Manuele Kirsch Pinheiro
Carine Souveyet
Philippe Roose
Luiz Angelo Steffenel *Editors*

# The Evolution of Pervasive Information Systems

Springer

*Editors*
Manuele Kirsch Pinheiro
Center for Research in Informatics
Pantheon-Sorbonne University
Paris, France

Carine Souveyet
Centre de Recherche en Informatique
Pantheon-Sorbonne University
Paris, France

Philippe Roose
Université de Pau
Anglet, France

Luiz Angelo Steffenel
University of Reims Champagne-Ardenne
Reims, France

Uncorrected Proof

ISBN 978-3-031-18175-7 ISBN 978-3-031-18176-4 (eBook)
https://doi.org/10.1007/978-3-031-18176-4

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Edge Computing and Learning

**Philippe Lalanda**

Uncorrected Proof

## 1 Introduction

The massive deployment of intelligent and communicating objects in our living environments generates immense quantities of data, unmatched in the past (Becker et al. 2019). This data, collected on the field, provides past, present and future information on physical phenomena and running infrastructures, but also on activities and behaviour of people. This information, which can be sensitive and private, must be secured so that it cannot be acquired by parties not entitled to it. This is for example the case for data related to individuals, such as medical data or location data, but also for confidential data concerning organizations, whether private or governmental.

Many of these smart objects are now connected to information systems through successive gateways and networks. This makes it possible to set up new high value-added services allowing, for example, to bring more flexibility to production activities, to better monitor the health of patients, or to set up more intelligent living spaces like smart cities, smart homes, or smart stations. It is safe to assume that most leading-edge companies could no longer do without this information.

Today, information systems are often located in cloud data centers. This implies constant and costly communications between connected objects and data centers that are usually located at a great distance. This architectural solution does not scale well and is challenged by the massive increase in data to be transported and stored in the cloud. Also, it appears that, in many cases, data is badly identified and looses its relevance rather quickly. Essentially, the data is kept for fear of losing important information. This only further clutters the communication channels and storage devices.

P. Lalanda (✉)
Grenoble University, Grenoble, France
e-mail: philippe.lalanda@univ-grenoble-alpes.fr

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

M. Kirsch Pinheiro et al. (eds.), *The Evolution of Pervasive Information Systems*,
https://doi.org/10.1007/978-3-031-18176-4_5

In fact, with the installation of tens of billions of connected objects, it is simply not appropriate to upload all the data that can be collected on the field to cloud data centers. There are several reasons for this. First, this architecture leads to a central collection of data, which is problematic for privacy-sensitive users. Also, this approach is very expensive. For instance, running a single Amazon EC2 instance of type "a1.2xlarge" with 8 cores and 16 GiB RAM costs 0.23 dollar per hour. Finally, in addition to that, recent trends in pervasive computing, like augmented reality and machine learning for instance, increase the number of applications that require fast processing of large amounts of data captured by devices. Analysing this data in the cloud does not meet the response time requirements of these applications due to the high communication latency between the devices and the cloud infrastructure (Sarkar et al. 2018). The same applies for highly interactive applications or games that require fast responses (Choy et al. 2012).

This major evolution, in both the amount of data and the application needs, has marked the end of all-cloud solutions and the inevitable emergence of more decentralized architectures where computations are done near the data sources. This new architectural approach is based on edge computing (Shi and Dustdar 2016; Garcia Lopez et al. 2015; Vaquero and Rodero-Merino 2015; Varghese et al. 2016), which refers to all the enabling technologies allowing computation to be performed at the edge of the network. It can be defined as any computing and network resources along the path between data sources and cloud infrastructures (Shi and Dustdar 2016; Shi et al. 2016). Typical resource providers in edge computing environments are electronic boxes of various types, personal computers, HMIs, smartphones, and even more powerfull "regional" servers, attached to cellular base stations for instance, that are in the close vicinity of the data sources. As we will see in more details, edge computing improves latency and security. It also leads to a better utilization of existing hardware (e.g., unused office PCs) (Hasan et al. 2015; Miluzzo et al. 2012) and economic benefits for both consumers and providers (Miluzzo et al. 2012; Fernando et al. 2012).

The aim of this chapter is to study this trend towards more distributed architectures and to study the associated software challenges. It is also to present a second major trend, namely the use of AI techniques on the edge. We will see that the effects of this second major evolution are profound. They call into question the software techniques currently used for the development and deployment of pervasive applications. This chapter is organized as follows. First, the notion of edge is defined and illustrated with examples from different pervasive spaces. Then, the main challenges related to the implementation of pervasive applications on the edge are presented. The solutions provided by the current pervasive platforms will then be detailed. Finally, we will turn to the notion of machine learning on the edge. We will show the main challenges at the different stages of the application lifecycle and outline some solutions currently being explored.

Uncorrected Proof

## 2 Edge Computing in Pervasive Computing

### *2.1 Principles and Examples*

As introduced, edge computing is not a technique per se, but rather an architectural pattern that aims to decentralize calculations and data management as close as possible to the field devices. This decentralization is done opportunistically according to the available machines in the vicinity of the data sources and to the available networks. This results in very varied and often heterogeneous software architectures. The availability of appropriate resources is highly dependent on the domains, but also on technological opportunities. For example, in the Telco domain, the massive deployment of cellular based stations has made possible the development of MEC (multi-access edge computing) where new services for specific customers or classes of customers are now installed.

The use of edge computing is now a reality in many areas such as smart homes, smart buildings, smart cities, smart cars, or smart manufacturing. Let us take two examples to illustrate this point. Let us start with the smart home domain which is informative and illustrative. A smart home is a house or an apartment equipped with electronic devices providing sensing and actuating capabilities. Although most existing homes were not designed to be smart, the emergence of low consumption wireless devices has allowed the easy instrumentation of homes and the development of increasingly intelligent services. These services, however, are characterized by stringent requirements regarding security, response times or interaction facilities. For example, the detection of a fall must be done as soon as possible and should not be revealed to unauthorized persons. Also, in general, it is often necessary to have smooth interactions with the inhabitants to allow a good acceptance of the technology, which implies very short response times.

