A Middleware for Managing Dynamic Software Adaptation

Rafael Oliveira Vasconcelos
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
{rvasconcelos}@inf.puc-rio.br

Igor Vasconcelos
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
{ivasconcelos}@inf.puc-rio.br

Markus Endler
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
{endler}@inf.puc-rio.br

ABSTRACT

The design and development of adaptive systems brings new challenges since the dynamism of such systems is a multifaceted concern that range from mechanisms to enable the adaptation on the software level to the (self-) management of the entire system using adaptation plans or system administrator, for instance. Networked and mobile embedded systems are examples of systems where dynamic adaptation become even more necessary as the applications must be capable of discovering the computing resources in their near environment. While most of the current research is concerned with low-level adaptation techniques (i.e., how to dynamically deploy new components or change parameters), we are focused in providing management of distributed dynamic adaptation and facilitating the development of adaptation plans. In this paper, we present a middleware tailored for mobile embedded systems that supports distributed dynamic software adaptation, in transactional and non-transactional fashion, among mobile devices. We also present results of initial evaluation.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Distributed applications*.

General Terms

Management, Measurement, Performance, Design, Experimentation, Verification.

Keywords

dynamic adaptation; mobile communication; middleware; adaptability; self-adaptive systems

1. INTRODUCTION

The coordination of decentralized dynamic adaptation of software components in distributed systems has always been a challenge, because of problems such as management of dynamism, service discovery, disruption of remote services, and transient situations where part of the system is updated [4] [16]. With the ongoing trend towards networked and mobile embedded systems, adaptation become even more necessary as the applications must be capable of discovering the computing resources in their near environment and the corresponding capabilities of sensors and actuators. This much more dynamic setting, however also brings new and not yet solved problems related to the management of dynamic adaptation in self-adjusting applications such as those for Internet of Things (IoT). IoT is part of a future Internet and ubiquitous computing and may be defined as a dynamic global network infrastructure with self-configuring capabilities where physical and virtual "things" (e.g., sensors, actuators, smartphones and vehicles) are seamlessly integrated into the information network [19]. Reasons for adaptation include hardware defects, software errors, sudden resource changes, or bursts of communication or processing demand.

In many applications, "things" are mobile things, such as smartphones, wearable devices, vehicles or autonomous robots with specific sensors and actuators. A typical scenario is thus the discovery and *ad hoc* interaction among such mobile devices for mutual exchange of sensed data. For example, when a user carrying a smartphone enters a vehicle, the mobile application on the smartphone could connect and interact with the vehicle´s control system in order to provide some features, such as local pre-processing and transmission of the vehicle's telemetric data to a cloud service or perform in-vehicle adjustment of equipment and media (e.g. seat and mirror position, and music selection), according to user preferences. However, there are many challenges to build such applications since vehicles have different types of sensors, actuators, general capabilities, and APIs (Application Programming Interfaces). Therefore, it is not feasible to develop a mobile application for all possible vehicles and passenger customization options from scratch. A better approach is to implement the integration software for each vehicle type or model as a component (i.e., plug-in). Thus, one can independently add the specific software component for new sensors and actuators of new vehicle models as needed, and even do this dynamically during the application´s execution.

Current middleware solutions for building adaptive systems do not provide the required support to handle dynamic changes in the cyber/physical context of the devices [6]. The authors of [6] also claim that there are few efforts aiming to adapt and tailor existing

ARM'14, December 9, 2014, Bordeaux, France.

http://dx.doi.org/10.1145/2677017.2677022

software to the specific aspects of IoT (e.g., high dynamicity and distribution, real-time reactivity, resource constraints, and error/defect-prone environments). One should also add scalability, fault- and connectivity-tolerance, mobility support and dynamic adaptation as further important issues when designing software for Internet of *Mobile* Things (IoMT) systems. To the best of our knowledge, there are no other work that cope with all the aforementioned aspects. As the main goal of our research is to manage and facilitate the development of adaptive systems, we present an approach that supports distributed dynamic adaptation, in transactional and non-transactional fashion, among Mobile Nodes (MNs) for adaptive systems.

