Self-Aware Context in Smart Home Pervasive Platforms

Philippe Lalanda[1], Eva Gerber-Gaillard[1] and Stéphanie Chollet[2]

[1] Grenoble Alpes University
700 Avenue Centrale, St Martin d'Hères
F-38000 Grenoble, France
firstname.name@imag.fr

[2] LCIS Grenoble INP
F-26000 Valence, France
stephanie.chollet@lcis.grenoble-inp.fr

***Abstract*— Pervasive computing envisions environments where computers are blended into everyday objects in order to provide added-value services to people. A growing number of advanced embedded systems, extended with computing and communication capabilities, are already appearing around us. However, pervasive applications raise major challenges in terms of software engineering and remain hard to develop, deploy, execute, and maintain. We believe that autonomic techniques are here needed, in particular to deal with context-awareness. In this paper, we propose to handle context management in a service-oriented pervasive platform by defining a self-aware solution and associated mechanisms. Our approach is illustrated with a smart home example implemented in our pervasive platform iCasa.**

Keywords— self-awareness; context; service-oriented pervasive platforms; pervasive applications; smart home.

I. INTRODUCTION

There is today a plethora of communication-enabled computing devices that can be embedded in our environment to interact with the physical world. This opens the way to the implementation of advanced pervasive applications, providing added-value services to human beings in their living places [1-3]. Current services, provided by telecom and software companies, are promising but limited in terms of functionalities and often kept isolated. This is essentially due to the difficulty to develop, deploy, and run applications based on a dynamic set of heterogeneous devices.

In this paper, we focus on pervasive services in smart homes, a very active and emblematic domain. Interestingly enough, the definition of smart home has been changing as technology evolved. A few years ago, "smartness" was essentially a matter of remotely controlling major home functions related to HVAC (Heating, Ventilation, and Air-Conditioning) or security for instance. Now, smart homes are seen as pervasive environments where a number of autonomic services for users, also called applications, can be installed and run to improve quality of life. Services pertain to a variety of domains like healthcare, senior care, energy management, security or comfort. Today, most applications, from high-level management to physical devices, are specific to a given provider (like Homelive by Orange [4], Smart Things by Samsung [5], or Smart Home by Panasonic [6]). We believe that, in the short or medium term, they will be independent of a provider and able to be run in heterogeneous environments.

A salient point is that smart home applications are context-aware by essence. They need to adapt their behaviors and the provided services according to environmental conditions. The notion of environment should be taken in its broad sense here. It includes any information that can be of interest for an application like, for instance, physical quantities, locations and expectations of human beings (implied or not in the functions provided by the application), the software itself, and even remote digital resources. Initially, context was essentially limited to location-awareness [7] and to information collection. Since then, it has evolved towards more elaborate models. As a consequence, developing and evolving context information for pervasive applications is still very challenging. This explains why research into context-awareness is a firmly grounded activity in computer science and will continue to expand with the recent emergence of Internet of Things, IoT [3].

We have developed a pervasive platform, called iCasa [8], providing a service-oriented component model and a number of facilities supporting the development and runtime management of pervasive applications. This platform also includes context management facilities. In our solution, the context is a programming module, connected to the physical environment and publishing information as a dynamic set of services. Context is dynamic in order to reflect the changing nature of the execution environment but also to deal with applications evolving needs. However, context adaptations cannot be done by remote administrators or, even less, by users. This is why we seek to apply autonomic solutions to context management in order to always provide relevant content. Specifically, the context module has to reason about the applications needs (its goals), its own possibilities of adaptation (its capabilities), and what is currently available in its environment (its resources). These notions are tightly coupled to the self-awareness vision of autonomic computing, as we will explain in this paper.

In this paper, we present a self-aware solution to dynamic context management in iCasa. Our proposition is tested with actual use cases from Orange Labs. The remainder of the paper is structured as follows. Next section provides background about pervasive platforms and self-awareness. Our proposal is developed in sections III and IV. First evaluations are presented in section V. Section VI is about related work and section VII concludes this paper.

