Workshop Proceedings of the 10th International Conference on Intelligent Environments
J.C. Augusto and T. Zhang (Eds.)

doi:10.3233/978-1-61499-411-4-206

Towards a Self-Adaptive Deployable Service Architecture for the Consistent Resource Management in Ubiquitous Environments

Gabriel GUERRERO-CONTRERAS [1], José Luis GARRIDO, Kawtar BENGHAZI, Sara BALDERAS-DÍAZ and Carlos RODRÍGUEZ-DOMÍNGUEZ
University of Granada
Software Engineering Department, E.T.S.I.I.T.
C/ Periodista Daniel Saucedo Aranda s/n, Granada, Spain
gjguerrero@ugr.es, jgarrido@ugr.es, benghazi@ugr.es, sarabd@correo.ugr.es and carlosrodriguez@ugr.es

Abstract. Ubiquitous environments present a series of specific challenges which must be faced in order to obtain the full potential that this kind of environments can provide to assist human beings in many scenarios. Especially, it can be challenging to appropriately manage the context changes. This feature directly affects the availability of the services deployed in the system, among other quality features, and implies additional development efforts from software developers. Service replication models and techniques may help to improve service availability and strengthen the system, e.g., against node disconnections. However, it is necessary to provide SOA with self-adaptive capabilities in order to be able to address a highly dynamic environment, creating and deploying service replicas at run-time and on demand. In this work, an analysis of the main elements involved in the dynamic deployment of service replicas is presented and how they can be combined to provide a self-adaptive software architecture to properly support the software development and dynamic service deployment in ubiquitous environments.

Keywords. Ubiquitous environments, self-adaptation, context-awareness, Service Oriented Architecture (SOA), Event Driven Architecture (EDA), SOA 2.0, resource consistency

Introduction

Ubiquitous computing refers to those computing devices which are distributed in the environment, interconnected, and with which the user interacts unconsciously. In particular, it is conceived as a distributed computing power in the

[1]Corresponding Author

environment. To achieve their full potential, new challenges that arising must be addressed, such as [1]: (1) the uneven conditioning of a ubiquitous environment must be transparent for the users. It is caused by the heterogeneous nature of the environment, where different devices with different capabilities exist, and also, due to the mobility of devices, given that some resources or features may not be always available; and (2) in order to resolve certain limitations in the use of resources, localized scalability should be supported, i.e., prevent communication between distant entities and therefore sending information beyond the local environment where it makes sense to avoid overload and bottlenecks.

These new computing environments are characterized, among other issues, by their highly dynamicity, i.e, the existence of frequent changes in the execution context. Therefore, the software system deployed in these environments needs to be sufficiently independent to adapt to changes that occur in its context, in order to maintain or even improve the user's experience, without his/her explicit intervention. This feature is known as self-adaptation (one of the areas that make up autonomic computing [2]). Self-adaptive software architectures can provide the appropriate support to address system requirements at run-time, due to changes in the availability of the resources or ongoing needs of the systems users. This kind of architectures can also improve not just system performance and efficiency, but also availability, reliability or flexibility.

In this paper, an auto-adaptive approach to support the development and deployment of software applications is presented. The approach uses and combines two previously developed solutions: the Bluerose middleware [3], an open source middleware based on event distribution approach, and adaptable services (monitoring and synchronization) [4], which have been designed to support the consistent management of shared resources in mobile and ubiquitous systems. This work intends to take a further step in availability and consistent resource management, through a self-adaptive approach which will enable a dynamic deployment of service replicas in a ubiquitous environment based on context information (context-awareness), in order to reduce the development effort of ubiquitous applications, delegating the responsibility of ensure system availability and consistent resources management at lower layers, i.e., at the service platform itself.

The rest of this paper is structured as follows. In Section 1, main technologies involved in this research are briefly described. Section 2 presents related work which have already addressed the dynamic features of these new environments; Section 3 proposes a service architecture approach to address dynamic service deployment challenge in ubiquitous environments; and finally, conclusions and future work are summarized in Section 4.