There is actually no shortage of computer resources in smart homes. Precisely, depending on the configuration, the edge infrastructure can include the following elements (see Fig. 1):

- smart communication-enable devices capable of hosting significant pieces of code like some connected cameras for example,


**Fig. 1** Smart Home Computing resources

Uncorrected Proof

- electronic appliances like Internet or TV boxes. Such boxes host more and more applications related, for instance, to home automation or to the safety or well-being of people.
- vocal assistants are getting more and more popular and can also host applications just like boxes.
- laptop or desktop computers. Such computers can also run various comfort, security or automation applications. WiFi signals between Internet boxes and personal computers can even be used to detect movements or identify people.
- smartphones and tablets. Such devices, more and more powerful, are able to run the above mentioned applications but also resource intensive games or augmented reality functions.
- telecom dispatchers located in the close neighborhood. Such equipment is regularly upgraded by the operators to accommodate more and more network functions but also applications. They can, for example, perform added-value analytic on the electrical, gas or water consumption in a house.

It should be noted that edge computers are not just about CPU performances. They must integrate various communication protocols, wired or wireless, and advanced middleware to support the deployment, execution and adaptation of applications. We will come back to this fundamental point later in this chapter.

Let us now turn to a second example, Industry 4.0. The purpose of this initiative is to bring together new technologies and production processes to enable the emergence of smart, connected manufacturing. The term, coined in 2011 by a government-funded German project, refers to what could be the fourth industrial revolution (Forschungsunion 2013). Industry 4.0 envisions new production techniques, new materials, and the generalized adoption of digital technologies. We focus here on the latter point, and in particular on the adoption of dense and pervasive networks of sensors and actuators in industrial environments in order to complement existing devices and systems. The generalized use of sensors/actuators allows the integration of field devices controlling operations on a plant floor and supervision systems, usually located in IT facilities. Among many benefits, such integration allows the systematic oversight and improvement of production activities and resource management and an overall increase in safety and sustainability of production systems. It also leads to more flexibility in production processes, which can be reconfigured to meet changing demands and enable novel products.

Given this new context, the notion of edge is now prevalent in manufacturing production environments. These environments are characterized by a great heterogeneity, but we can nevertheless identify some common computing nodes that can host edge-type functions (see Fig. 2). This is the case for example of the following elements:

- desktop computers. Such computers are usually quite numerous in the plant floor and not used at full power. They can house various analytic allowing to follow and possibly adjust the processes in progress.
- smartphone and tablets. These devices are now heavily used in the plant floor, in particular to run augmented reality applications in order to get information without opening any industrial box for instance.

Uncorrected Proof


**Fig. 2** Smart Manufacturing resources

- HMI devices. These appliances are very present in production environments and constitute prime computing and storage resources. They are often quite powerful and, above all, are usually connected to a number of production equipment via integrated fieldbus connectors.
- IT servers. Several major software packages, such as MES (Manufacturing Execution System), ERP (Enterprise resource Planing) or SCADA (Supervisory Control and Data Acquisition), run on servers located near the automation islands, usually in the same building. These servers are increasingly used to deploy new software functions.

It is very interesting to note that the new sensors integrated on the plant floor must not interfere with the control functions. They provide different information that allows, for example, to set up analytics or to better configure manufacturing devices (Lalanda et al. 2017), but they are not intended to replace the current controllers like PLCs (Programmable Logic Controllers) for instance.

There are many other areas today where edge infrastructures are put in place successfully. For instance, autonomous vehicles rely more and more on edge resources in order to process big amount of data in real time. Data collected from in-vehicle sensors are processed locally in order to reduce latency and to deal with intermittent network connection. Also, autonomous vehicles are able to opportunistically use edge servers located in the vicinity to offload some computing or receive software updates. In another very popular domain, content delivery networks have deployed servers situated close to the users in order to allow rapid loading of websites and to support smooth video streaming.

Uncorrected Proof

## *2.2 Terminology*

The great diversity of the domains and their use of the edge have led to the introduction of particular terminologies. In particular, the terms device edge, far edge and near edge are often used today:

- The Device edge, also called the IoT (Internet of Things) edge, includes low to medium power computers, directly connected to sensors/actuators by non-Internet protocols like field buses or WiFi. They correspond for instance to Internet boxes and smartphones in a smart home, or HMI and tablets in a smart plant. These are the elements closest to data sources and are usually provide ultra low latency and high throughput. Due to their geographical location, they provide optimal conditions for managing data security.
- The Far Edge, also called user edge, corresponds to powerful computing infrastructures that are deployed purposefully in the vicinity of the data sources, typically within a distance of some kilometers. They usually provide low latency, good throughput and also high scalability. For example, telecom companies may deploy far edge facilities in the cell phone towers they own, near the mobile base stations. Such infrastructures can also be found in large factories, big shopping centers or smart buildings. Far edge computers are designed to run resource-intensive applications that are specific to the location in which they are deployed.
- The Near Edge, also called enterprise edge, corresponds very powerful computing infrastructures that are deployed in a central location, usually connected to one of several Far Edge facilities. In fact, they look like small data centers designed to run generic services needed by the applications deployed on Device or Far edges. They generally provide very large computing and storage capacities.

It clearly appears here that the notion of edge covers a great diversity of locations and technologies. Edge infrastructures are actually installed in order to meet the needs but also to the financial resources of the owner companies. Typically, Near Edge infrastructures are only set up and managed by large companies that can afford to own them. Gaming, Live video streaming applications are good examples of applications that require the installation of advanced infrastructures both in the Near and Far edges.

## 3 Edge Pervasive Applications

Executing services at the edge and managing their life cycle is however challenging due to the dynamic, heterogeneous, and stochastic nature of the pervasive computing environments. This is further complicated by the fact that some edge computers have limited resources that must be managed explicitly. In addition, developers and administrators can no longer benefit from advanced services provided by cloud infrastructures and must use *edge-specific* tools that are often less sophisticated. As we will see here, there is nevertheless a rapid and significant improvement in these tools, often inspired by those previously developed for the cloud.