The remainder of this paper is organized as follows. Section 2 provides an overview of the key concepts and technologies used in our approach. Section 3 presents our proposed approach for mobile dynamic adaptation and gives some details about the implemented prototype. Section 4 shows initial performance results of our prototype. Section 5 reviews some of the most relevant works related to ours. Finally, Section 6 contains concluding remarks and discusses future work on the central ideas presented in this paper.

2. BASIC CONCEPTS

This section presents and discusses the main concepts and technologies that underlie our approach for dynamic adaptation.

2.1 Scalable Data Distribution Layer (SDDL)

SDDL [18] is a communication middleware that connects stationary DDS (Data Distribution Service) [13] nodes of a wired "core" network with MNs that have an IP-based wireless data connection. SDDL employs two communication protocols: Real-Time Publish-Subscribe RTPS Wire Protocol [13] for the wired communication within the SDDL core network, and the Mobile Reliable UDP protocol (MR-UDP) [3] for the inbound and outbound communication between the core network and the mobile nodes. DDS delivers many advantages in performance, scalability, availability and QoS (Quality Of Service) mechanisms [1] [17] and defines a fully distributed Peer-to-Peer (i.e., broker-less) and scalable middleware architecture based on the Real-Time Data-Centric Publish-Subscribe (DCPS) communication model [17]. SDDL provides a communication infrastructure to interconnect mobile devices so to build IoMT systems.

As part of the SDDL core, the Gateway plays an important role concerning our work. It defines a unique point of attachment for connections with the mobile nodes. Thus, the Gateway is responsible for managing a separate MR-UDP connection with each of these nodes, forwarding any application-specific message or context information into the core network, and in the opposite direction, converting DDS messages to MR-UDP messages and delivering them reliably to the corresponding Mobile Node(s).

2.2 Dynamic Software Adaptation

Dynamic software adaptation aims at behavioral, functional or structural modification of a software component (e.g., class, service, module or functionality, for instance) at run-time [9]. It allows systems to act in response to new application requirements and/or context changes in the physical and/or virtual execution environments [6] [16]. For example, aircraft avionics systems must respond and adapt quickly to situations such as hardware/software defects and sudden changes of resource availability, in order to allow continued and safe operation [15]. Due to the mobility of things in IoMT system, the availability of different sensor/actuator technologies at each place may vary constantly and unexpectedly. Moreover, the need of introducing new (or else optimizing existing) in-network processing functionalities, and the need to change the IoMT system's adaptation capabilities, for instance, are also some examples where dynamic software adaptation is required in IoMT.

The two most popular approaches to perform software adaptation are parameter and compositional adaptations [9] [10]. Parameter adaptation is related to the change of software parameters (i.e., tunable variables) with the aim of modifying the system's behavior. Adjusting software parameters may enhance the system's suitability to the current context (e.g., system overload). However, with parameter adaptation, it is impossible to deploy new software components or algorithms into a running system, and hence this type of adaptation is usually not sufficient for extending a system's functionality or capabilities. Conversely, compositional adaptation enables not only simple parameter modification and component selection (i.e., selection of different strategy implemented at the design time), but also the dynamic exchange/addition of algorithms and protocols in the system, in order to cope with the new requirements and context situations that may arise after software deployment [16] [10]. Thus, compositional dynamic software adaptation is required for applications/systems which are faced with unforeseen requirements, heterogeneous and evolvable technologies, and unexpected situations [9], such as in IoMT.

3. SYSTEM ARCHITECTURE

With the aim of facilitating the development of adaptive applications for IoMT, our approach (**Error! Reference source not found.**) supports both parameter and compositional adaptation. One of the most widespread approaches in software engineering is to separate the application's business logic from adaptation logic [8] [4] [9] [6]. Following this trend, we provide an API where adaptation engineers, the ones responsible for building the application´s adaptation plans, can register for notifications of system events (e.g., availability of new devices, component failure reports, and other application-specific events) so as to create specific adaptation plans and tasks, as means of reacting to those events.