1. Background

Among the approaches to successfully address the design of software in ubiquitous environments, Service Oriented Architecture (SOA) and middleware technologies based on event-driven architectures (EDA) can be highlighted. Consequently, a software architecture for this type of environments generally is divided

into three layers: middleware, services and applications [5]. However, recent trends propose the combination of SOA and EDA paradigms, obtaining what is called SOA 2.0 [6] (a.k.a. advanced SOA), in which services are not just passive entities, but also they are able to receive and generate events proactively, thus getting the benefits of both approaches: interoperability, platform independence, flexibility and a modular design by SOA, and low coupling between components of the system, thanks to the distribution of events by EDA.

However, SOA 2.0 is not sufficient and specific aspects in ubiquitous systems need some kind of additional support. In general, the dynamic nature of this kind of systems must be also addressed in a very flexible manner, thus complementary techniques and methods to such approaches must be adopted at architectural level. To this end, for example, centralized services or static deployments should be avoided, owing to the network topology may change rapidly and unpredictably and therefore the service can be easily isolated, with serious consequences for the system operation. Faced with this problem, ad-hoc solutions are adopted for software development in ubiquitous systems, which, in the absence of a standardized method, leads to limit the functionality of the applications developed, (e.g., limit the offline operations) and imposing conditions in the execution environment to ensure the correct operation of the system (e.g., a permanent connection to the Internet). This results in lower availability and scalability in incorporating new functionality, users and technologies.

Replication services techniques can help to strengthen the system against node disconnections, improve response time and balance the workload. However, these replicas should be created and deployed at runtime in response to specific situations of context [7], in order to respond properly to a dynamic environment. Moreover, an ubiquitous environment involves some degree of collaboration between the different participants of the system, so it is common to find shared resources, such as files, registers, and knowledge bases, which are distributed and/or replicated along the system. These resources are modified by several participants (users, agents or services) frequently and concurrently, which implies to manage access/modification of these resources consistently in order to ensure proper system operation. Maintaining the correctness of the shared data is not a simple task, particularly in environments where network disconnections are frequent [8].

2. Related Work

Several research works have addressed challenges of dynamic environments through software adaptation, however, as mentioned earlier, software adaptation is a wide field of research and although some of these works do not address the problem entirely, they usually address some issues involved in the dynamic deployment of services, as mentioned by this section.

In order to avoid that some devices become isolated and can not access a service, network partitions detection mechanism are useful, this allows deploying a services replica while it is still possible. In [9], disjoints paths between client and server are calculated and for each path, robustness of link nodes is estimated.

To this respect, services replicas can be placed to improve the availability of the service for clients. With the same objective, in [10] TORA routing protocol [11] is used in combination with a estimation of the residual link lifetime of wireless links. This residual link lifetime between two devices is calculated using response time of periodical beacon messages, and when a node predicts a partition, this node will host a replica of the service.

With the aim of reducing bandwidth consumption and to detect potential network partitions too, some proposals identify mobility groups and create devices clusters. Dustdar et al. [12] use devices clusters to perform an efficient monitoring. In each cluster only the most powerful device performs an active monitoring. These clusters are created on the basis of the distance between devices. However, Wang et al. [13] show that travel speed of devices is a better measure than distance to create clusters. In their work an algorithm, called Reference Velocity Group Mobility (RVGM), to identify the mobility groups is proposed. In this way, a disconnection between two groups can be predicted, e.g., when two groups have different directions and/or velocities.

Regarding to replication protocols, in [12] a copy-primary replication scheme for stateful Web services is used, aiming to reduce synchronization messages. In [14], replicas will be created when too many service requests are made from nodes that belong to other clusters, and clusters leader will host the replicas, however, synchronization of replicas is not considered. Hamdy et al. [15] propose a replication protocol based on the interest of devices in use the service. In [16], REDMAN middleware for dense MANET is proposed. The replication protocol of REDMAN is based in a gossip strategy, when a resource or replica must be replicated, this request is sent to service and it replicates itself between its neighbors with a r-hop distance previously defined, nevertheless, the replicated resources are read-only, this is, replicas synchronization is not considered.

In this way, some these works do not take account certain features when deploying replicas: energy consumption, host device capabilities, network topology or replicas synchronization. The aim of this paper is to provide an architectural approach and a methodological approach to address previous issues, i.e., the dynamic deployment of service replicas by taking into consideration synchronization and efficient consumption of resources.