Unconfected Proof

## *3.1 Challenges*

### 3.1.1 Application Design

A first challenge, of course, is the definition of distributed architectures where computation and data management are appropriately distributed between devices, edge resources and cloud data centers. Such architectures are dependent on the targeted applications and are difficult to generalize from one domain to another. Also, the optimal distribution of computational and data oriented components may evolve over time. Today, there are a number of research works around the dynamic and autonomic distribution of these architectural elements along a device/edge/cloud continuum (Shi and Dustdar 2016).

Another issue is the design and implementation of applications, or parts of applications, that are intended to be placed on edge machines. They are usually executed on heterogeneous computers, often limited in terms of resources. This imposes strong constraints on the development teams regarding the programming language, the COTS (Components Off The Shelves) and the architectural patterns that can be used. For instance, a language as popular as Java is not well suited to constrained environments because of its heavy software stack. Similarly, some communication middleware can be too costly in terms of memory footprint or CPU required. These constraints concerning the tools and patterns that can be used also extend to deployment, updates and monitoring activities.

### 3.1.2 Application Security

Security is a major concern for any pervasive system. It is true that, by nature, edge computing considerably limits the transfer of data to remote servers, and thus significantly reduces the risks of interception or tampering. But, in a way, the security problems are partly transferred from the network to the edge computers, which can result in centralized hotspots for data security. These machines must be perfectly secure to counter any malicious attack. Remember that pervasive data, in a house or a factory, betrays the activities and habits of the occupants and, as a consequence, are highly prized.

Securing an edge computer, like a gateway for instance, is particularly costly and has a profound impact on all enclosing developments. It is indeed a matter of securing both the services provided by the computer (communication, middleware for execution, deployment, etc.) and all the applications. Concerning the latter, all the internal software components and their interaction must be secured. This requires specific architectural patterns and techniques, high-level skills and often leads to long development times. Nevertheless, despite all these difficulties, advanced security is a requirement that often comes first in many industries when it comes to deploying applications on the edge (Lalanda et al. 2021).

Uncorrected Proof

### 3.1.3 Application Data

Another major constraint is directly related to the quality, dynamism and heterogeneity of the data. Regarding quality, cleaning data is a critical first step since it can be incomplete, uncertain or noisy. Pre-processing data is thus necessary to obtain reliable and well presented data. In addition, some data is simply not available on all environments or it may come and go depending on the availability of certain sensors. Recovery mechanisms, sometimes complex, must be put in place at the edge or a the applications levels to avoid crashes when data is not or no longer available. This is generally more difficult than in the cloud where applications are not directly connected to data sources but rather to databases.

The notion of data model, that is the way data is structured and named, is also of major importance. In most domains, such as manufacturing, there are several standards for data models, pushed by competing vendors. The correspondence between data collected on a field bus, often identified by simple register numbers, and a structured, symbolic data model has then to be done for each application or for each edge machine. It is important to have tools that facilitate this rather technical operation which consists in aligning data from communication frames, in Modbus (see modbus.org) for instance, with higher level models, like OPC/UA for example (see opcfoundation.org).

### 3.1.4 Application Context

Abstracting is a fundamental technique in software engineering and is particularly important in edge computing. This allows applications to rely on high-level data that matches their computational needs, and thus facilitates their programming and management in general. Data models, introduced here before, provide a certain level of abstraction which nevertheless remains limited since they usually focus on a single device or on a single function. The notion of pervasive context, introduced in the 1990s by Schilit et al. (1994), allows to aggregate and extend information provided by different devices. Dey and Abowd define context 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, including the user and applications themselves" (Dey and Abowd 1999). Contextual information thus covers various aspects related to users (like localization, interaction, physiological parameters, or general expectations and preferences), environments (like temperature, pressure, or CO2 levels), and characteristics of the supporting computing infrastructure.

Collecting information to build a relevant, useful context and keeping it up to date has proved to be particularly complex. In most cases, it is necessary to integrate disparate devices, applications and network protocols in order to get the needed information. Also, as previously said, pervasive environments are highly dynamic and volatile. Devices can connect and disconnect without warning, quality of the network connectivity can dramatically vary, and failures can happen anytime on

Uncorrected Proof

various system levels. A number of context management platforms with specific properties have been introduced (see Becker et al. 2013 for instance) but research on that topic is still going on. An open question regarding context, and by extension data models, is to know whether its construction should be application-specific or not.

### 3.1.5 Application Placement

Edge environments can be complex and composed of several computers, including mobile ones. The question of placing applications on the right machine(s) is therefore paramount. Note that the placement of tasks has always been crucial in pervasive systems to meet requirements related to ed or energy consumption. For instance, the completion time of a remote execution depends heavily on the speed of the selected provider device. The choice of the offloading target happens either statically or dynamically. An example for static placement is for instance Clone2Clone (Kosta et al. 2013). In this approach, each provider serves a fixed set of devices. Most offloading approaches, however, perform dynamic placement, i.e., a separate choice of the executing provider for each offloadable part.

Dynamic offloading has been applied in a variety of distributed computing paradigms including cluster and grid computing. Effective placement in grid computing systems often requires some knowledge about context, which also makes sense in edge environments. Context-aware placement gathers context information about the surrounding computing environment and is therefore able to react to changes. A context-aware placement approach can consider a number of context dimensions such as provider performance (Jonathan et al. 2017) and reliability (Edinger et al. 2017), system load (Ousterhout et al. 2013), task complexity (Flinn et al. 2002), data size (Cicconetti et al. 2019) or bandwidth (Kosta et al. 2012). Motivated by these observations, several novel placement approaches like Voltaire (Breitbach et al. 2021) apply ML to benefit from past experience. The core advantage is that the placement approach adjusts itself to the current environment and learns from past performance to improve continuously. Based on several input features of an upcoming task, e.g., the corresponding application, the input data size, the parameters, and the current system load, the ML-based approaches estimate whether offloading is expected to be beneficial.

