changes). The proactive component can be used to maintain the QoS dynamically.

Figure 2 shows a service component where all interfaces are enabled. We obtain a composite component incorporating a QoS component and a control component. You can also associate a monitoring component. It is an autonomous Service Component (SC independent) satisfying stateless, and self-management, because it supervises the compliance of the delivered QoS. The OpenCloudware project adopted a Service Component approach based on QoS [9].

Remark : Link Component

In accordance with the <Node, Link, Network> abstract model, we see that the service is a resource composed of nodes (Service Components) and Links (network link, interconnection) of the same visibility level. We have assigned links to different network protocols (HTTP, SIP). The QoS Link will be integrated in the template OVF/XML.

3.2 Service Composition

In the Referential of the OpenCloudware Self-Service Portal (Figure 1) we have the service components. These are basic services.

To build the VPSN that will meet the customer request, we must identify and then select the right *VPSN nodes* (SC) - regarding their processing functionality and QoS- and establish the right *VPSN links* - regarding their communication capabilities and QoS that connect the selected nodes. Identifying the required VPSN nodes means to find which nodes are necessary to carry out the VPSN processing and thus the customer requested service functionalities. This is done thanks to service profiles that are created for each offered service by the service supplier. Among the identified nodes, we have to choose which one can carry out the requested behavior. This service node selection must be carried out in parallel with the link establishment to connect the selected nodes.

The VPSN uses the logic (the sequence) that specifies how the selected nodes must be connected according to the semantic of the requested service. The link establishment is thus carried out according to the VPSN logic.

The concept of "everything is a service" or "X-as-a-Service" allows the composition of a mobile application in the Cloud (Figure 3). The MCC application is composed by Service Components, for example SC1, SC2, SC3, SC4, SC5 *{security as a service authentication, security as a service authorization, service logic builder as a service, mobility as a service, and charging as a service, etc}*. But also, this application is composed by meta-services for example MSC1 *{monitoring as a service},* see Figure 3.

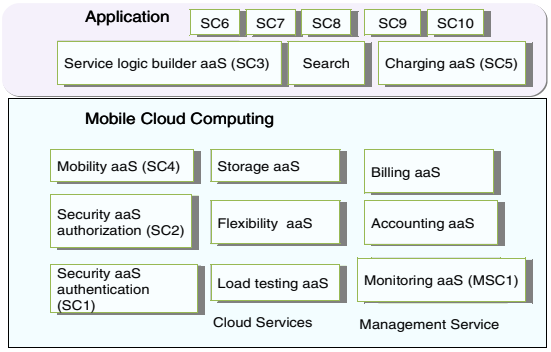

Fig. 3. Service Composition

The meta-services are the supervision services used for the management of each service component. These services manage the end-to-end application. Virtual resources will have a provisioning and will be instantiated.

3.3 Piloting: Event Driven Approach

As we have seen, the Virtual Private Service Network represents the selection of the services components and their sequencing. The link represents the interactions between services at the logical level. The entities of this level provide an application service. The VPSN is created at the time when the developer connects to the platform. When the developer connects, the platform knows the commercial offers it has subscribed to. These commercial offers are available in Referentials. A component of the platform, the "translator", knows the mapping between offers and the Services Components that provide this offer. The VPSN is created when all of SC corresponding to the commercial offers of the developer have been attached to the VPSN.

In our proposition, we favor an event-based approach. Our QoS management architecture is associated with a computing model that analyzes the QoS-based event and consequently adapts the VPSN configuration. Its QoS-component detects a QoS degradation and notifies the cloud management system. This latter monitors the event.

In order to support our event-driven approach, we create an Events Manager on the platform. The latter receives the notification (QoS degradation event) and matches this event to a specific action.

More precisely, the developer has at his disposal a decision table where for each event he indicates the next SC. The SC may be an application component or management component. Normally, a management action is a notification sent inside the service platform to a management service that is already subscribed to this type of events. In our example, a specific manager is subscribed to QoS degradation event. Hence, the Events Manager sends to the right manager, the right notification, to inform it that he is now able to start its control action and management process on the service layer. For the QoS degradation, the VSC service (that is explained later) is invoked.

For the mobility we have “spatial mobility event”.

To illustrate the QoS integration in Service Creation Environment, in the following section we describe an E2E Process of the life cycle.

3.4 End to End Process

We propose to take into account the four processes of the life cycle: design, (THINK), Deployment (BUILD), Provisioning, and Operating (RUN), see Figure 4.