3. A Self-Adaptive Architecture Approach

In this section, an adaptive approach to support the development and deployment of software applications is presented, in order to reduce the development effort of ubiquitous applications, delegating the responsibility of ensure system availability and consistent resource management at lower layers. The software architecture is based on three main layers: middleware, services and applications (Figure 1). This solution combines two previously developed proposals: the Bluerose middleware, an open source middleware that implements both the Request-Response and the Publish-Subscribe paradigms, and at service level, the use of adaptable services (monitoring and synchronization), which have been designed to facilitate the consistent management of shared resources in mobile and

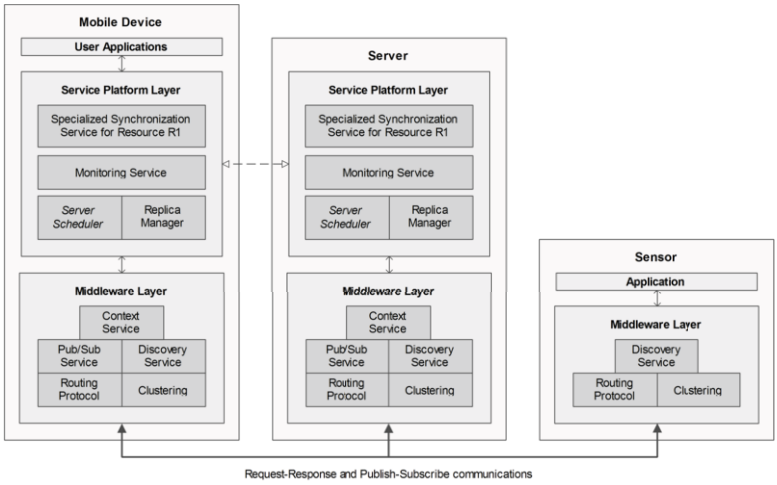

Figure 1. Service platform design, layers and main components.

ubiquitous systems. This work intends to take a further step in availability and consistent resource management, through an self-adaptive approach which aims to enable a dynamic deployment of service replicas in a ubiquitous environment based on context information. The following subsections describe in detail the main features that are required from the middleware layer, the service platform to be used in order to achieve the objective of this research work, and finally a case of study where the feasibility of the approach is shown.

3.1. Middleware Layer

The main sought features of a middleware supporting mobile/ubiquitous applications, and therefore based on EDA, are (Figure 1) [3]: *event dissemination*, which allows, through a publish/subscribe communication mechanism, a broadcast and asynchronous communication with loose coupling between interfaces, what implies to achieve a scalable solution at communication level; *dynamic discovery*, in order to know what devices and services are available in a specific moment, in an automatic and transparent way; *context management*, the middleware of a self-adaptive software architecture must also be adaptive, therefore it must manage basic context information for its effective functioning. The device resources (battery or communication capabilities), network topology, available services or entities, device's stability and mobility, are cross-cutting properties, that can also affect to the adaptation of the architecture; *devices clustering*: in order to facilitate the monitoring and the deployment of replicas in the network, clustering may help to split the issue under local areas and facilitate the scalability. This philosophy is already adopted in routing problems (Cluster-based Routing Protocol [17], "CBRP"); and *routing*: the possible absence of a predetermined topology or central control implies that the communication middleware must be prepared to act in wireless ad-hoc networks, where traditional routing algorithms cannot be used.

3.2. Service Layer

At service layer, two issues are addressed: the consistency of shared resources of the system and the availability of the services. The first issue has been addressed by defining a methodological approach to develop services. Due to the synchronization algorithms are dependent of the resource type, it is not possible to provide a general service for the synchronization. For this reason and with the goal of providing a reusable service, an abstract service, called Synchronization Service, is provided. This service must be specialized as required by the specific resource to be shared. This is intended to provide a service applicable to any type of shared resource, facilitating the development of application services in ubiquitous environments (Figure 1). This abstract service is based on a Monitoring Service. It is a basic service which stores all kind of information about changes on shared data, which are represented and managed as events. Regarding synchronization, this information will be required when the synchronization algorithms will be applied. Additionally, when a disconnection occurs, this information is fundamental to allow offline operations in the system. The Monitoring Service supports different configurations in the system, it can be accessed from other system component to store some event, for example from the Synchronization Service; or it can work as a subscriber too, following the approach SOA 2.0. This feature allows an efficient monitoring, due to the monitoring work can be divided between different replicas of the service, distributing the workload and improving scalability.