Uncorrected Proof

## *3.2 Pervasive Platforms*

The most common solution today to face the challenges previously mentioned is to use a specific platform, also called middleware, to host the applications. Such middleware provides a development language adapted to the targeted domain (pervasive in this case) as well as a set of technical services that are used, in more or less transparent ways, by the applications. These services can concern any of the

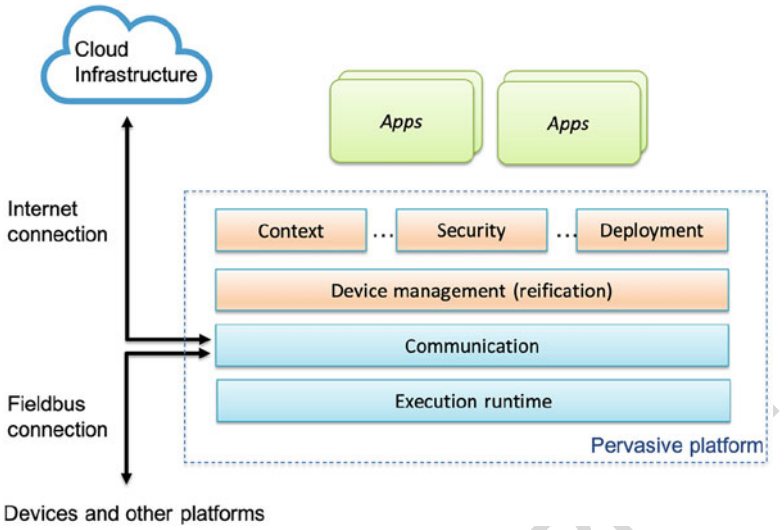


**Fig. 3** Pervasive middleware

above issues like communication, data management, security, self-awareness or data contextualization. Decoupling applications and technical services is a well-known approach in software engineering, referred to as separation of concerns, which has proven successful in many areas. It relieves programmers of complicated code by moving it to the platform. This approach is illustrated by Fig. 3 where platform-dependent services are built on top of the execution runtime. Various techniques can be used to implement those services.

In the pervasive domain, numerous platforms have been developed and evaluated. In the smart home domain for instance, let us mention iCasa (Lalanda and Hamon 2020), PCOM (Becker et al. 2004), AutoHome (Bourcier et al. 2011), DigiHome (Romero et al. 2013) or Microsoft's HomeOS (Dixon et al. 2012). Most of these platforms first dealt with the definition of a programming language for the applications adapted to the dynamicity of pervasive environments. Also, many of them strongly focused on the reification of devices, i.e. their transformation into objects that can be manipulated by the defined programming language. For this, the service approach has often been favored. The service-oriented computing paradigm promotes the development of applications through the late composition of independent and modular software elements, called services (Papazoglou 2003). Services are dynamically published on a registry and, thus, made available to potential consumers that can invoke them, possibly with a quality of service contract. In our case, devices are presented as services that come and go, and that can be used by the pervasive applications.

Another element brought by most platforms is some notion of execution context. This serves as an abstraction layer between the physical world and the applications run on the platform. Context provides information about the surrounding physical

environment but also the current state of execution of the running applications if needed. Some information is static, most of it is very dynamic though. The context module usually constitutes the only means for applications to interact with physical devices. Research on context-aware computing dates back to the 1990s. Schilit et al. (1994) introduced a still current definition of context and details of a comprehensive context management platform. Since then, a number of context management platforms with specific properties have been introduced. Various technologies have been developed, including tuples, object-orientation, databases with SQL-like approaches, or ontologies. Context can also be used implicitly, where the spatial distribution of data is implicitly used as context. More recently, contextual information has been represented as services (Lalanda and Hamon 2020). Here, context appears as a dynamic set of services. Depending on the availability of context sources and the applications needs, different services can be published and withdrawn.

Let us focus on the iCasa platform for illustration. A particularity of this platform is to use the service paradigm for many of its constituent elements. It is based on the iPOJO service-oriented component model (Escoffier et al. 2007), which in turn is based on the OSGi framework. Applications are all programmed in iPOJO and seamlessly interact with physical devices and remote platforms through services provided by the context module. Applications can be dynamically updated, completely or partially, by relying on the deployment mechanisms specific to OSGi and iPOJO. All the contextual information is presented as services. Precisely, the context appears as a dynamic set of iPOJO components. Context is dynamic in order to reflect the changing nature of the execution environment but also to deal with applications evolving needs. Context adaptations are carried out by an autonomic manager that follows the applications requirements and the available resources. Contextual services are then opportunistically and seamlessly used by the pervasive applications. The context module also tracks any change about a used contextual service and triggers a callback to alert consumer applications.

Finally, a communication manager (Bardin et al. 2010) reifies devices as iPOJO components. This manager supports an open set of protocols, including Zwave, Zigbee, X10, UPnP, DPWS and Bluetooth. Whenever a device disappears or a remote service becomes unreachable, the manager detects the disruption and removes the corresponding component. If the services provided by that component were needed by an application, the communication manager tries to find a new device or a remote service satisfying the needs (Roth et al. 2018).

## *3.3 Conclusion*

Moving applications to the edge has many advantages, but also raises significant technical and architectural issues. Many pervasive platforms have been proposed and are often distinguished according to their areas of focus. For example, the

Uncorrected Proof

platforms used for industry are often more robust and less dynamic than those used on smartphones or in smart homes.

Nevertheless, all these platforms offer interesting solutions for application, device, context and security management for instance. The service-oriented approach plays a major role in many cases and has proved successful. As we have seen, the service paradigm allows both to reify objects as services in pervasive platforms and to build modular and dynamic applications. It also facilitates interoperability between platforms by publishing their various functional capabilities as services (Roth et al. 2018).

Nevertheless, this service-based approach finds its limits when the amount of data to be processed increases. The service approach sees computing elements, including devices or contextual elements, essentially as service providers. The fact is that most of them are also data providers. In some domains, like smart manufacturing, they are even intense data providers. Work is then needed to allow interactions of almost continuous data flows between pervasive elements in service-based environments.