2474-0756/17 $31.00 © 2017 IEEE
DOI 10.1109/ICAC.2017.1

II. BACKGROUND

Service-oriented computing [9] is at the heart of smart home solutions. Indeed, devices are implemented as services. They can be searched and invoked on local or remote networks. Also, most pervasive platforms are built on top of service-oriented models defining suitable programming abstractions and providing technical facilities. This is for instance the case for iCasa that is based on the Apache Felix iPOJO [10], a service-oriented component model, and that provides a number of facilities including service discovery and publication, action scheduling, data storage, etc. Services may correspond to devices like a lamp for instance, or to any computing unit providing useful functions like luminosity in a room. This is illustrated hereafter by figure 1.

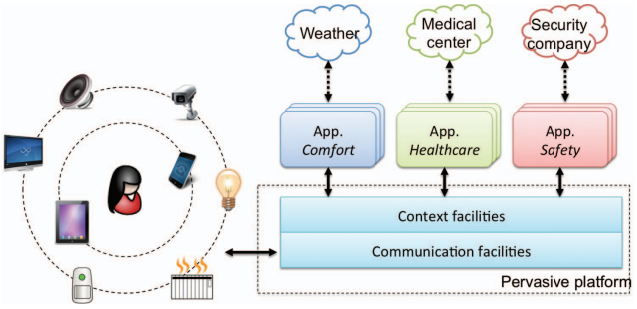

Figure 1. Pervasive platform.

Many pervasive platforms include some sort of facilities regarding context management, thus providing the necessary middleware between applications and changing environments. The explicit definition and use of a context layer enables application code to be less coupled with the executing environment and favors adaptations.

In iCasa, the context module holds synchronized descriptions of concepts and relationships between them pertaining to the execution environment. Precisely, contextual information comes from the computing environment (network, application needs, non-functional goals, etc.), the user environment (needs, preferences, profile, etc.), the physical environment (areas, time, temperature, brightness, noise, etc.), and the aggregations and relations between them (location of a user/device, average temperature of a room, etc.). It can be the continuous description of a fact, an entity, a value, or even the depiction of a discrete event. Context is usually structured as four complementary levels [11], that is:

- Acquisition: collection and synchronization of data from heterogeneous sources;
- Modeling: representation that can be computationally interpreted and manipulated;
- Reasoning: data processing, aggregation, inference, to get a more abstract view from problem space perspective;
- Dissemination: presentation, distribution to context-aware applications.

Context is often considered in a unidirectional way, from the environment to the applications, because its first purpose is to help qualifying the situation so the applications can adapt their behavior. However, the boundary between the context layer and the applications is more blurred that it seems, especially for processing task, and the context layer can also be designed to ease environment manipulation through actuators.

Programming and managing the context layer is still quite a difficult task. This is due to the dynamicity and heterogeneity of pervasive environments, but also to the fact that the relevancy of contextual information heavily relies on current applications needs. This requires to constantly adjust the contextual information to be captured, processed, and presented. A context layer, thus, needs to be regularly adapted to always meet the current applications needs. Such dynamic adaptation has to be done autonomically. Indeed, there is no administrator in pervasive environment, but only users with no, or limited, technical background. We cannot expect them to perform advanced administration tasks. Using remote skilled administrators is not a solution either because of its lack of scalability.

Our purpose is to make use of autonomic techniques to implement a context module able to self-adapt to new internal and external conditions, including new applications, new needs, or new resources. Precisely, our approach relies on self-aware notion to implement dynamic context management. Self-awareness comes from philosophy and psychology. It can be defined as the capacity to become the object of one's own attention [12]. In [13], a self-aware computing system is defined as a system that learns models capturing knowledge about itself and its environment, reasons using the models in accordance with higher-level goals, and acts upon its environment or internal structure.

Self-awareness could thus be achieved by implementing a model-based learning, reasoning, and acting loop (see figure 2 here after for illustration). It is interesting to note that those models are mostly based on contextual information, as it is the case in pervasive systems.

