A Model-Based Approach to Context Management in Pervasive Platforms

Colin Aygalinc[1], Eva Gerbert-Gaillard[1], German Vega[1],
Philippe Lalanda[1] and Stéphanie Chollet[2]

[1]Université Grenoble Alpes
Laboratoire LIG, CS 40700
38058 Grenoble, France
firstname.lastname@imag.fr

[2]Université Grenoble Alpes
Laboratoire LCIS
26900 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. This new form of computing gives rise to huge economical and societal expectations. However, pervasive applications raise major challenges in term of software engineering and remain hard to develop, deploy, execute and maintain. Context-awareness, in particular, is a salient and difficult property that must be met by pervasive applications. In this paper, we propose a service-oriented framework facilitating the design and execution of a context management module in pervasive platforms. Our approach is illustrated with a smart home example and implemented on top of iPOJO, the Service-Oriented Component Model of our pervasive platform iCasa.**

Keywords—context; software engineering; service-oriented components; pervasive computing.

I. INTRODUCTION

Pervasive computing envisions environments where smart devices are blended into everyday objects in order to provide added-value services to people [1] [2] [3] [4]. These devices are communication-enabled and can cooperate with each other in order to build up advanced applications (services). This new form of computing is set to become reality and is raising huge economical and societal expectations in domains like home, office, transportation, shopping, or even healthcare. Pervasive applications are designed to be invisible and non-obstructive. To provide the expected added-value services to users, they must rely on information extracted from the environment. Such information is called *context*.

Context-awareness is indeed a salient property of pervasive applications. It means that an application is able to adapt its operations according to changes in its environment. In the first days of pervasive computing, context was essentially limited to location-awareness. Since then, it has evolved towards more elaborate models. Context represents any information that can be used to characterize an entity that is relevant to the interaction between a user and an application [5]. Building and maintaining context for one or several applications is very challenging. This is partly due to the fact that smart sensors used to capture information are heterogeneous, dynamic, unreliable and sometimes mobile (like a phone for instance).

Current solutions are mostly *ad hoc* and do not exceed vertical, proprietary solutions very limited in terms of provided services. In particular, they are based on a closed world assumption and cannot deal gracefully with evolutions. We believe that advanced software engineering support is crucially needed here. Dealing with device heterogeneity and volatility, and continuous context evolution, is just too hard without adapted tooling. It demands too many hard-to-find technical skills, leading to difficult maintenance and error-prone code.

We believe that developing pervasive applications must be based upon specific software environments hiding most of the complexity presented here before. A well-established solution is to delegate some technical features to an execution platform, also called middleware [6]. Middleware forms an abstraction layer that encompasses a set of common features and services, allowing developers to focus on the development of the added-value service logic of the application.

This approach can be applied to context management [5] [7] [8] [9]. Here, some context management operations are handled by the execution platform, and not by the application developers. Specifically, context middleware provides ways to support the development of context-aware applications, emphasizing the fundamental concerns of gathering, modeling, processing, and disseminating environmental information. However, we believe that some limitations remain in terms of usability, extensibility and degree of assistance.

In this paper, we propose a service-oriented framework integrated on top of our pervasive platform, iCasa. This framework allows the straightforward definition of a context module and, at runtime, the dynamic construction of synchronized and observable entities. It also comes with architectural guidelines to ease development.

This paper is divided into seven sections. We provide background about pervasive platforms and a motivating example in section II and III. Our approach and proposition are developed in sections IV and V. Section VI focuses on the implementation of our example. Finally, we conclude in section VII and VIII with related works, limitations of our approach and envisioned future work.

II. PERVASIVE PLATFORMS AND CONTEXT

An execution platform provides a development model and a set of non-functional services that can be used (often through annotations) by the applications. Generally, execution platforms handle a limited number of such services like, for instance, persistence, security or remote management, on behalf of the applications. Making the distinction between the execution platform and the hosted applications simplifies the complexity of development, debugging and administration. Pervasive platforms are usually based on the principles of Service-Oriented Computing (SOC) paradigm [10] where loose coupling between components enables dynamic adaptation and runtime evolution. This architecture level adaptability is crucial in pervasive computing in order to drive seamless adaptations when the environment changes.