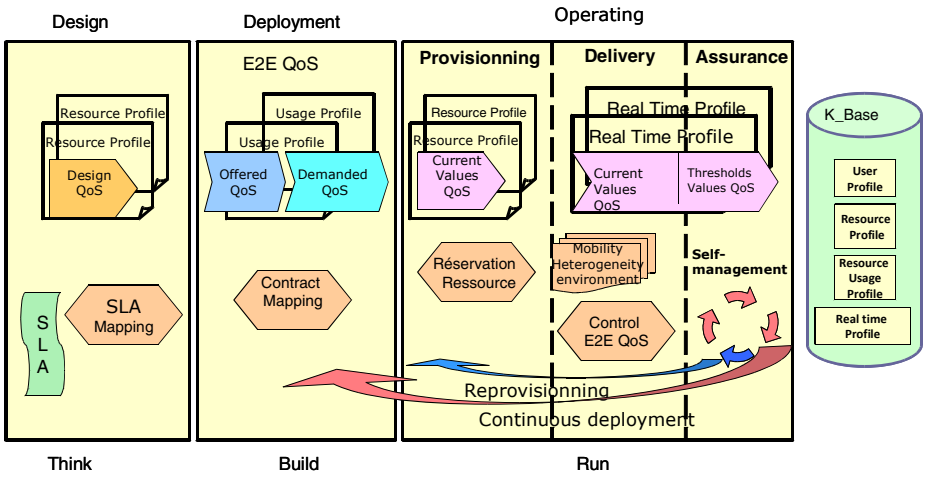

Fig. 4. End to End Process

The QoS component managed the behavior of the Service Component. Figure 4 shows the functions that it performs at each life cycle phase. The information is contained in appropriate profiles that are located in the referential.

In the *design phase* the QoS is evaluated as a design value in the service profile (instance of “Resource Profile”).

In the *deployment phase* we can install the component in the right place by comparing the values that we find in the profile called “Usage Profile”: the QoS offered and QoS requested.

In the *provisioning phase*, the QoS component is evaluated according to resource availability, to determine whether answer the query with the required QoS.

In the *operating phase*, the QoS component monitors continuously the availability of the Service Component resources. If the availability of the resources does not fulfill the QoS, then the QoS component emits an event of “out_contract” type. It uses its Management Interface. This event is received by the Event Manager of *MCC session* which will then seek SC replacement by Ubiquitous Services.

In the following section we propose Self-Management, i.e., Self-Management by the Ubiquitous Services compatible with the objectives in the Mobile Cloud Computing, and Virtual Service Community creation.

4 Second Proposal: Self-management Mechanisms

We introduce *Self-Management Mechanisms*. The management of the QoS is possible thanks to reactive and proactive role of the Service Component.

In our research, we rely on the objectives of the OpenCloudware project which will incorporate: (1) the components of modeling (THINK), (2) the development and production of applications (BUILD), (3) a PaaS platform compatible with multi-IaaS (RUN) for deployment, orchestration, performance testing, and self-management, (4) and the provisioning of virtualized multi-tier applications of type JavaEE.

During the cloud user session each Service Component is self-managed. This ensures automation and decentralization of service control and management. In this section, the question is: how to create and manage a MCC session based on QoS between a user and a service provider? To answer this, we present:

- Ubiquitous Service Creation (Section 4.1).
- Virtual Service Community Creation (Section 4.2)

4.1 Ubiquitous Service Creation

To enable the replacement of a service, we define the concept of Ubiquitous Service. *Ubiquitous Service* is a Service Component offering the same functionality and the same QoS depending on the model.

Ubiquitous Service will ensure the replacement of a degraded service by an equivalent service to maintain the QoS contract during the different movements. Thus, the mobility of service will result in the replacement of a service element by a Ubiquitous Service element. This replacement is done without breaking session thanks to the concept of VSC. The Virtual Service Community (VSC) contains a set of ubiquitous SCs (section 4.2).

4.2 Virtual Service Community Creation

To guarantee a continuous end-user's session, the Service Creation Platform proposes to regroup elements into Virtual Service Communities in order to have a dynamic E2E mobility management solution. This concept is also called the Community of Interest (CoI) concept. Each CoI defines a group of elements that share a common interest. Community members are self-managed and can exchange information with each other in pursuit of shared goals.

When deploying a new service in the supplier platforms, the Virtual Service Community management provides the service called *VSC Creation*. VSC Profile is included in the referential on the platform. This VSC Profile contains the following attributes: Service ID, Functionality, QoS criteria, Provider ID, SIP URI, Longitude, and Latitude, but also other attributes.