However, this solution is not enough. A service can manage a shared resource and the synchronization of different changes in the resource is possible, nevertheless, if the user's device loses its connection, due to user's mobility, he/she will not be able to be carried out changes on the shared resource and therefore he/she cannot continue working. In order to avoid this situation, the availability of the services is improved through a self-adaptive deployable service architecture. To this end, adaptation logic is encapsulated in the Replica Manager component (Figure 1). It is divided into two parts: the Server Scheduler and the Replica Manager itself. Each instance of the Replica Manager handles the replication and deployment of only one service, while the Server Scheduler is responsible of managing the different instances of the Replica Manager. This is a more efficient and scalable solution than having only one Replica Manager component to manage the replication and deployment of all services in the system, since each instance is focused on the requirements of replication and deployment of a unique service.

The interaction between the platform's components during the self-adaptation process is shown using a BPMN diagram in Figure 2. When a service wants to be registered in the system, it sends its requirements to the Server Scheduler. The Server Scheduler creates a new instance of the Replica Manager, which is associated with the new service registered in the system. This instance of the Replica Manager processes the requirements of the service and it subscribes to the corresponding events (e.g., if the service requires a certain free amount of memory, the Replica Manager will subscribe to events notifying less amount of memory). When the Replica Manager receives one event it will consult to the

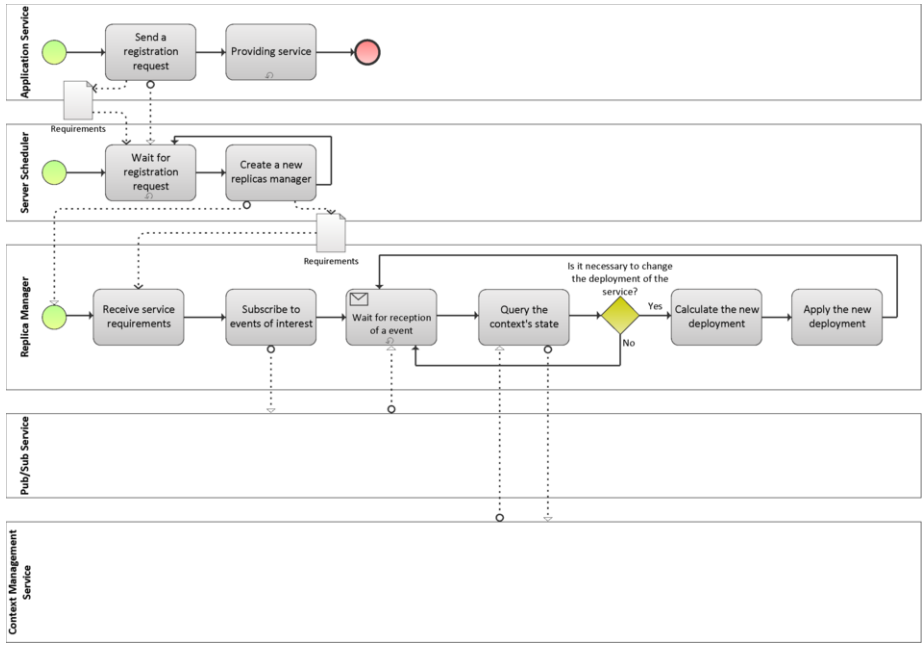

Figure 2. Collaboration diagram between platforms services in order to provide an adaptive deployment to application services.

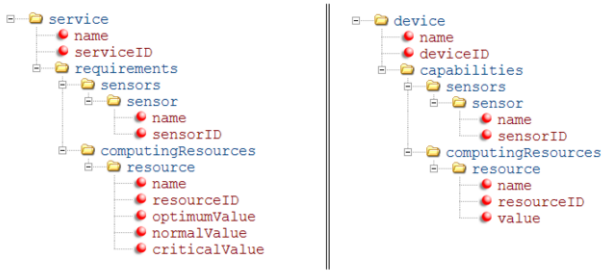

Figure 3. XML schemas that specify the format for defining service requirements and device capabilities

Context Management Service to check if it is necessary a node change for the service. For example, if an event informing low battery level is received, this may indicate that the service must be migrated, however, if the rest of devices have less battery, migrate the service could make no sense. If it is necessary to change the deployment of the service, the Replica Manager selects the most adequate node to deploy the service.