Current platforms are today able to provide very effective technical services. For instance, GatorTech [11] extends the OSGi framework with heterogeneous device access services. This is also the case of for PCOM/Base [12] or RoSe [13]. Other platforms introduced autonomic features. For instance, AMUSE [14] and ACCORD [15] propose specific component models to build autonomic applications.

Integrating context management in pervasive platforms is a crucial and challenging task. Context can be defined as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application." [5]. Entities are generally classified in the following categories:

- Computing environment, including available memory, network, connectivity, devices, etc.
- User environment, including current location, needs, situation, activity, preferences, etc.
- Physical environment, including temperature, luminosity, ambient noise, humidity, etc.

Entities are domain specific. Indeed, context is developed for a unique purpose: building applications. Previous works [16] [17] [18] have identified the main concerns that context middleware must handle and current techniques:

- Context gathering: There are varieties of data sources to be exploited with specific acquisition processes (single/double way) and synchronization mechanisms.
- Context modeling: Context is primarily a model of the environment. Many methods can be used to represent knowledge such as Key-Value, OO model, logic based model ontology or hybrid approach that combine precedent previous approaches.
- Context processing: To ease application development, low-level entities like device state are not sufficient. Operations like inference, mediation, aggregation, or enrichment are needed to get higher-level information.
- Context dissemination: Applications need to retrieve and interact with context information through queries or programming APIs for instance.

However, implementing these facilities involves highly error-prone technical code from low-level synchronization to high-level mediation or enrichment operations.

III. MOTIVATING SCENARIO

Let us introduce our motivating scenario, based on iCasa[1] [19]. The iCasa environment is made of three tools:

- An Integrated Development Environment based on an eclipse plug-in. It supports the development of iPOJO-based application.
- An execution platform, based on OSGi and RoSe, running on a home gateway, which hosts several applications and offers dynamic deployment facilities.
- A smart home simulator (see Fig. 1) that supports the execution of predefined scenario, in order to quickly test pervasive applications.

Fig. 1. iCasa simulator

In this environment, several applications belonging to domains like safety, comfort or healthcare have been developed. These applications vary a lot in terms of technical requirements and needed context. For example, iCasa hosts part of a home care application called actimetrics which measures and analyses the motor activity of elderly. Its purpose is to track behavioral changes to early diagnosis degenerative diseases like Alzheimer [19].

For our motivating scenario, we will consider an application which runs only on the home gateway in order to focus on the capabilities delivered by the platform itself. We designed a simple *Light Follow Me* application: for each room of a house, it turns the lights off and on depending on the presence of a user, and adjusts light intensity as a function of the moment of the day (morning, afternoon, evening or night).

The context needs of the *Light Follow Me* can be decomposed as it follows. First, the application relies on appropriate abstractions of lightning devices, spatial zones and time (*modeling*). Such abstractions must be synchronized with the physical devices (*gathering*). Then, the application requires knowing which device is in which zone (*enrichment*). The application computes physical parameters: presence and illumination per zone (*processing*). Finally, the application necessitates retrieving all this information (*disseminating*) and applying its business logic to affect the environment.

In this paper, we show how to clearly separate context and application during development and execution phase. We also demonstrate how our tool helps the definition, deployment, and execution of a context program module on top of iCasa.

[1] http://self-star.imag.fr

IV. Proposal Overview

We have defined and implemented a context management framework on top of iCasa, our operational pervasive platform. Its purpose is twofold. First, it provides assistance to developers for context definition through a well-defined set of abstractions and associated code. Second, it supports the execution of this context in dynamic environments. Most non-functional aspects are kept hidden from application developers to help them focus on business code. Generic technical details of implementation, like synchronization, are simply specified with annotations and corresponding code is integrated at build time.

In our approach, a context is a dynamic program module provided by the pervasive platform and used through API by the applications running on top of the platform. This is in contrast with the "database vision" of context where all possible information is collected and kept in a storage facility and made available through queries.