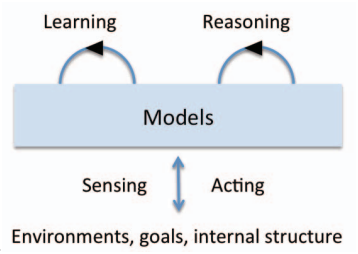

Figure 2. Self-aware basic loop.

The self-aware basic loop has to be addressed during architectural design. This allows engineers to explicitly decide and reason about the system self-awareness capabilities. However, although supported by early works [14], this architecture remains very abstract and insufficiently backed up. We are actually only at the beginning to understand what is crucial for the architecture of self-aware computing systems and how architectural styles and decisions impact the self-awareness capabilities of a system [13]. Also, there is no widely accepted way to represent and manipulate models in self-aware systems. For all these reasons, designing and implementing such systems remain a major challenge.

III. ARCHITECTURE

A. Overview

We have defined and implemented a new architecture for iCasa, which is now explicitly made of three interacting layers (see figure 3). Each layer of the architecture is self-aware in the sense that it knows its goals, its internal structure, and the external resources it can use. Based on this information, an autonomic manager dynamically maintains the best architectural configuration within the layer.

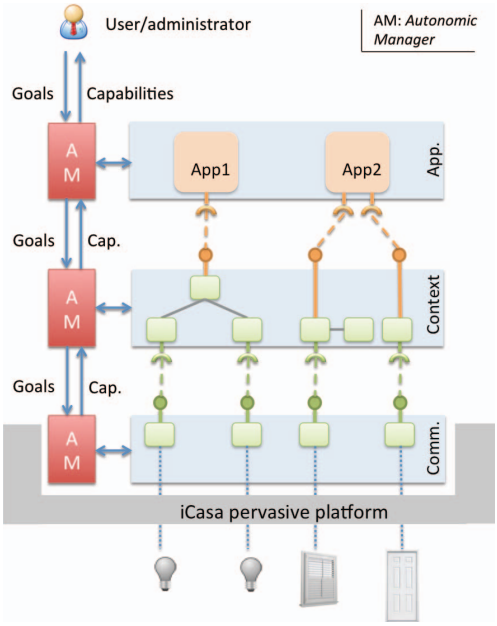

Figure 3. Proposal overview.

More precisely, iCasa includes the following layers:

- The communication layer handles interactions with dynamic and heterogeneous resources like devices, pervasive platforms or Web services. It is based on the RoSe middleware [15] and presents remote resources as services to the next layer.
- The context layer provides information of interest for running applications. It is designed to abstract and enrich external resources and to ease application development. Contextual information is presented as a dynamic set of services to the next layer.
- The application layer contains application code that is expected to focus on business aspects. It leaves complex technical code related to communication or context management to the platform.

The three layers are implemented with iPOJO service-oriented components. This allows local autonomic adaptation at each layer. IPOJO relies on the "inversion of control" pattern (it controls execution flow of applications through dependency injection) and provides an extensible component container that manages dynamism issues. In particular, it manages all the service-oriented interactions: service publication, service instantiation, service selection, and service discovery.

In the architecture, iPOJO components are accessible through different scopes. Some components are published at the platform level, in a global registry, and can be used by other layers. Other components provide services that are used within a layer as computing resources. They are not published at the platform level, but kept in the layer scope.

As introduced, each layer includes an autonomic manager. This manager is in charge with the creation, configuration, and removal of the iPOJO components implementing the goals of the layer. Bindings between components are still managed by the iPOJO containers. If needed, however, the AM can re-configure the architecture of its layer to adapt to new or unsatisfying situation.

Each autonomic manager relies on three models representing its goals, its resources, and the layer runtime architecture. These models are described in more details in the next section focusing on the context layer.

B. Goals and capabilities

Autonomic managers of adjacent layers communicate which each other through goals and capabilities. A goal can be seen as a request of services. A service is defined by:

- A set of Java methods defined by their signature,
- A set of properties (like the service owner or the version number),
- A quality of services (defined as symbolic attributes in the current implementation).

Precisely, goals are expressed as statements relating services with conjunctions or disjunctions. Using logical connectors is a way to express some variability. It provides some flexibility to target autonomic managers in their task.