Uncorrected Proof

# 4 Machine Learning on the Edge

## *4.1 Principles*

In the last few years, Machine Learning (ML) based approaches have received a very strong interest for the development of pervasive applications. This can be explained by the difficulty to build models of dynamic phenomena in complex physical environments. The goal of an ML system is to train an algorithm to automatically make decisions by identifying patterns that may be hidden within massive data sets whose exact nature is unknown and therefore cannot be programmed explicitly. The growing attention towards machine learning stems from different sources: efficient algorithms, availability of massive amounts of data in pervasive environments, advances in high-performance computing, broad accessibility of these technologies, and impressive successes reported by industry, academia, and research communities, in fields such as vision, natural language processing or decision making.

The development of software systems based on learning techniques is however very different than the development of "traditional" software systems since the behavior of the system is not specified in the code by programmers but it is learned by a machine from data. To do so, two main workflows are set up. The first workflow, called model development, is primarily performed by data scientists. This workflow assumes that a business problem has been properly identified along with sources of historical data. During the first steps, raw historical data is analyzed, cleaned and transformed into appropriate numeric representations called features. Feature engineering is a complex task which purpose is to find out the most relevant data representations given the available data, the task at hand and the targeted model.

Features are then used to train a model with a carefully chosen machine learning algorithm.

Once a model has been developed, it is deployed on a target execution machine. This is the essence of the second workflow that is often implemented by software engineers. Its purpose is first to collect appropriate data and build features. The data is then used to make predictions. As the name suggests, predictions are only . . . predictions. They are founded on data which has not been seen during the training phase and can therefore be incorrect. Finally, collected data might be set aside in order to be used for further training.

The use of Machine Learning techniques opens the way for many new, more advanced applications on the edge. The expected added value motivates many manufacturers and service providers, in both the industrial and consumer sectors. But the development of these new applications raises formidable challenges that are not addressed by traditional software engineering techniques. ML-based applications are not developed in the same way as more traditional applications. They bring together different teams with new skills. Also they are built, tested, installed, configured, run, monitored and updated differently. In a word, they implement totally different life cycles.

## *4.2 A Variety of Actors*

First of all, it should be understood that ML-based components are usually only a small part of a system. They are often developed independently and integrated into a larger software architecture. Thus, implementing ML applications on the edge involves several different actors, with different cultures and skills. These actors develop complementary software components, learning-based and more traditional, that have to be assembled within an architecture. This architecture is usually deployed on a single edge machine, resulting in complex engineering work to allow interaction between programs of a differing nature. This generates important constraints on the edge platform in terms of execution mechanisms and technical services to be offered.

To illustrate this problem, let us take the example of analytic development in a smart plant (Lalanda et al. 2017). To do this, three complementary profiles, and usually three distinct teams, are needed:

- A team of software engineers who are in charge of managing the platform and programming specific code related for instance to data collection or data mediation (Morand et al. 2011; Garcia et al. 2010)
- A team of domain experts who are in charge of defining and developing the business-oriented analytics,
- A team of data scientists who are in charge of developing learning-based models and keeping them relevant and up to date. This includes selecting the needed data, creating the appropriate features, choosing the learning algorithm to be used, etc.

Uncorrected Proof

This diversity, if it is a source of value, is also a source of difficulties, misunderstandings and cultural gaps. These different actors use different environments, tools, processes and programming languages. On this last point, for instance, domain experts generally use Matlab for analytics, data scientists make use of Python to build models, and computer engineers user languages such as Java, Go, or C to develop business code or technical services.

### *4.3 A Specific Life-Cycle*

The development of "traditional" software systems is carried out according to activities that are now well defined: requirement management, design specification, implementation, and validation. Deployment then starts when a software system has been duly approved for delivery. It handles the transfer, installation, configuration, and integration of concrete artefacts therein. It also initiates the different executable components and deals with subsequent updates. Maintenance begins after the software initial installation. Its purpose is to modify the software being used in order to fix bugs, to improve quality of service, or to address new conditions for execution. Maintenance comprises a number of activities, ranging from the "simple" reconfiguration of certain parameters to more complex operations, like the development of new pieces of code or the migration to new running platforms. Simply put, system administrators, or autonomic managers, observe the systems at run time, change local configurations when needed, and otherwise send a request to developers if something serious happens.

The development of a prediction model takes a completely different turn because of the focus on data. Here, no requirements, design or coding, just data. The initial development uses historical data; its relevance depends on the quality of the available data and adequacy of the selected learning algorithm. In the case of pervasive applications, this data must come from the field, or from very high quality simulations, to be usable. Tests are done against a subset of the available data, set aside for this purpose. As illustrated by Fig. 4 (left part), it is an iterative process where the steps of data collection, feature selection and model training are repeated until a satisfactory result is obtained.

Model deployment activities look like traditional deployment tasks but are nevertheless significantly different (see Fig. 4, right part). First of all, the model is transferred to the execution (edge) machine and installed like any other software artifact. It is also configured to fit properly into its execution environment. Configuration can be complex and actually depends on lot on the services provided by the host middleware. In some cases, the model has to be further trained with local data in order to adjust to the specificity of the execution environment. Then appropriate data has to be collected in order to run the model, which can thus provide the expected predictions. The model is constantly monitored by system administrators, through appropriate tools. Finally, an update step completes the loop. The purpose of this

Uncorrected Proof

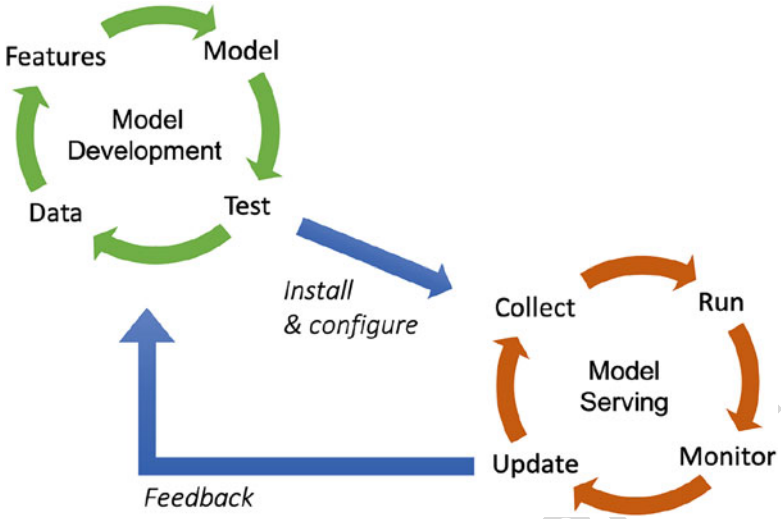


**Fig. 4** Life-cycle of ML-based components

step is either to update the model directly on site or to send data back to the data scientist teams to allow them to implement off-line updates.