As illustrated by Fig. 2, we are defending a two-step approach. First, domain engineers define a context that can be used by a set of applications that are expected to run on a same platform. Applications developers use this context in order to simplify their code and concentrate on business logic. Specific tooling based on model-driven engineering [20] is provided for that purpose. Second, context is transformed into an iCasa program module and is dynamically executed. This module relies on service-oriented computing, enhancing flexibility.

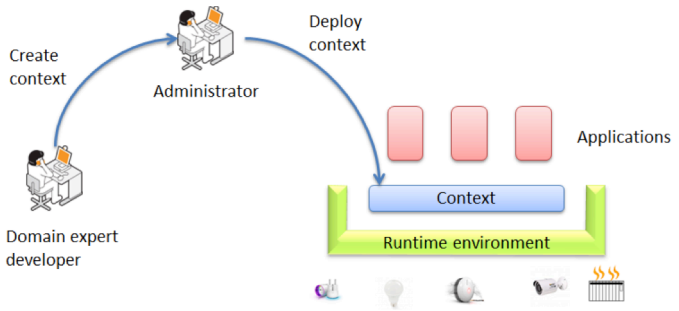

Fig. 2. Overall apporach

The first step, called Domain Engineering, can actually be complex and may involve several persons with different skills. The tasks to be performed are the following:

- Identify and determine the necessary concepts to be included in the context.
- Define the context *per se* (entities and relations between them) in conformance with a meta-model.
- Select and integrate the communication drivers to access real devices at runtime.

The execution platform relies on a domain-specific service-oriented component model, iPOJO [21]. Context entities and relations are modeled as components. Properties of components represent the state of the associated entity. Code in charge of gathering information from the environment, and keeping the corresponding model synchronized, is encapsulated in each component. Each component can be exposed as a service, allowing easy dissemination of context information to applications. Context relations are represented as links between entities.

V. Proposal Details

A. Design-time support

Context is a representation of the surrounding environment. As such, it can be regarded as a model [20] of the environment used by the pervasive applications. In our approach, context runtime management is delegated to the pervasive platform. The context model must then be made explicit, and kept separated from the application business code.

Context is defined by domain experts. It must conform to a meta-model defining the different elements of a context and their possible relationships. This meta-model (presented in Fig. 3) encompasses the different concerns (as depicted by the enclosed areas in the figure) that must be handled by the platform.

Specifically, the core of the meta-model concerns context modeling, and is based on two main concepts:

- *Entity* represents a concept of the context, characterized by a finite set of state properties.
- *Relation* represents a binary relationship between entities in the context. This relation allows the context to be navigated by applications. It also enables to extend the definition of an entity through the extraction of state property values from a related entity.

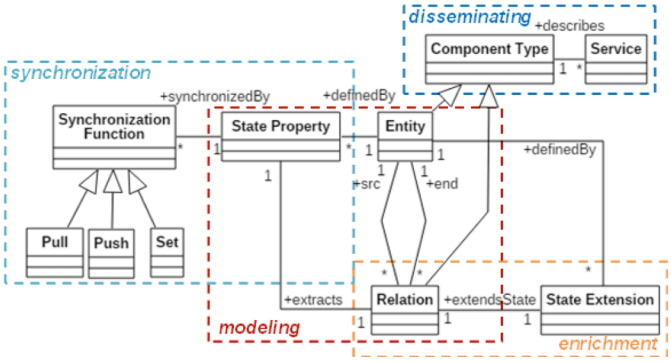

Fig. 3. Context metamodel

After designing the conceptual model, made of entities and relations, a domain expert has to specify how this model has to be synchronized with the real environment. This is achieved by associating synchronization functions to the properties of the defined entities (as shown in the left part of the meta-model in Fig. 3). Synchronization functions are programmed in Java, and encapsulated in the code of the component corresponding to the entity.

To support these design-time activities, our approach is based on a domain-specific specialization of the iPOJO component model [21]. Our proposed meta-model is in fact an extension (see the top right part of Fig. 3) of the iPOJO meta-model, with the specific concepts required to deal with context.