To achieve this task, the decision of the Replica Manager is based on two kinds of information: service requirements and cross-cutting features. While service requirements are features that affect specifically to a service, the cross-cutting

features affect to any service deployed in the system. On the one hand, regarding the requirements of a service, these must be clearly specified. The service developer indicates which capabilities must provide the devices in which the service could be deployed. These capabilities affect to services differently, one service could need processing power and another high storage capacities, for this reason the services must specify its requirements independently through the XML schema defined in Figure 3 (left side). In this schema should be given: service name and ID, sensors required (e.g., GPS or camera) and computational resources (e.g., CPU or memory), which should further indicate optimal, normal and critical value (e.g., Memory: 0.2GB critical, 1GB normal and 2GB optimal). Moreover, each device must specify through a similar XML schema Figure 3 (right) its current available resources. On the other hand, in this work the following main cross-cutting features have been identified: battery, position within network topology and distance to service's clients. The Replica Manager, through an evaluation function which encapsulates the above information (Algorithm 1), can evaluate the nodes and deducts if another node can be a better candidate for hosting, in which case the service will be migrated to the new node host, or if a new replica is necessary, it can select the more suitable node to place it.

Algorithm 1 Calculate node ranking to host a service's replica

```
nodeRanking ← 0
for all sensorRequired ∈ service.sensorsRequired do
  if sensorRequired ∉ node.sensors then
    return 0
  end if
end for
for all RRequired ∈ service.resourcesRequired do
  if RRequired.criticalVal > node.resource.value then
    return 0
  else if RRequired.criticalVal ≤ node.resource.value < RRequired.normalVal then
    nodeRanking ← nodeRanking + 0.2
  else if RRequired.normalVal ≤ node.resource.value < RRequired.optimumVal then
    nodeRanking ← nodeRanking + 1
  else
    nodeRanking ← nodeRanking + 1.2
  end if
end for
nodeRanking ← nodeRanking + node.batteryLevel/100 * W1
nodeRanking ← nodeRanking + node.directConnections/node.maxConnections * W2
nodeRanking ← nodeRanking + node.averageDistance/node.furthestClient * W3
return nodeRanking
```

3.3. Case Study

The case of a Mobile Forensic Workspace [18] is of interest to show the feasibility of the approach. There are different scenarios: natural disasters, accidents, terrorist attacks, murders, etc., where security forces apply protocols of action intended to support victim identification. In order to support data sharing between

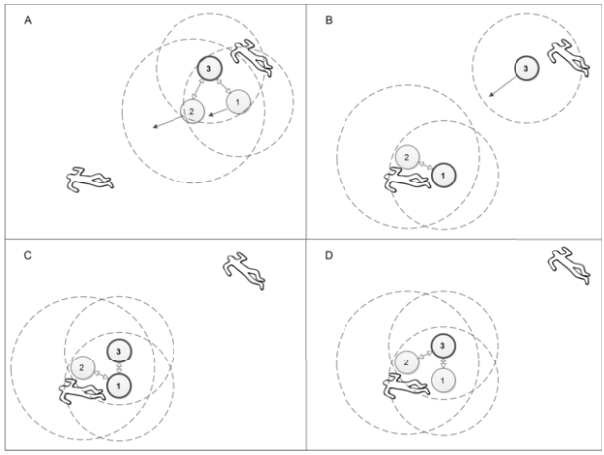

Figure 4. Example scenario of the case study. Circles 1, 2 and 3 are devices of the members of the forensic team. The highlighted devices (e.g., 3 in step A) are devices which host a replica of the service. The dashed circles indicate the coverage of devices, the black arrows the direction of users' movement, and the white ones the client-server relations.

forensic experts, the forensic support system must allow exchange information with nearby applications, devices, etc. However, the consistency of this information can be compromised owing to in some scenarios (e.g., natural disasters or rural environments) common network infrastructures may not be available and the users are moving around the scene, which implies unstable connections (disconnections and network partitions). This makes collaboration more difficult and therefore techniques must be applied to ensure the availability of the system and the consistency of the shared resources.