A model is updated when a new one has been devised by data scientists, which happens fairly often. This can be done to enhance its performance, to improve its compactness, or even to extend its functional scope. Sometimes, this also comes with a change in the input data, which may require updates on data gathering activities. Finally, let us note that the models are extremely sensitive to data evolution, even very slight one. The smallest modification can invalidate a prediction model. Therefore, they must be continuously adapted to the execution environment and the corresponding data to stay relevant.

## *4.4 Conclusion*

The use of learning techniques on the edge is today more than a trend, it is a must. However, it raises major issues. It requires the creation of development teams with different and complementary skills, using different languages, processes and tools. Such heterogeneity is difficult to manage from a human point of view but also from a technical point of view. It requires to build architectures, and middleware, capable of integrating very different modules in order to form a single system.

Strong heterogeneity is also introduced in the management of the software components life cycle. Learning-based components have their own cycle, entirely data-driven, that is very different from the one of more traditional software

Uncorrected Pr

components. While the major activities remain the same, they are treated in a very different way and require, here again, specific skills and tools. All these differences call into question the current Software Engineering techniques and processes but also the support that has to be provided by the edge platforms.

Throughout the following section, we examine the challenges associated with each of the life cycle activities and outline possible solutions.

# 5 Challenges

## *5.1 Model Development*

To start with, a consistent and sufficiently large dataset is needed to create a relevant initial model. The data must be complete, well distributed, and clean. A low quality of data inevitably leads to models of little value.

It is therefore necessary to set up a structured and systematic data collection campaign. To provide real value, this data must be collected in the field, for example with some of the edge machines that will be used to run the models later. Such campaigns take time and effort and must be integrated into the development cycle. The most difficult part is to obtain data corresponding to extreme, problematic or even dangerous cases. Often, it is necessary to use simulation to do so, with all the risks that this entails for the actual quality of the data.

The choice of an adapted learning algorithm is also of major importance. It must take into account the specificity of the domain but also those of the targeted execution machines. As explained earlier, machines on the edge are often characterized by limited computing and storage capabilities. The learning models must therefore have a size in accordance with these limitations, which is often challenging since data scientists tend to build large models to get better performances.

## *5.2 Installation*

A learning model is usually only a (small) part of a larger software system. It is encapsulated in a software component and inserted into an encompassing software architecture. Within this architecture, it interacts with other components to implement higher-level functionality.

The installation of a learning-based component in a target environment can be done individually or with the architecture (and all other components). In the latter case, all components are compiled and linked together to create an executable. It is this executable that is installed on the clients site. This approach is easier to manage because most of the work is done off-line but, in general, it makes individual component updates more technically complicated, if not impossible. This

Uncorrected Proof

is a problem because learning-based components are updated often and usually at a different rate than other more traditional components.

The alternative is therefore to deploy the learning-based components independently and integrate them with the encompassing architecture afterwards. However, this requires a middleware capable of managing components with autonomous life cycles, and, as previously said, written with different programming languages.

## 5.3 Configuration

The learning models are built off-line with data usually coming from purpose-built facilities, or from selected customers' sites. But, in any case, data remains relatively specific to these training environments, as do the models obtained. Thus, when models are deployed to clients' sites, it is necessary to adapt them to their new conditions of execution. This problem is particularly acute in pervasive computing where there is no two environments alike.

By nature, model configuration is done on site, at the edge level, using data collected from the environment. Two approaches are possible. The first is to use a large amount of data that has been previously collected and stored for this purpose. With this data, the general model received can be efficiently fine-tuned. This approach is very effective but requires significant storage capacity on the edge. The second approach, which does not rely on such capacity, is to use data collected after the general model is received. Here, fine tuning is not usable because the well-known catastrophic forgetting problem occurs very quickly. It is necessary to use Continual Learning techniques, like regularization for instance, to adapt the learning model. Continual Learning, also called lifelong learning or online machine learning, is the ability of a model to learn continually from an infinite stream of data, gradually integrating new acquired knowledge into old knowledge (Chen and Liu 2018). This however results in complex algorithms.

## 5.4 Data Collection

The use of prediction models adds constraints when running pervasive applications. Indeed, it is not only necessary to collect data to make punctual predictions when the model is invoked, but also to allow the continuous improvement of the models. An update can only be meaningful if it is based on a sufficient amount of data, and it is therefore needed to collect data very regularly in order to reach a critical mass. Simply put, data must be continuously retrieved in order to keep models relevant and in sync with their execution environment.

This is a significant difference from current practices where data is fetched only to meet occasional application needs. Often, as explained in Sect. 4, context modules, general or application-specific, are built by pervasive platforms. But

Uncorrected Proof

these contexts usually contain little data and keep only significant data. At best, caching techniques are implemented to save recent data and then provide more responsiveness (Lalanda et al. 2018).

Moreover, when supervised learning is used by the learning algorithms, collected data must be labeled, which means that ground truth is associated with it. Data labeling is well known to be a complex activity. It can be done manually by users or experts via dedicated tools. But this approach is seen as boring and massively rejected by the involved people. It is also very error-prone. In some cases, labels can be calculated automatically, often with a delay, using other data. This approach is safer but requires to collect even more data and to create specific code to effectively compute the labels and associate them with the corresponding data.