The concrete language used in iPOJO to define components is based on plain Java classes marked with annotations to declare non-functional aspects handled by the platform, see for

illustration the component definition in Fig. 4. We have enhanced the iPOJO design-time tools to handle the specific annotations required to declare specialized context concepts (for example, annotation `@ContextEntity` in Fig. 4 allows specifying a context entity with a set of state properties).

Sensors/actuators are represented as services. This is transparently managed by iPOJO and RoSe [13], as we will see in the implementation section. These facilities, along with the use of annotations to declaratively define the components, enable to keep the code at an abstract level, close to the conceptual problem space. Moreover, because context entities are themselves components, they can provide business services that can be easily discovered and consumed by the applications, thus facilitating context utilization.

Notice that facilities to actually instantiate the component, publish the provided service in the service registry, discover and select the required services are automatically provided by the iPOJO framework at runtime, without cluttering the application code.

Let us illustrate this on the *Light Follow Me* scenario. To model a lighting device, we can define an entity characterized, in particular, by a serial number and a property indicating the current state (on or off).

```
1 @Component
2 @Provides(specifications = ContextEntity.class)
3 @ContextEntity
4    @State({
5              BinaryLightContextEntityImpl.DEVICE_TYPE,
6              BinaryLight.DEVICE_SERIAL_NUMBER,
7              BinaryLight.BINARY_LIGHT_POWER_STATUS,
8              BinaryLight.BINARY_LIGHT_MAX_POWER_LEVEL
9    })
10
11 public class BinaryLightContextEntityImpl implements
12    fr.liglab.adele.icasa.context.model.ContextEntity,
13    DeviceListener<BinaryLight> {
14
15    @Requires(id = "context.entity.device",
16    filter ="(device.serialNumber=${context.entity.id})")
17
18    BinaryLight device;
19
20    @Pull(state = BinaryLight.BINARY_LIGHT_POWER_STATUS,
21              period = 30, unit = TimeUnit.SECONDS)
22    private final Function<String,Boolean> syncStatus =
23    (String property)-> { return device.getPowerStatus();};
24
25    @Set(state = BinaryLight.BINARY_LIGHT_POWER_STATUS)
26    private final Consumer<Boolean> propagateStatus =
27    (Boolean newStatus)-> {
28              if (newStatus) { device.turnOn(); }
29              else { device.turnOff();}
30    };
31
32    @Override public void devicePropertyModified(
33              BinaryLight device, String propertyName,
34              Object oldValue, Object newValue) {
35              pushState(propertyName, newValue);
36    }
37 }
```

Fig. 4. Java code for a context entity component

The corresponding iPOJO component is shown in Fig. 4: the component is implemented by class `BinaryLightContextEntityImpl` (marked as a `@Component`, and concretely a `@ContextEntity`) that provides (line `2`) and requires (line `15-18`) services (in this case, we require a reference to the physical actuator that allows to control the light). The example also shows a `@Pull` synchronization function (line `20-23`) that will be invoked periodically (every 30 seconds) to keep the context property `BINARY_LIGHT_POWER_STATUS` updated with the last value from the physical sensor. Similarly, it shows a `@Set` function (line `25-30`) that will be invoked each time an application changes the value of the state property, to propagate it to the environment, using the appropriate physical actuator.

Likewise, spatial zones can be represented by another entity. These entities can further be enriched by adding relations. For example, relating a light with a room will provide an additional location state for the light. The main interest of clearly separating entity and relation is to enhance the extensibility of our system. At runtime, the platform will dynamically build a graph of instances of the defined components to represent the current state of the environment.

B. *Runtime support*

Our framework has been designed to be integrated in the operational pervasive platform iCasa [19]. iCasa supports the execution of dynamic Java applications, on top of the OSGi service platform and the iPOJO service-oriented component model [21]. The iCasa infrastructure offers various technical services necessary to develop pervasive applications, like task scheduling, event management or autonomic managers. However, iCasa currently focuses only on developing pervasive applications, and does not provide specific tools to handle dynamic and extensible context management.