All adaptations performed at the MNs are driven and orchestrated from the SDDL Core Network, where *Adaptation Manager* monitor all MNs, manage software component deployments, and coordinate the execution of the adaptation actions by the MNs.
The Adaptation Managers are responsible for coordinating the system-wide adaptation process (e.g. deployment of new

software components or customization of application´s parameters) on many mobile nodes. It consists of three modules: *Monitoring Manager*, *Global Repository Module*, and *Global Adaptation Service*. The Monitoring Manager is responsible for discovering and monitoring resource usage at the MNs (e.g., available sensors/actuators, available core memory, and version of the components deployed) yielding this information as input for the global adaptation management processes. The Global Repository Module stores components that may be deployed to the MNs. Finally, the Global Adaptation Service is responsible to initiate and coordinate the execution of all the operations that encompass the decentralized, system-wide adaptation. For example, if the global adaptation is the deployment of a new functionality to several MNs, which may consist of some components, the Global Adaptation Service will send the code that implements the new functionality to all MNs and then verify whether all MNs successfully deployed it.

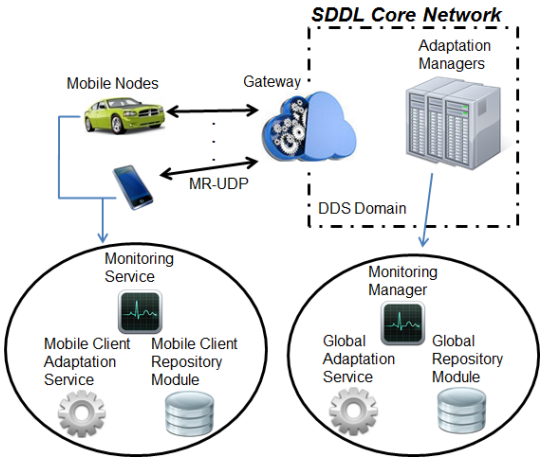

Figure 1. Overview of the proposed approach

Our approach further requires following three components executing on the MNs: *Monitoring Service, Mobile Client Repository Module* and *Mobile Client Adaptation Service*. The Monitoring Service continuously monitors the MNs looking if there is any change in the MN´s environment (e.g., MN became aware of a new sensor, for example at a nearby smart device), and if so, informs the Monitoring Manager at the Adaptation Manager in the SDDL Core. The Mobile Client Repository Module stores components previously received from the Adaptation Manager so to avoid the need for resending components in case of a MN reboot. In this way, the MN is able to restore its latest state (i.e., all the deployed components). The Mobile Client Adaptation Service is the mechanism that executes the adaptation at the MN (e.g., deployment of a new functionality) and informs the Adaptation Manager whether this operation could or not be successfully performed.

3.1 Management of Distributed Adaptation among MNs

While some adaptations affect only a single MN with no impact on other MNs, such as the example of the interaction among vehicles and MNs, other situations require that adaptations be performed in parallel in some, or even all, MNs. To illustrate the latter case, consider a situation where the Adaptation Manager needs to update a component used for the interaction between MNs (e.g. a new communication protocol module, or a specific cryptography algorithm). Thus, the Adaptation Manager has to inform that all, or none, online MNs should perform the deployment of the new component. We call this sort of adaptation as *transactional adaptation*, in contrast to *non-transactional* adaptation. The transactional adaptation was designed to ensure that all online MNs involved in such adaptation successfully execute an adaptation plan (i.e., execute the same steps without errors), or there is no change on the online MNs. Whether some MN goes offline during the procedure, the Adaptation Manager drives a rollback and the SDDL guarantees that the offline MNs receive the messages when they go back online. To the best of our knowledge, our work seems to be the first one that supports transactional and distributed dynamic adaptation over MNs.