Suppose for example the scenario depicted in the Figure 4. There are three mobile devices, each belonging to a member of the forensic team. These users are making collaborative annotations about a accident scene through a service, called Annotation Repository Service. This service is a specialization of Synchronization Service (Figure 1) and it ensures the consistent management of the annotations performed by the forensic team. The service is initially deployed in the device 3 (Figure 4.A) and provides service to devices 1 and 2. This service is registered through the Server Scheduler by sending its requirements, which has been specified through XML scheme proposed:

```
<service name="annotationsRespository" serviceID="service130563">
  <requirements>
    <computingResources>
      <resource name="CPU" resourceID="#CR01" optimumValue="1.4 GHz"
                normalValue="1.2 GHz" criticalValue="1 GHz" />
      <resource name="Memory" resourceID="#CR02" optimumValue="2GB"
                normalValue="1GB" criticalValue="0.5GB" />
      <resource name="Storage" resourceID="#CR04" optimumValue="4GB"
                normalValue="2GB" criticalValue="1GB" />
    </computingResources>
  </requirements>
</service>
```

And, therefore, an instance of the Replica Manager has been created and assigned to this service. However, the users of devices 1 and 2 are moving to another location of the scenario, therefore an event of a possible disconnection is published. The Replica Manager, which is associated with the service, receives this event and searches an adequate device, on the basis of the service requirements, to deploy a service replica and thus ensuring the availability of the service. To this end, the Replica Manager queries the Context Management Service the information about the devices which form the group and receives its specifications:

```
<device name="Device01" deviceID="01">
  <capabilities>
    <sensors>
      <sensor name="Camera" sensorID="#S11"/>
      <sensor name="GPS" sensorID="#S12"/>
      <sensor name="Accelerometer" sensorID="#S13"/>
      <sensor name="Light Sensor" sensorID="#S14"/>
    </sensors>
    <computingResources>
      <resource name="CPU" resourceID="#CR01" value="1.2 GHz"/>
      <resource name="Memory" resourceID="#CR02" value="1 GB"/>
      <resource name="Storage" resourceID="#CR04" value="4 GB"/>
    </computingResources>
  </capabilities>
</device>


<device name="Device02" deviceID="02">
  <capabilities>
    <sensors>
      <sensor name="Camera" sensorID="#S11"/>
      <sensor name="GPS" sensorID="#S12"/>
      <sensor name="Accelerometer" sensorID="#S13"/>
    </sensors>
    <computingResources>
      <resource name="CPU" resourceID="#CR01" value="1.7 GHz"/>
      <resource name="Memory" resourceID="#CR02" value="2 GB"/>
      <resource name="Storage" resourceID="#CR04" value="1 GB"/>
    </computingResources>
  </capabilities>
</device>
```

The Replica Manager through the evaluation function can deduct that device 1, in current situation is most suitable device to host a replica of the service. The position in the network topology and distance to services clients of the service are equal in this scenario, and we can assume the same level of battery, then in this case only service requirements make the difference in the choice. Therefore, device 1 obtains 3.2 scoring and the device obtains 2.6 (when applying Algorithm 1). This is because although the device 2 is more powerful (better CPU and memory) the storage capacity is in a critical level taking into account the Annotation Repository Service requirements. This may be because to the device 2 has other components which make use of storage capacity too. Therefore, a replica is deployed in the device 1 to ensure the availability of the service.

Later, when the user of device 3 has finished working in his/her area, meets with the rest of the team (Figure 4.C). First, when a connection is established between the two replicas, a process of synchronization occurs. When the two replicas are correctly synchronized and the shared annotations are in a consistent state, due to it is a small team, one replica will be turned off by the Replica Manager in order to save resources (Figure 4.D).

4. Conclusions and Future Work

In this work, some of the main challenges associated to the software development for ubiquitous environments have been identified. The approach is based on the dynamic deployment of service replicas in ubiquitous environments, also describing how the replication may help to address the identified challenges. The different responsibilities of middleware and service layers have been exposed. The foundations have been laid to provide a self-adaptive architecture to reduce the efforts of developers in the development and deployment of software applications in ubiquitous environments, and thus increasing service availability and consistent management of shared resources. The solution proposed also helps to make transparent the uneven conditioning, due to the developer only describes the requirements of the service and does not need to be worried about the environmental characteristics; and it also addresses localized scalability, through the self-adaptive replication of the services, communication between distant entities is avoided, since services can be carried to areas where are the users who are using them. It should be noted that this solution is independent of the services technology used (e.g., SOAP or REST), and an approach based on events would be only needed. Moreover, contrary to related works, this solution takes into account both the synchronization of the shared resources and cross-cutting context features, such as network topology, to place replicas, not only the device's resources. Finally the requirements of the service can be adapted to each particular service, providing a flexible solution.