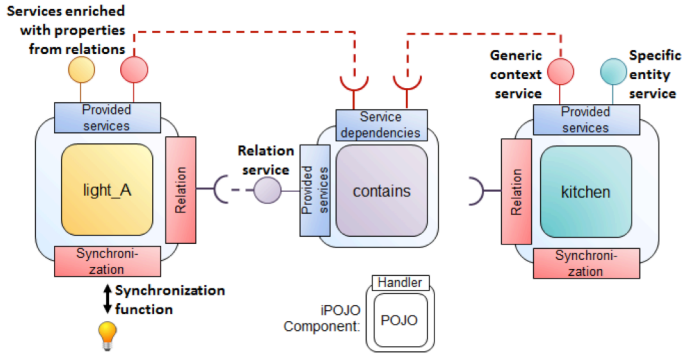

Fig. 5. Simplified example of context model – Component and service view

Because context components are iPOJO components, they can be directly packaged and deployed to the iCasa platform. However, the iPOJO runtime must be extended to handle the newly added context-related functions. The iPOJO framework is based on the concept of runtime containers, in charge of the life-cycle of components, as depicted in Fig. 5. Containers use bytecode instrumentation to take full control of the execution of the component: they can, for example, instantiate objects, intercept method invocations or field accesses.

Component containers can be extended by iPOJO modules, called handers. The framework provides a number of off-the-shelf handlers in charge of global aspects of the platform, like service publishing or dependency injection. More importantly, as shown in Fig. 5, it is possible to develop new iPOJO handlers to take charge at runtime of particular concerns.

We implemented two new iPOJO handlers dealing with specific context concerns. The handlers are described as follow:

- *Synchronization Handler*: It is in charge of dealing with the state synchronization of entity components. It keeps the state properties up-to-date by managing the synchronization functions. Different strategies can be specified to do so. For example, the handler can periodically call *pull* functions or just wait for *push* callbacks to keep the state up-to-date. Additionally, the handler is in charge of publishing state properties as service properties. This publication has two main interests. It allows processing of more advanced filters and state updates can be reported to the application without the burden of an Observer pattern, by relying on iPOJO notification mechanism.
- *Relation Handler*: It is in charge of dealing with relations and extending properly the state of an entity. It tracks and filters all the relations targeting its attached entity component. If an instance of entity disappears (e.g. a light is removed from a house), associated relations and linked state extensions are automatically invalidated and deleted.

VI. Implementation and Application

In order to validate the framework, we developed the *Light Follow Me* application; it corresponds to the scenario previously described in part III. Fig. 6 shows an overview of the application on the iCasa simulator. This is a typical pervasive scenario in which the light must "follow" the user (simulation shown in the left part of Fig. 6). The idea is simple but the implementation fulfilling pervasive needs is complex. Indeed, it is not an easy task to model the relevant concepts as a whole and not independently.

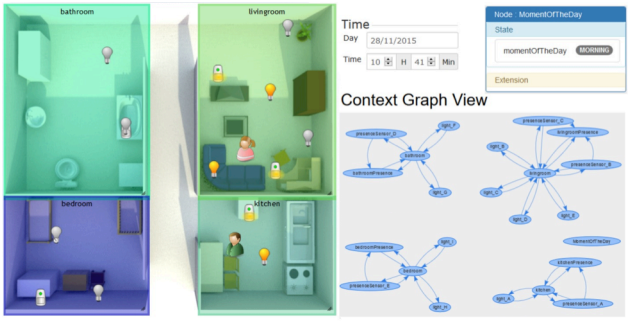

Fig. 6. *Light Follow Me* application – iCasa simulator overview

Our framework gives an understandable representation of the global context. The bottom right corner of Fig. 6 presents the graph of the model displayed on the iCasa simulator: devices are gathered by location, each room is enhanced with a physical parameter aggregating and synthesizing presence status, and an independent entity provides the moment of the day. In addition of the graph view, the simulator can display state properties and state extensions of an entity. This functionality is shown on Fig. 7 with light_A. Both reflect data at runtime.