An adaptation plan may be composed of a sequence of *commands*, i.e. actions that are executed at the MNs. Currently, the implemented commands are *insert*, *remove*, *enable* and *disable* a component, plus *update* a component instance. Using the previous example of updating the communication protocol, the adaptation plan for such case has the following commands: insert (deploy) component, enable component and update component's instance. In a transactional adaptation, the update command generates a second command, *commit update*. Thus, in the first phase, MNs inform whether they are able to update the component's instances and, in the second phase, they conclude the update process by committing all changes.

As a dynamic adaptation may cause some error or side effect (e.g., degrading the system´s performance), we support *adaptation rollback*. Thus, if some adaptation causes any undesirable situation, the Adaptation Manager is able to restore automatically the system to the previous system´s valid configuration. Considering that some MN failed to update the component's instances in our example, the commands to restore the system are *rollback update*, *disable component* and *remove component*.

3.2 Implementation Details

Two key artifacts on our implementation are the component itself and the wrapper (i.e., a kind of Java proxy). The component is the software artefact that may be dynamically deployed on MNs or may be updated for a newer version. When we update a component *A*, we have to update all previous instances of component A. For such, we need a mechanism able to transparently update the component instances and to keep their references for the application [4]. Thus, we have developed the wrapper, which is a container for component instances, that manages the dynamism related to the process of update component instances. In order to be managed by the wrapper, the component must implement a simple interface, shown in Figure 2. With the IComponent interface, the wrapper is able to initialize a component and to transfer the state (i.e., any variables and values

that has to be passed to clone the state from the old to the new instance) from one component to another.

```
public interface IComponent<State, Parameters> {
  public boolean initialize(Parameters state);

  public boolean loadState(State state);

  public State getState();
}
```

Figure 2. IComponent interface

Figure 3 exemplifies the implementation of a component. This component has three methods (foo, bar and doWork) related to its own logic and other three methods related to the dynamism. Although the component implementation does not need to know how the update process is done, it needs to be aware that an update process may occur.

```
public class ComponentTestAlternative
             implements ComponentInterfaceTest {
  private String state;

  public void foo() {
  public void bar(final String paramenter) {
  public void doWork(final int number) {

  @Override
  public boolean initialize(final String state) {
    this.state = state;
    return false;
  }
  @Override
  public boolean loadState(final String state) {
    this.state = state;
    return true;
  }
  @Override
  public String getState() {
    return this.state;
  }
}
```

Figure 3. Example of a component implementation

On the other hand, the Adaptation Manager exposes only the method *sendAdaptationPlan()* to the adaptation engineer at the current development stage. An *adaptation plan* consists of a list commands (e.g., insert or remove component) to be executed at the MNs, a list of commands that were already executed, another list containing the MNs involved in such adaptation plan, and a flag informing whether the plan is transactional or not. The list of the executed commands is applied when a transactional adaptation plan fails and it has to be undone using the rollback capability. With the aim of managing the transactions, the Adaptation Manager applies the traditional 2PC (Two-Phase Commit) protocol [11] at this point. Though newer atomic commit protocols (ACPs) has been proposed specific for mobile environments [12], the 2PC protocol was chosen for validation purpose only due to its implementation simplicity.

4. PRELIMINARY PERFORMANCE TESTS AND RESULTS

So far, we have developed and tested only some features of our approach. More specifically, we implemented just non-transactional and compositional adaptation into our current prototype. Hence, we are currently able to deploy new components and update existing components at the MNs. In our performance tests, we have measured the time required to deploy and to update a component (with non-transactional adaptation), as well as the overhead of invoking (i.e., calling) a component through our component wrapper. We also tried to identify the impact of an adaptation on the regularity of a data flow produced by a MN.

4.1 Experimental Setup

Our hardware test was composed of Dell Laptop Intel i5-3210M 2.5GHz, 8GB DDR3 1333MHz and Wi-Fi (802.11n) interface running Windows 8.1 64 bits, Dell Laptop Intel i5 M 480 2.66GHz, 8GB DDR3 1333MHz and Wi-Fi (802.11n) interface running Ubuntu 12.04 LTS 64 bits, and a Fast Ethernet router with 802.11n wireless connection. Our prototype has been implemented using the Java programing language and we have simulated the MNs as a Java application.