In the future work, the final implementation of the platform, currently under development, is intended, in order to perform a study about the different configuration parameters, such as monitoring intervals, and to compare in an empirical way the behavior and the performance of different approaches, such as different services replication techniques. In this way, a more comprehensive study of behavior adaptation for a wider range of complex context situations is expected to be obtained.

Acknowledgment

This research has been partially supported by the Spanish Ministry of Economy and Competitiveness with European Regional Development Funds (FEDER) under the research project TIN2012-38600.

References

[1] M. Satyanarayanan, Pervasive computing: Vision and challenges, *Personal Communications, IEEE* **8** (2001), 10–17.

[2] J. Kephart and D. M. Chess, The vision of autonomic computing, *Computer* **38** (2003), 41–50.

[3] C. Rodríguez-Domínguez, T. Ruiz-López, K. Benghazi and J. L. Garrido, Designing a middleware-based framework to support multiparadigm communications in ubiquitous systems, *Advances in Intelligent and Soft Computing* **153** (2012), 163–170.

[4] G. Guerrero-Contreras, J. L. Garrido, C. Rodríguez-Domínguez, M. Noguera, and K. Benghazi, Designing a service platform for sharing internet resources in MANETs, *Advances in Service-Oriented and Cloud Computing* **393** (2013), 331–345.

[5] C. Machado, E. Silva, T. Batista, J. Leite, and E. Yumi Nakagawa, Architectural elements of ubiquitous systems: A systematic review, *International Conference on Software Engineering Advances* **8** (2013) 208–213.

[6] P. Krill, Make way for soa 2.0, *http://www.infoworld.com/t/architecture/make-way-soa-20-420* (2006).

[7] K. Kakousis, N. Paspallis, and G. A. Papadopoulos, A survey of software adaptation in mobile and ubiquitous computing, *Enterprise Information Systems* **4** (2010) 355–389.

[8] S. B. Davidson, H. Garcia-Molina, and D. Skeen, Consistency in a partitioned network: a survey, *ACM Computing Surveys (CSUR)* **17** (1985) 341–370.

[9] M. Hauspie, J. Carle, D. Simplot, Partition detection in mobile ad-hoc networks using multiple disjoint paths set, *International Workshop on Objects models and Multimedia technologies* **15** (2003) 1–15.

[10] A. Derhab, N. Badache, and A. Bouabdallah, A partition prediction algorithm for service replication in mobile ad hoc networks, *Wireless On-demand Network Systems and Services* (2005) 236–245.

[11] V. D. Park and M. S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, *Conference of the IEEE Computer and Communications Societies* **3** (1997) 1405–1413.

[12] S. Dustdar and L. Juszczyk, Dynamic replication and synchronization of web services for high availability in mobile ad-hoc networks, *Annual Joint Conference of the IEEE Computer and Communications Societies* **1** (1997) 19–33.

[13] K. H. Wang and B. Li, Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks, *Service Oriented Computing and Applications* **2** (2002) 1089–1098.

[14] A. Ahmed, K. Yasumoto, N. Shibata, and T. Kitani, Hdar: Highly distributed adaptive service replication for MANETs, *IEICE transactions on information and systems* **94** (2011) 91–103.

[15] M. Hamdy and B. König-Ries, A service distribution protocol for mobile ad hoc networks, *Proceedings of the 5th international conference on Pervasive services* (2008) 141–146.

[16] P. Bellavista, A. Corradi, and E. Magistretti, Redman: An optimistic replication middleware for read-only resources in dense MANETs, *Pervasive and Mobile Computing* **1** (2005) 279–310.

[17] L. E. Quispe and L. M. Galan, Behavior of ad hoc routing protocols, analyzed for emergency and rescue scenarios, on a real urban area, *IEICE transactions on information and systems* **41** (2014) 2565–2573.

[18] C. Rodríguez-Domínguez, K. Benghazi, J. L. Garrido, and A. V. Garach, Designing a communication platform for ubiquitous systems: The case study of a mobile forensic workspace, *New Trends in Interaction, Virtual Reality and Modeling* **41** (2013) 97–111.