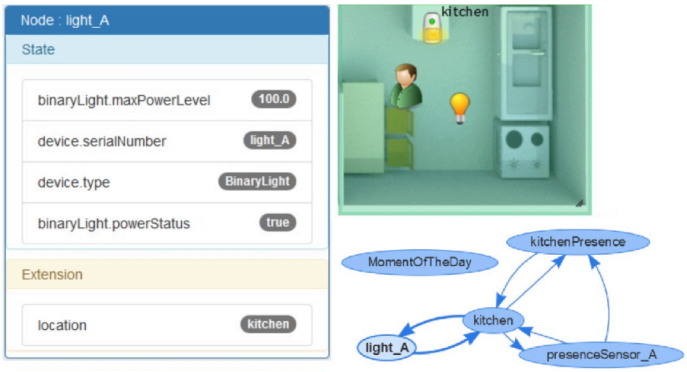

Fig. 7. *Light Follow Me* application – zoom on the kitchen

Our framework gives significant results from application development perspective (see TABLE I). Inevitably, formally building the context adds an additional development task and the resulting architecture is more complicated. However, it's easier to implement applications on top of it. The context is shared among applications, is extensible, and can be adjusted to fit new needs. The whole software is more consistent, testable and maintainable.

TABLE I. *Light Follow Me* Application Implementation Statistics

Implementation Statistics	Application impl.		Number of LOC[a] for the moment of the day impl.
	Total number of LOC[a]	Number of LOC[a] per component	
Initial impl.	829	581	95
Impl. with our framework	623	217	107
Framework improvement	25%	62%	-13%

[a] LOC: Lines Of Code

VII. Related Work

Developing context-aware applications is one of the hot research topics for the last decade. Naturally many software architectures emerged to reach this goal. We compare our proposed architecture to the existing ones: The Context Toolkit [5], COSMOS [7], SOCAM [8] and another SOC-based context model [9]. Many more architectures are available, but these ones are representative of the global trend.

The Context Toolkit promotes code-reuse through the composition of distinct artifact called widgets to build the context. These widgets are used to hide the complexity of sensors and abstract context information in a suitable way to fit applications need. These reusable blocks are explicitly linked at design-time, each block deciding which blocks to use. Our approach is similar in the sense that we divide the context in individual small pieces. The key differences with our work are that we delegate the composition at runtime with more variability expressed at design-time thanks to SOCM properties. Moreover our entity relation like model offers more flexibility to design complex context.

COSMOS, COntext entitieS coMpositiOn and Sharing, is a component based context middleware. Each pieces of context is reified as a component called Context Node organized in a

hierarchical structure. This approach and ours address the separation of concerns by offering several built-in mechanisms like push/pull notifications and reduce the developer's work. However, the strictly hierarchical approach of COSMOS context makes difficult to model it with horizontal relation. Moreover, component specifications are strictly defined at design-time, so runtime extensibility proposed by our system of relation will be hard to achieve.

The Service-Oriented Context Aware Middleware (SOCAM) is an ontology based context middleware. SOCAM architectures relying on several components: Context Providers (extracting context from internal and external data sources, and converting them in ontological instance), Context Interpreter (reasoning engine performing inference to extract high-level context and store it in knowledge base), Context-aware Mobile Service (application that consume context), and Service Locating Service (a registry where provider and interpreter are registered, where other components can search specific providers or interpreters to fit their needs). SOCAM envisions a highly structured context model with ontology in order to benefit from all the powerful processing tools induced by this approach, like reasoning engine. So it generates a programming model through a query language and rules, contrary to our programming model that relies on Java services specifications that we consider more adapted to develop added-value services.

In [9], a work dealing with proactive adaptation and context management based on a SOCM architecture is presented. 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 the context through actuators. Our approach, in this sense, is very similar because previous works say little about how to influence context. However, to achieve this goal, a specific query language that generates a cost on the learning curve is provided, whereas we prefer a traditional Java programming model.