In order to evaluate the time required to deploy or update a component, we did two experiments. One experiment measured the elapsed local time on the MN to complete the deployment/update of the component. In each case, we repeated the experiment 10 times. The JAR (Java ARchive) file that encapsulates the deployed component has 1.5KB (kilobytes) and the Java class that represents the component has 51 lines of code. While the first experiment measured the local time, the second experiment measured the Round-trip Delay (RTD), which encompasses the time interval from the instant of time the Adaptation Manager sends the adaptation plan until it receives an acknowledgment informing that all MNs completed the execution of all commands that compose the adaptation plan, of both transactional and non-transactional adaptations. We repeated the latter experiment with 1, 10 and 100 MNs. Finally, with the aim of assessing the invocation overhead that our wrapper imposes, we measured the time of 50,000 calls using direct invocation (i.e., calling the component's method without our wrapper) and invocation through the wrapper that represents the component.

4.2 Results

The results of the two experiments that measure the time required to deploy/update a component and the RTD of the transactional and non-transactional adaptations are shown in Table 1. The confidence level for all results is 95%. The deployment of a component on the MN took 2.03 ms on average, while the process of update a single component instance took 0.05 ms on average. While the number of MNs was increased in a ratio of 100 times, the RTD increased 16.38 times for the non-transactional adaptation and 8.55 times for the transactional adaptation. Comparing the non-transactional adaptation with the transactional one, the overhead imposed by the transactional mechanism was 170.92%, 52.11% and 41.41% for 1, 10 and 100 MNs, respectively.

The time elapsed to execute 50,000 times the component´s method was 467.36 ms using the wrapper and 452.76 ms calling directly the component´s method. Thus, the overhead imposed by

our wrapper was 3.22%, which seems to be reasonable for the most of the applications considering the functionality provided by the wrapper.

Table 1. Time required to deploy and update a component, and to execute an adaptation plan

Experiment	Mean RTD	Standard Deviation	Confidence Interval
Local Deployment	2.03 ms	0.23	+/-0.14 ms
Local Update	0.05 ms	0.01	+/-0.01 ms
Non-transaction 1 MN	5.64 ms	0.41	+/-0.25 ms
Non-transaction 10 MN	23.95 ms	1.44	+/-0.89 ms
Non-transaction 100 MN	92.37 ms	2.10	+/-1.30 ms
Transaction 1 MN	15.28 ms	0.85	+/-0.53 ms
Transaction 10 MN	36.43 ms	4.30	+/-2.67 ms
Transaction 100 MN	130.62 ms	10.07	+/-6.24 ms

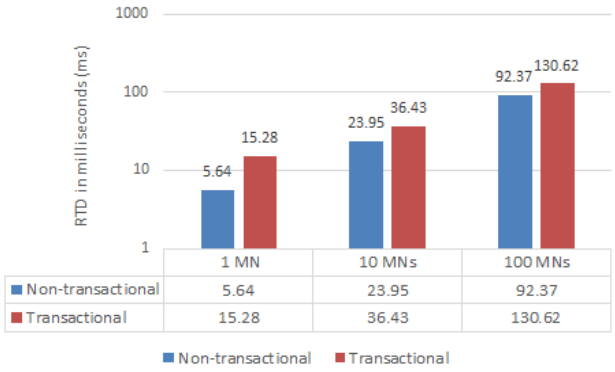

Figure 4. RTD for transactional and non-transactional adaptation plan

5. RELATED WORK

Although not exhaustive, the works discussed in this section are the most relevant ones we encountered with respect to our approach. The work by [14] proposes a mobile application – Mobile Sensor Hub (MoSHub) – that allows a variety of different sensors to be connected to a mobile phone. The authors developed an architecture to dynamically interconnect sensors to a mobile application by generating a wrapper class. While the MoSHub approach is tailored for generating wrappers class for sensor devices, our approach is more general and intends to perform generic dynamic adaptation on mobile applications. Hence, dynamic configuration of sensor would be one use case for us. Authors [14] also do not discuss about features such as scalability, fault tolerance and distributed adaptation, for instance.