For each deployment, there is a search for the existence of the appropriate VSC. When found, this new component is attached to the VSC. Thus the VSC will contain all components functionally equivalent and with the same QOS.

However, due to mobility, some components used in the end user's session may not continue to fulfill their SLA or end-to-end user QoS requirements. To solve these mobility problems, according to the VSC concept, a ubiquitous counterpart that respects these requirements (a member of the same VSC) is dynamically found to replace the current component. As a result, this ubiquitous solution that is based on gathering QoS and functionality equivalent elements into communities guarantees all mobility cases by maintaining end-to-end sessions and their E2E QoS.

5 Use Case Service Creation and MCC Session Self-management

Today, the cloud users have become nomadic and expect to access their services without any temporal, geographical, technical or economic constraints (Figure 5). The developer defines the components they will need with their QoS requirements.

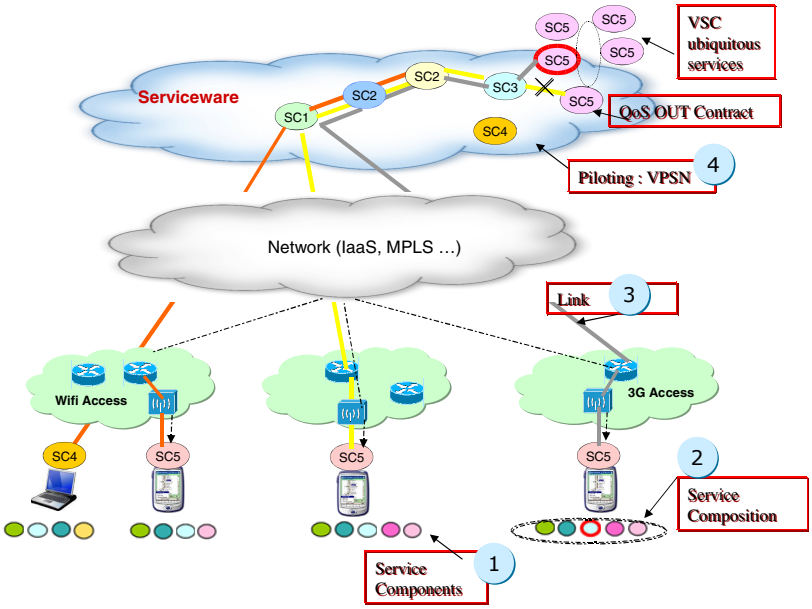

Fig. 5. Mobile Cloud Computing Use Case

Thanks to Service Creation Platform the developer will be able to in design phase (THINK):

- (1) Select the Service Components (SC) in the Referential (if they exist, otherwise it will create them), see Figure 5.
- (2) Define the service composition.
- (3) Select link, see Figure 6.
- (4) Define the service logic since the control is done on the basis of its VPSN.

In case the developer would also like to implement self-management, in the deployment phase (BUILD) he will be able select Ubiquitous Services (SC5, see Figure 5), and create the VSC.

```
<!—Network as a Service (NaaS) constraint element definitions -->
<xs:element name="appE2EDelay" type="tns:linkSetVmSetConstraint"
                  QoS requested=" currentvalue " QoS offered=" thresholdvalue " />
<xs:element name="appE2EAvaibility" type="tns:linkSetVmSetConstraint"
                  QoS requested=" currentvalue " QoS offered=" thresholdvalue " />
<xs:element name="appE2EReliability" type="tns:linkSetVmSetConstraint"
                  QoS requested=" currentvalue " QoS offered=" thresholdvalue " />
<xs:element name="appE2ECapacity" type="tns:linkSetVmSetResourceConstraint"
                  QoS requested=" currentvalue " QoS offered=" thresholdvalue " />
```

Fig. 6. OVF network link

During the operating phase (RUN), depending on the mobility, the system provides the services requested by maintaining QoS and service continuity according to these rules. As a result, VPSN and VSC establish a dynamic session that best suits the users' preferences and their QoS requirements.

Our proposals are evaluated on an autonomous Self-Management infrastructure and especially for the self-configuring of applications. Self-configuring represents the ability of a system to configure itself and to adapt the various changes in its environment. It allows the configuration and reconfiguration of a system through its autonomic and dynamic components. This autonomic infrastructure relies on JADE, a framework for the construction of autonomic systems using the Fractal reflective component model, and TUNe, a global autonomic management system [12]. It is used in OpenCloudware as part of the autonomic PaaS management layer.