VIII. CONCLUSION

In this paper we presented a comprehensive approach to build and run context that specifically addresses fundamental concerns like gathering, modeling, processing and disseminating information. This solution can be integrated in an enriched execution platform as demonstrated. Our work focuses on providing tools to build and execute a runtime extensible model of context. This model is synchronized with external entities and can be enriched dynamically by new relations or entities. Applications developed using SOC paradigm upon this model can also dynamically add new elements in the context in order to better fit their needs.

So far, we have focused on feasibility of the approach, so a number of limitations remain, especially at the implementation level. While the model is extensible at runtime, applications can only access extended entities using a generic API, losing all usability advantages of Java strong typing. Moreover, our runtime implementation currently maintains a possibly large, in-memory, representation of context. Additional performance optimizations are required to cope with the needs of realistic applications, in terms of memory scalability and footprint.

Further perspectives of our work include extensions to handle more complex synchronization scenarios. Particularly, the platform should support developers to handle automatic creation of application-specific entities and relations. We argue that the usage of software engineering tools, like the proposed domain-specific context language, can provide an adapted support for this case.

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] F. Mattern, "The vision and technical foundations of ubiquitous computing," in Upgrade European Online Magazine, pp. 5-8, 2001.

[4] L. Atzori, A. Iera and G. Morabito, "The Internet of things: a survey," in Computer Networks, vol. 54, pp. 2787-2805, October 2010.

[5] A. K. Dey, "Understanding and using context," in Personal and ubiquitous computing, vol. 5, pp. 4-7, 2001.

[6] V. Issarny, M. Caporuscio and N. Georgantas, "A perspective on the future of middleware-based software engineering," in 2007 Future of Software Engineering, pp. 244-258, May 2007.

[7] D. Conan, R. Rouvoy and L. Seinturier, "Scalable processing of context information with COSMOS," in Distributed Applications and Interoperable Systems, pp. 210-224, January 2007.

[8] T. Gu, H. K. Pung and D. Q. Zhang, "A service-oriented middleware for building context-aware services," in Journal of Network and computer applications, vol. 28,pp. 1-18, 2005.

[9] S. VanSyckel, G. Schiele and C. Becker, "Extending context management for proactive adaptation in pervasive environments," in Ubiquitous Information Technologies and Applications, pp. 823-831, 2013.

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

[11] S. Helal et al., "The gator tech smart house: a programmable pervasive space," in Computer, vol. 38, pp. 50–60, 2005.

[12] C. Becker, M. Handte, G.Schiele and K.Rothermel, "PCOM – a component system for pervasive computing," in Pervasive Computing Communications, pp.67-76, 2004.

[13] J. Bardin, P. Lalanda and C. Escoffier, "Towards an automatic integration of heterogeneous services and devices," in Service Computing Conference, pp. 171-178, December 2010.

[14] E. Lupu et al., "AMUSE: Autonomic management of ubiquitous e-health systems," in Concurrency and Computation: Practice and Experience, vol. 20, pp. 277–295, 2008.

[15] H. Liu, M. Parashar, and S. Hariri, "A component-based programming model for autonomic applications," in Autonomic Computing, pp. 10-17, 2004.

[16] S. Lee, J. Chang and S. G. Lee, "Survey and trend analysis of context-aware systems," in Information-An International Interdisciplinary Journal, vol. 14, pp. 527-548, 2011.

[17] C. Perera, A. Zaslavsky, P. Christen and D. Georgakopoulos, "Context aware computing for the internet of things: a survey," in Communications Surveys & Tutorials, vol. 16, pp. 414-454, 2014.

[18] C. Bettini et al., "A survey of context modelling and reasoning techniques," in Pervasive and Mobile Computing, pp. 161-180, 2010.

[19] P. Lalanda, S. Chollet, C. Aygalinc and E. Gerbert-Gaillard, "Service-based architecture and frameworks for pervasive health application," in Emerging Technologies & Factory Automation, pp. 1-8, 2015.

[20] E. Seidewitz, "What models mean," in IEEE software, pp. 26-32, 2003.

[21] C. Escoffier, R. S. Hall and P. Lalanda, "iPOJO: an extensible service-oriented component framework," in Service Computing, pp. 474-481, 2007.