Another relevant work to us is the one proposed by [2]. The authors present an architectural model addressing flexible and adaptive composition of services in Very Large Scale (VLS) IoT systems by exploiting the concepts of service orchestration (i.e., centralized approach) and choreography (i.e., decentralized approach). While the authors follow a service orchestration/choreography model, which seems to be more adequate for web applications, we chose following the service-oriented component approach [4] [5]. Although the authors address VLS IoT systems, there is no information about how the architecture achieves scalability, and how the adaptation engineer defines the service composition.

iPOJO (injected Plain Old Java Object) [5] [4] is a service-oriented component framework that aims to simplify the development of dynamic service applications. The iPOJO framework is implemented on top of the OSGI (Open Service Gateway initiative) [7] service platform, which is a framework to deploy services in a centralized and non-distributed environment. The basic idea of iPOJO is using containers to inject handlers with the aim of managing non-functional behavior, such as dynamism and service discovery. The most remarkable differences among iPOJO and our approach is that it completely adheres to the Service-Oriented Computing (SOC) and there is no notion of distributed adaptation that involves many MNs. On the other hand, by being inspired by SOC, we can create an adaptation plan logic to decide which components (or services) have to be deployed on each MN. Thereby, we can enable the development of autonomic managers in charge of managing the whole system. Although such differences, we share ideas such as the concept of containers (or wrappers) and handlers, and the separation among the business logic and the non-functional properties such as dynamism.

6. CONCLUSION AND DISCUSSION

In this paper, we propose an approach to manage the complexity of building adaptive mobile application. Instead of managing low-level adaptation techniques (i.e., how to dynamically deploy new components or change parameters), we are focused in providing management of distributed dynamic adaptation and facilitating the development of adaptation plans. Hence, the main contributions of this paper are (i) the design of (non-) transactional distributed adaptation, (ii) the design of an interface to decompose the system in small and independent components, and (iii) an evaluation that validates and measures our prototype. Several studies have been conducted in the field of middleware for adaptive applications; however, most current efforts does not take into account problems such as mobility, scalability and manageability. Problems such as parametric variability and adaptation reasoner, which is responsible for deciding when an adaptation is required, which alternative best satisfies the overall system goal, and what adaptation operations are needed in order to drive the system to the next state (i.e., an optimal state or state with a new functionality), are not covered by our research. On the other hand, we are focused in providing scalable management of coordinated and distributed dynamic adaptation, and facilitating the development of adaptation plans.

Security is an important concern for many real systems, particularly for distributed systems since there is communication involved with other devices, things and services, for instance. Therefore, authenticity, integrity and confidentiality emerge as

key aspects. In our scenario of dynamic software adaptation for IoMT, ensuring that only the adaptation engineer or the system itself have the ability to drive a software adaptation will avoid unauthorized component deployments, such as viruses, on the MNs. However, at the current stage of our research, we are not concerned with security requirements.

We are aware that much work and research is still needed; however, considering the encouraging preliminary performance evaluation, we are confident that our approach will facilitate the development of adaptive systems. We expect the following contributions in this and the next years: (i) an API tailored to develop mobile adaptive applications; (ii) a mechanism to enable adaptation engineers to receive and handle events generated by the MNs; and (iii) the support for parameter adaptation.

7. REFERENCES

[1] Baptista, G.L.B., Roriz, M., Vasconcelos, R., Olivieri, B., Vasconcelos, I. and Endler, M. 2013. *On-line Detection of Collective Mobility Patterns through Distributed Complex Event Processing*. Monografias em Ciência da Computação - MCC 12/2013, Dep. de Informática, PUC-Rio, ISSN 0103-9741.