Capabilities represent services that can be provided on demand by a layer. Here again, services are defined by their signature, properties and quality of service. Capabilities are service potentials. This means that iPOJO components implementing the services are not instantiated. They are instantiated by the autonomic manager if a goal asks for it. Capabilities are dynamically updated depending on the environmental evolutions.

In order to achieve its goals, a layer relies on its internal computing resources, implemented as iPOJO components, and on the capabilities of the lower layer. If those capabilities are needed, the layer sends a goal to the lower layer autonomic manager that will provide the requested services through appropriate components instantiations.

Our architecture also introduces feedback control loops, interconnecting the autonomic managers. When a goal is issued, feedback about the goal achievement is provided. If a goal cannot be achieved or if it requires too much time because some resources are not available anymore, the requesting AM is alerted. It can then change its internal architecture and ask for different services. The requesting AM has to record when alternating configurations in order to avoid adaptation oscillations.

IV. SELF-AWARE CONTEXT

As introduced, we turned the iCasa context into a self-aware component, reasoning about its goals, resources and runtime architecture. It can therefore decide on the best information to provide to running applications, given their needs and the available resources. In the remainder of this paper, we concentrate on this context layer, which is original and representative of the global approach.

As suggested in [13], the context layer autonomic manager relies on several models that are built at runtime:

- A goal model contains logical statements about services required by the application layer,
- A resource model contains the services that can be provided on demand by the communication layer,
- An architecture model contains a description of the runtime iPOJO-based architecture of the context layer.

Those models are the basis of the AM reasoning capabilities, which allows it to manage the context components in order to feed the application layer with the appropriate services. This architecture is illustrated here after by figure 4.

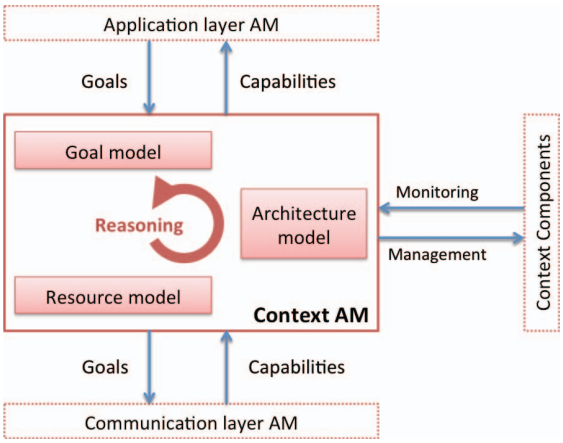

Figure 4. Context layer autonomic manager.

Let us examine these three models. The goal model represents the services the layer has to provide. It is built from the applications goal requests. It also maintains the necessary information to feedback the application layer, like the goals implementation status. Goals to achieve change every time an application is installed or uninstalled. They can also be updated during the execution of an application when different architectural configurations are explored, but this requires additional efforts on adaptation coordination.

The resource model represents services that are potentially available in the environment. It is built and dynamically updated from the communication layer capabilities. The list of available services changes according to network discoveries, but also every time defined goals (filters on requested device services) are modified. It is to be noted that specific request can be issued, asking the communication layer to look for a service in the environment.

The architecture model represents the internal configuration of the context in terms of iPOJO components and bindings. The model is a simplified view of the architecture focusing on aspects that can be managed by the AM. Precisely, the architecture model provides the following information:

- IPOJO components and their properties,
- Dependencies between iPOJO components,
- Bundles deployed in the platform.

In order to get the architecture model, we have extended iPOJO containers. We have added a monitoring handler that provides information about the component and its runtime properties. We have also added an interceptor handler that is called when a binding is created or updated. This is illustrated by figure 5 here after.

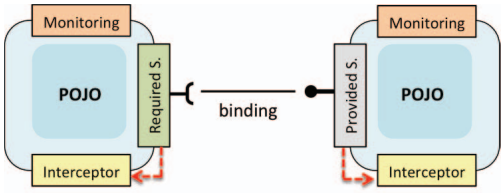

Figure 5. iPOJO container extensions.