The language used is OVF++. It is based on a well known formalism for describing virtual machines (OVF) and it integrates an architecture description language Fractal ADL (Architecture Definition Language). ADL is an open and extensible language to define component architectures for the Fractal component model. To treat the behavioral aspects (QoS), have introduced a new control interface (Management Interface). This interface manages and notifies the QoS of each Fractal component.

The Fractal component incorporating QoS will integrate Self-Management right from the design phase. The Self-Management Mechanisms to satisfy the continuity of service in MCC are assured by Ubiquitous Services and VSC.

6 Conclusion and Future Works

We have presented an innovative approach to domain engineering based on Service Components and Self-Management Mechanisms. This approach has been assessed and refined based on our experience in the Service Creation Environment. Our SCE adopted a Service Composition approach. Thus, the Service Components proposed have the properties recommended by SOA, namely: reusable, interoperable and autonomous, enhanced by the following properties: *mutualizable, stateless, ubiquitous and self-managed.* The Fractal *Service Component is based on a QoS* applicable in all phases of the life cycle. This architecture ensures cloud users have a composition of mobile cloud services in a seamless and dynamic way.

We have proposed the dynamic composition of the user session and a Self-Management by the Ubiquitous Services and VSC. This flexible way to create a service is essential if operators wish to by competitive.

Our future works will be the integration of our *Self-Management approach* in the PaaS platform of the OpenCloudware project.

Acknowledgments. This work is supported by the OpenCloudware project. OpenCloudware is funded by the French FSN (Fond national pour la Société Numérique), and is supported by Pôles Minalogic, Systematic and SCS.

References

1. ABI Research. Research report on Entreprise Mobile Cloud Computing: cloud Services, mobile devices, and the IT supply chain analysis (2009), http://www.abiresearch.com/research/1004608
2. ITU-T recommendation Y.2001: general overview of NGN. Y-Series recommendations: global information infrastructure, internet protocol aspects and next-generation networks, next generation networks - frameworks and functional architecture models (December 2004)
3. Mikoczy, E., Kotuliak, I., Deventer, O.V.: Evolution of the Converged NGN Service Platforms Towards Future Networks. Future Internet Journal 3(1) (March 2011)
4. Brunner, M.: Service Management in a Telecom Environment based on Active Network Technology. Ph.D. thesis No. 13433, Swiss Federal Institute of Technology Zurich (ETH Zurich), Switzerland (2000)
5. Aubonnet, T., Simoni, N.: PILOTE: A Service Creation Environment in Next Generation Network. In: Proceedings of the IEEE Intelligent Network Workshop, IN 2001, Boston, USA, Mai 6-9, pp. 36–40 (2001)
6. Chapman, C., Emmerich, W., Marquez, F.G., Clayman, S., Galis, A.: Software architecture definition for on-demand cloud provisioning. In: Hariri, S., Keahey, K. (eds.) HPDC, pp. 61–72. ACM (2010)
7. Distefano, S., Puliafito, A., Rak, M., Venticinque, S., Villano, U., Cuomo, A., Di Modica, G., Tomarchio, O.: QoS management in Cloud@Home infrastructures. In: Proc. of CyberC, Beijing, China, pp. 190–197 (October 2011)
8. Simoni, N., Xiong, X., Yin, C.: Virtual Community for the Dynamic Management of NGN Mobility. In: Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems, ICAS 2009, Valencia, April 20-25, 2009, pp. 82–87 (2009)
9. Opencloudware project (2011-2014), http://www.opencloudware.org/
10. Modica, G.D.: Resource and Service Discovery in SOAs: A P2P Oriented Semantic Approach. International Journal of Applied Mathematics and Computer Science 21(2), 285–294 (2011)
11. Zhang, X., Yin, Y., Zhang, M., Zhang, B.: A Composite Web services Discovery Technique Based on Community Mining. In: Proc. IEEE Asia-Pacific Service Computing Conference (APSCC 2009), Singapore, December 7-11, pp. 445–450 (2009)
12. Bouchenak, S., Boyer, F., Hagimont, D., Krakowiak, S., Mos, A., De Palma, N., Quema, V., Stefani, J.-B.: Architecture-based autonomous repair management: An application to J2EE clusters. In: Proceedings of the IEEE Reliable Distributed Systems (2005)
13. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D.: Interacting with the SOA-based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. IEEE Transactions on Services Computing 3(3), 223–235 (2010)
14. Mikoczy, E., Kotuliak, I., Deventer, O.V.: Evolution of the Converged NGN Service Platforms Towards Future Networks. Future Internet Journal 3(1), 67–86 (2011)