[2] Dar, K., Taherkordi, A., Rouvoy, R. and Eliassen, F. 2011. Adaptable service composition for very-large-scale internet of things systems. *Proceedings of the 8th Middleware Doctoral Symposium on - MDS '11* (New York, New York, USA, 2011), 1–6.

[3] David, L., Vasconcelos, R., Alves, L., Andre, R., Baptista, G. and Endler, M. 2012. A Communication Middleware for Scalable Real-Time Mobile Collaboration. *IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (Jun. 2012), 54–59.

[4] Escoffier, C., Bourret, P. and Lalanda, P. 2013. Describing Dynamism in Service Dependencies: Industrial Experience and Feedbacks. *Proceedings of the 2013 IEEE International Conference on Services Computing* (Washington, DC, USA, 2013), 328–335.

[5] Escoffier, C., Hall, R.S. and Lalanda, P. 2007. iPOJO: an Extensible Service-Oriented Component Framework. *IEEE International Conference on Services Computing (SCC 2007)* (2007), 474–481.

[6] Gurgen, L., Gunalp, O., Benazzouz, Y. and Gallissot, M. 2013. Self-aware cyber-physical systems and applications in smart buildings and cities. *Proceedings of the Conference on Design, Automation and Test in Europe* (San Jose, CA, USA, 2013), 1149–1154.

[7] Hall, R., Pauls, K., McCulloch, S. and Savage, D. 2011. *Osgi in Action: Creating Modular Applications in Java*. Manning Publications Co.

[8] Jacques-Silva, G., Gedik, B., Wagle, R., Wu, K.-L. and Kumar, V. 2012. Building user-defined runtime adaptation routines for stream processing applications. *Proceedings of the VLDB Endowment*. 5, 12 (2012), 1826–1837.

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

[10] McKinley, P.K., Sadjadi, S.M., Kasten, E.P. and Cheng, B.H.C. 2004. Composing adaptive software. *Computer*. 37, 7 (Jul. 2004), 56–64.

[11] Nouali, N., Doucet, A. and Drias, H. 2005. A Two-phase Commit Protocol for Mobile Wireless Environment. *Proceedings of the 16th Australasian Database Conference - Volume 39* (Darlinghurst, Australia, Australia, 2005), 135–143.

[12] Nouali-Taboudjemat, N., Chehbour, F. and Drias, H. 2010. On Performance Evaluation and Design of Atomic Commit Protocols for Mobile Transactions. *Distrib. Parallel Databases*. 27, 1 (2010), 53–94.

[13] OMG Data Distribution Portal: 2014. *http://portals.omg.org/dds/*. Accessed: 2014-09-15.

[14] Perera, C., Jayaraman, P., Zaslavsky, A., Christen, P. and Georgakopoulos, D. 2013. Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm. *IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing* (Apr. 2013), 473–478.

[15] Phan, L.T.X. and Lee, I. 2011. Towards a Compositional Multi-modal Framework for Adaptive Cyber-physical Systems. *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications* (Aug. 2011), 67–73.

[16] Ramirez, A.J. and Cheng, B.H.C. 2010. Design patterns for developing dynamically adaptive systems. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '10* (New York, New York, USA, 2010), 49–58.

[17] Vasconcelos, R.O., Endler, M., Gomes, B. and Silva, F. 2013. Autonomous Load Balancing of Data Stream Processing and Mobile Communications in Scalable Data Distribution Systems. *International Journal On Advances in Intelligent Systems (IARIA)*. 6, 3&4 (2013), 300–317.

[18] Vasconcelos, R.O., Silva, L.D.N. e and Endler, M. 2014. Towards Efficient Group Management and Communication for Large-Scale Mobile Applications. *5th International Workshop on Pervasive Collaboration and Social Networking (PerCol), co-located with Percom* (Budapest, 2014).

[19] Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I.S., Mazura, M., Harrison, M., Eisenhauer, M. and others 2011. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*. (2011), 9–52.