The context layer autonomic manager reasons about these models to determine the appropriate context layer architecture. It tries to satisfy applications demands: if something required is not present in the resource model, it will seek for suitable alternatives. The adaptation strategies are application dependent. It is to be noted that the AM is not based on trivial code. First, as the number of components grows, combinatorial explosion may happen. Moreover, adaptation can be triggered by several events occurring with different frequencies, at different levels of abstraction, such as application installation or change of value of a service property. Also, the autonomic manager should not react to every change because an adaptation comes at a cost. The designing of the resolution mechanism and strategies needs to take these concerns into account.

In order to better manage the context, we have recently developed a Domain Specific Language (DSL) that is used by the autonomic manager. Indeed, developers implementing a context service must meet thorny requirements. In general, contextual entities require timely interaction with devices or computing resources. Such interaction requires error-prone code for synchronization and event handling. The DSL specifically copes with this need. It is implemented in the iCasa platform with Java annotations and heavily relies on iPOJO mechanisms.

The purpose of the DSL is also to enforce architectural patterns commonly encountered when dealing with context management. For instance, a common implementation pattern is to maintain within the component an in-memory representation of the current state of the environment, and keep this representation synchronized. Our DSL supports this pattern using state fields. The interested reader will find more technical details about the DSL in [25].

V. VALIDATION

ICasa is tested and continuously improved with several Orange Labs use cases: *Actimetry* (medical application), *LightFollowMe* (comfort), *Alarm management* (security) *Device Installation (*user assistance) and *Simulator* (development and test). For the evaluation of the work presented in this paper, we focused on two aspects: the complexity of the new solution and its overhead.

A. Complexity

For the first part of the evaluation, we developed the simulator application with the older iCasa version and with the new one, meeting the same functional requirements. The first implementation turned out to be rather straightforward but difficult to extend and evolve, especially at runtime. We then redesigned the code, using the new architecture, with self-awareness and dynamic adaptation in mind. To compare the two versions, we measured the following code metrics: number of lines of code, cyclomatic complexity (number of linearly-independent paths) and technical debt (evaluation of the effort needed to fix all issues). These metrics are computed by an open source quality management tool, SonarQube.

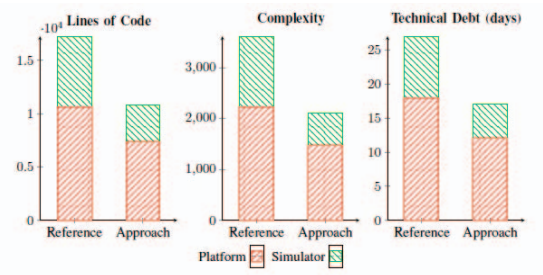

Figure 5. Metrics for the *simulator* application.

Results are given by figure 5. Thanks to Java annotations and associated autonomic mechanisms, the number of lines of code decreased. By clearly identifying synchronization functions and limiting their number, cyclomatic complexity has been reduced too. The DSL notably lightens the context layer development. It offers non-functional technical facilities. The context is modularized and extensible, and provides the necessary touchpoints for self-adaptation. The whole software is more consistent, testable and maintainable.

One reason of the decreased complexity is that tricky technical code is now handled by the platform and the context manager. This includes mechanisms like context service lookup and monitoring, context entities state synchronization and event propagation, context provisioning. Context acquisition and dissemination are now mostly automated.

The second experiment compares two versions of LightFollowMe. This application does not imply complex algorithms, but it has to deal with complex pervasive issues like environment dynamism and heterogeneity (lights and sensors can appear/disappear, someone may interfere). In the first implementation, the application handled contextual information like presence per zone by directly accessing sensors. In the new version, such information is dynamically managed by the context layer and provided as services.

LightFollowMe is a small application; so modularizing the presence service brought an important overhead in terms of lines of code and complexity. However, this context service can be shared between multiple applications and evolves independently from the applications. Also, the new architecture gives the possibility for platform and applications developers to easily extend the context module with additional services. Every context entity is instrumented with touchpoints that enable its manipulation at runtime. This extra-feature can be a burden when developing small applications, especially for the externalization of contextual concepts that are usually hard-coded. Figures are provided by figure 6.

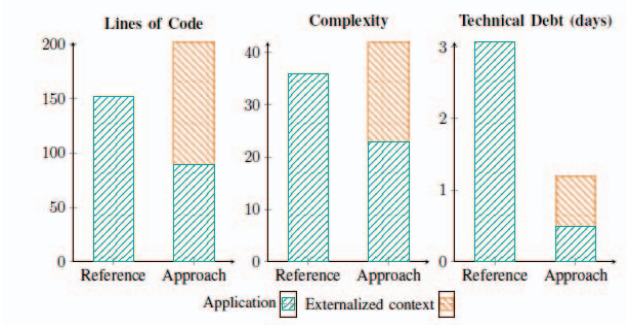

Figure 6. Metrics for the *LightFollowMe* application.

B. Adaptation and overhead

We have also evaluated the overhead in terms of execution time for three applications: Simulator, LightFollowMe and Alarm management. The execution time of the applications based on the new architecture turned out to be very slightly greater than the execution time of the initial applications. This overhead is mainly related to the time needed for building the models and for asking the creation of contextual services (instead of creating them when application is launched). For the three applications, the time needed is negligible. It is not significant regarding the time scale of the related scenarios.

Finally, we have explored the adaptability of the three applications. To do so, we run many scenarios on top of iCasa. ICasa includes a script language allowing to dynamically instantiate simulated device, location and user. This script also allows performing timely interactions and, then, defining very precise scenarios. The architecture showed great adaptability because of the diversity of services that can be provided. However, adaptability has to be provisioned in the applications. During the experiments, it appeared that the adaptation solutions implemented in the autonomic manager were not always satisfying, depending on the runtime situations. Running the scenarios allowed us to refine the adaptation strategy, which is a key point (once deadlocks are detected).

To study adaptation strategies, we implemented several "context creators". They handle the creation of specific context entities upon some conditions, reasoning on local parts of the architecture. Context creators can be activated/deactivated from a dedicated dashboard. Thus, we could decide on the context creators to be used, dynamically or statically. We believe that the selection of creators could be automated based on high-level goals, but the implementation is not finalized yet.

VI. RELATED WORK

Many surveys about context middleware have been published [16,17,18,19]. Most works are applied to pervasive computing, in domains like smart buildings, smart cities, or smart factories [20]. In this section, we compare our approach to context middleware specifically using autonomic solutions for adaptation [21,22].

ACoMS (Autonomic Context Management System) [23] promotes an approach dealing with fault-tolerance regarding the dynamism of context provisioning by using autonomic solutions. In this work, applications describe their needs in terms of context facts and ACoMS can autonomously configure and reconfigure its context acquisition and pre-processing functionality. ACoMS promotes autonomic behavior but context sources are at a sensor level and no clear guidelines are provided to construct more abstract concepts. Moreover, we think that by infusing autonomic touchpoints at a finer granularity, more advanced autonomic behavior can be brought to context management.

In [24], authors deal with proactive adaptation and context management based on a service-oriented architecture. It underlines the fact that context interactivity is not just about providing the most powerful modeling and reasoning engine. Indeed, applications also can deal with context in a proactive manner, with the ability to change context through actuators. Our approach, in this sense, is very similar. To do so, a specific query language is provided, with a steep learning curve.

To our knowledge, there is no work that considers applying self-awareness principles on context middleware or pervasive platforms yet. Self-awareness is still a recent notion in autonomic computing, and it has not been applied extensively. In [13], architectures and techniques are described to deal with smart homes. However, there is no focus on context. Also, these proposals are not yet validated on industrial use cases. On the contrary, this work is closely developed with Orange Labs and applied to concrete use cases, in real settings when possible.

VII. CONCLUSION

In this paper, we propose an approach applying self-aware techniques to context management in pervasive platform. More precisely, we build upon model-based architectures as proposed in [13]. Three models, representing goals, resources and architecture have been implemented to guide reasoning. Our proposal is implemented within the iCasa pervasive platform, developed with Orange Labs, and used to implement our common use cases with success. Indeed, it turned out that the overhead of the solution is negligible given the use case real-time constraints and that contextual information is provided in accordance with applications needs.

Further perspectives of our work consist in evaluating the ability of our system to support scalability. Actually, the system is tested using a limited number of applications. However, the greater the number of the pervasive applications is, the greater the number of context adaptations is necessary. This may increase the overhead to the system. A solution to be investigated is to manage applications QoS globally in order to converge rapidly to an acceptable solution for all.

REFERENCES

[1] M. Weiser, “The computer for the 21st century,” in Scientific american, vol. 265, pp. 94-104, 1991.

[2] M. Satyanarayanan, “Pervasive computing: vision and challenges,” in Personal Communications, IEEE, vol. 8, pp. 10–17, 2001.

[3] R. Caceres and A. Friday, “Ubicomp systems at 20: Progress, opportunities, and challenges,” IEEE Pervasive Computing, vol. 11, no. 1, pp. 14–21, 2012.

[4] http://homelive.orange.fr/accueil/

[5] https://www.smartthings.com/

[6] https://www.panasonic.com/uk/consumer/smart-home.html

[7] Martin Bauer, Christian Becker et Kurt Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. Personal and Ubiquitous Computing, vol. 6, no. 5, pages 322 – 328, 2002

[8] C. Escoffier, S. Chollet and P. Lalanda, "Lessons learned in building pervasive platforms," in Consumer Communications and Networking Conference, pp. 7-12, 2014.

[9] P. Papazouglou and D. Georgakopoulos, “Service-oriented computing,” in Communications of the ACM, vol. 46, issue 10, 2003.

[10] C. Escoffier, R. S. Hall and P. Lalanda, “iPOJO: an extensible service-oriented component framework,” in Service Computing, pp. 474-481,07.

[11] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 414–454, 2014.

[12] A. Morin, “Levels of consciousness and self-awareness: A comparison and integration of various neurocognitive views,” *Consciousness and cognition*, vol. 15, no. 2, pp. 358–371, 2006

[13] S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, Self-Aware Computing Systems, Springer, 2017.

[14] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, “A survey of self-awareness and its application in computing systems,” in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*. plus 0.5em minus 0.4emIEEE, 2011, pp. 102–107.

[15] Bardin, J., Lalanda, P., Escoffier, C.: Towards an automatic integration of heterogeneous services and devices. In: 5th IEEE Asia-Pacific Services Computing Conference, APSCC 2010,, Hangzhou, IEEE Computer Society (2010) 171{178

[16] D. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. IJAHUC 2(4) (2007) 263-277

[17] Perera, C., Liu, C.H., Jayawardena, S., Chen, M.: A survey on internet of things from industrial market perspective. IEEE Access 2, (2014).

[18] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing, vol. 6, 2010.

[19] Bellavista, P., Corradi, A., Fanelli, M., Foschini, L.: A survey of context data distribution for mobile ubiquitous systems. ACM Comput. Surv. 44(4), 2012.

[20] P. Lalanda, D. Morand and S. Chollet, “Autonomic mediation middleware for smart manufactoring”, IEEE Internet Computing, January 2017.

[21] J. O. Kephart and D. M. Chess. The vision of autonomic computing. Computer, 36(1):41{50, 2003

[22] P. Lalanda, J. A. McCann, and A. Diaconescu, Autonomic Computing-Principles, Design and Implementation, ser. Undergraduate Topics in Computer Science. Springer, 2013.

[23] P. Hu, J. Indulska, and R. Robinson, “An autonomic context management system for pervasive computing,” in IEEE Pervasive Computing and Communications, 2008. PerCom 2008.

[24] S. VanSyckel, G. Schiele, and C. Becker, “Extending context management for proactive adaptation in pervasive environments,” in *Ubiquitous Information Technologies and Applications*. plus 0.5em minus 0.4emSpringer, 2013, pp. 823–831.

[25] C. Aygalinc, E. Gerbert-Gaillard, G. Vega, and P. Lalanda, “Service-oriented autonomic pervasive context,” in *IEEE ICSOC*, 2016.