## 5.5 Model Execution

Resorting to predictions for the execution of an application is a powerful but risky approach. In some cases, it is always beneficial because it provides information that would not otherwise be available. Even if the prediction is not perfect, its mere existence is better than the absence of information. Especially if, on average, it is accurate. But, in other cases, an inaccurate prediction can lead to important malfunctions, or even endanger the existence of people or infrastructures.

Guaranteeing the relevance of a prediction seems difficult without human intervention, because it depends on the correlation between training data and runtime data, but also on runtime environments that cannot be completely characterized (some information cannot be captured by sensors). A commonly used approach is to multiply the models on a runtime platform. Several models, based on different algorithms and sometimes on different features, are deployed and used together. Then, as in redundancy approaches, voting techniques can determine the "good" predictions. Using several models is likely to be more robust since final results are based on the output of several models. As in redundancy approaches, voting techniques can be used to select a prediction. However, here again, support from the pervasive platform is highly effective to realize those functions.

## 5.6 Model Monitoring

Monitoring is a systematic collection of relevant information with the purpose of understanding, evaluating and controlling the system. For the models, it is a matter of presenting the data used, the predictions made and, if appropriate, all other data allowing to understand or verify the results provided. Let us note that the monitoring function can be adaptive by either adapting the focus of what is being monitored—deciding on which components are monitored and when, or adapting the amount of times or numbers of samples that are required.

Uncorrected Proof

It is of major importance to include human, essentially experts, in the monitoring activity. A simple solution is to monitor the applications behavior and switch to a human mode when some tasks (data collection, model predictions, and so on) are uncertain. Here, visualization tools are definitively needed to assist the domain experts in reviewing the current tasks and in providing help. It is also needed to define a global process making explicit the situations and moments where human can be included.

### *5.7 Model Update*

Trained models are data dependent, making it necessary to retrain them when data changes. Data drifts happen very frequently. This is the case for instance in the manufacturing domain due to the natural wear of certain parts. The decision to initiate re-training is however delicate because it is costly, both in money and in carbon footprint. At its simplest, it can be done on a periodic basis. A more complex solution, and also more effective, would be to rely on observations from monitoring. For example, changes in the value or distribution of data can be observed, or an expert can identify suspicious results.

A major decision is to then know where the new training has to be done. Doing it at the data center level is a natural solution since the most significant computing resources are there. However, this poses a major problem: the (labeled) data collected on the edge must be uploaded to the cloud. This raises obvious security problems and also significant financial costs. This approach is sometimes simply rejected by users who do not want their data stored in a cloud.

The second solution is to build the new model directly on the edge. This brings the opposite advantages to the previous approach but also the opposite problems. In particular, it requires considerable computing and storage capacities on the edge to train a model, quite different from the capacities needed to simply run a model. Such capacities are not always available. Another problem is that, if they each build their own model, the clients using the initial learning model will quickly diverge and not be able to share their data. Nevertheless, there are significant developments in edge machines that tend towards this solution. For example, smartphones are more and more equipped with electronic cards adapted to models training.

Uncorrected Proof

## 6 Recent Trends

As demonstrated in the previous section, challenges to develop ML-based pervasive applications on the edge are not only about developing the best models and algorithms but also to provide support for the entire applications lifecycle. The current techniques are unfortunately not adapted to support activities related to data collection, model installation and configuration, model monitoring and model

frequent updates. There is a clear need to provide new Software Engineering support for these different steps.

In this last part, we explore two promising trends to deal with these new challenges: the use of microservices in pervasive platforms and the notion of Federated Learning.

## *6.1 Microservice-Based Platform*

Microservices can be seen as an extension of the service-oriented architectural style as described previously. It structures a software application as a set of loosely coupled components, called microservices, interacting with each others. Microservices implement specific services that are integrated in a global architecture in order to deliver the expected functionality of an application. Each microservice can implement its own life cycle: it is developed, deployed and maintained individually. It can be thus modified independently without impacting the overall architecture. Microservices are actually a combination of service-based and component-based architectures.

Microservices use lightweight APIs to communicate with each other. While REST APIs have often been used to enable interactions between microservices, there is no requirement to do so. A clear benefit of using REST is that updates of a resource can be limited to the microservice containing that resource. However, not all domains are suitable for a REST expression of the possible results of a component and, moreover, this approach requires relatively rare technical skills. We believe that techniques such as gRPC are preferable today, even if they induce a stronger coupling between the microservices.

Communication between microservices can also be implemented with events. Such events can be used to send business data but also control or synchronization data between the different microservices. In this case, it is interesting to use a communication middleware implementing a publish/subscribe pattern allowing high dynamicity in the interactions between microservices.

Today, a common way to implement microservice-based architectures is to use container-based solutions like Dockers or Kubernetes. These solutions automate the management, deployment, and scaling of services across multiple servers by abstracting the underlying infrastructure.

Today, edge platforms based on microservice architectures are emerging. This indeed solves some of the problems mentioned in this chapter, including the need to decouple the life cycles of traditional and machine learning components. Figure 5 presents the new architecture of the iCasa platform (Lalanda et al. 2021). The different technical services (data collection, data storage, context, ML manager) are implemented on dockers. Thus, each of these elements can be:

Uncorrected Proof

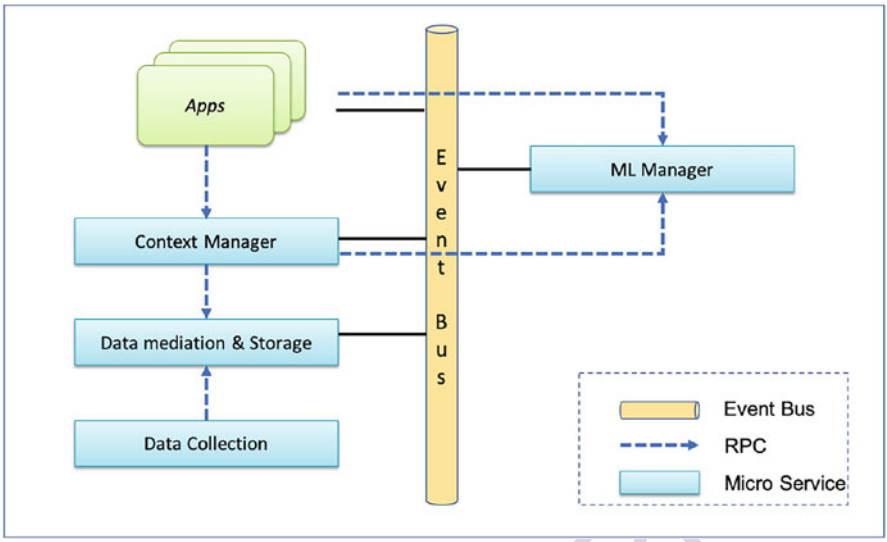


**Fig. 5** Microservice-based platform

- built and deployed independently,
- considered as an independent process that can be implemented in any language, using any framework,
- easily managed with container-based solutions.

## 6.2 *Federated Learning*

Federated Learning promotes the computation of local models on devices and the aggregation of these models on a server to produce a new model that is sent back to clients for use and further training. When the client model has been significantly updated or when a certain time has elapsed, the client model is sent to the server for a new aggregation (see Fig. 6). A core incentive of FL in pervasive computing is to utilize mutual learning goals between clients to indirectly share knowledge with one another. However, effective collaboration between unmoderated clients, is still a challenge. This is due to statistical heterogeneity (differences in individuals' usage) and system heterogeneity (difference between client data caused by different system traits).

A key point in Federated Learning is the way specialized models are aggregated. In the case of deep learning, two families of algorithms can be considered. The first strategy is to emphasize generalization. The aggregation algorithm considers local models in their entirety and builds a new model that potentially calls into question all layers and all weights associated with neurons. This approach is shown

Uncorrected P

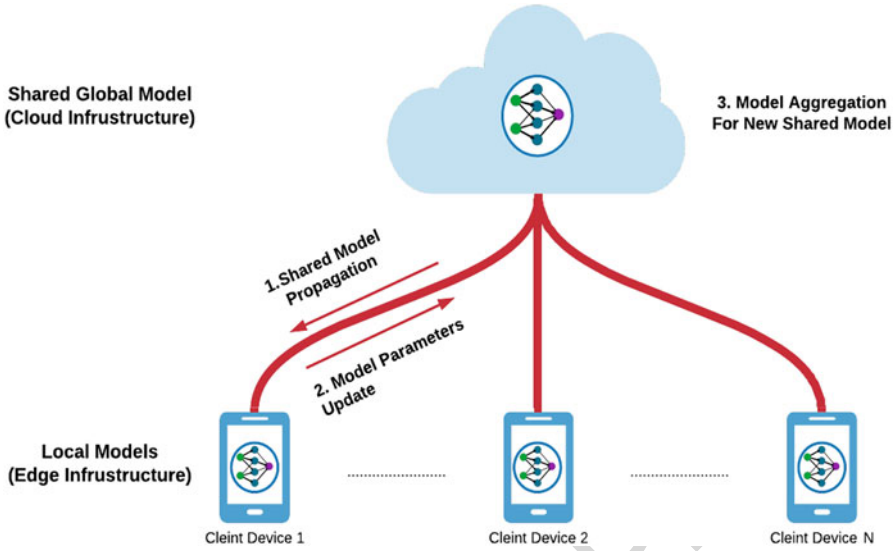


**Fig. 6** Federated learning architecture

in FedAvg (McMahan et al. 2017) and FedMA (Wang et al. 2020) algorithms. The second strategy focuses more on client specialization. Here, the algorithm does not question certain parts of the local models. Precisely, only the base layers are sent to the server for generalization, while the last layers are kept unchanged. This approach is used in FedPer algorithm (Arivazhagan et al. 2019).

Federated Learning has been only recently used to implement pervasive services and, ultimately, does not address all the specifics of this domain well. However, this clearly solves the heterogeneity problems of pervasive environments and also improves the security issues. With this approach, data is not sent to a cloud for retraining. It is only the models that are transferred, which is much better for privacy preservation.

## 7 Conclusion

The integration of machine learning-based models in pervasive information systems is more than a trend, it is a necessity. It allows to face evolving, uncertain, and incompletely characterized environments.

However, this raises unprecedented technological and scientific challenges, notably by calling into question current software engineering practices. It requires bringing together communities that are not familiar with each other and developing new tools, in particular middleware for deployment, execution and integration of models. These models have their own life cycle that must be managed explicitly.

Uncorrected P

In this chapter, we have mentioned two avenues that seem promising, although of course insufficient. The use of microservices within a single execution unit allows the decoupling of the execution of models and traditional software components. Also, distributed learning approaches such as Federated Learning allow to improve models while keeping their specificity at the client level.

## References


Becker, C., Julien, C., Lalanda, P., Zambonelli, F.: Pervasive computing middleware: current trends and emerging challenges. CCF Transactions on Pervasive Computing and Interaction **1**(1) (2019)

Sarkar, S., Chatterjee, S., Misra, S.: Assessment of the suitability of fog computing in the context of internet of things. IEEE Trans. Cloud Comput. **6**(1) (2018)

Choy, S., Wong, B., Simon, G., Rosenberg, C.: The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency. In proc. IEEE NetGames, 2012 (2012)

Shi, W., Dustdar, S.: The promise of edge computing. Comp. **49**(5) (2016)

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., Riviere, E.: Edge-centric computing: Vision and challenges. ACM SIGCOMM Comput. Commun. Rev. **45**(5) (2015)

Vaquero, L.M., Rodero-Merino, L.: Finding your way in the fog: Towards a comprehensive definition of fog computing. ACM SIGCOMM Comput. Commun. **44**(5) (2015)

Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S.: Challenges and Opportunities in Edge Computing. in Proc. IEEE SmartCloud. IEEE, 2016 (2016)

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet Things J. **3**(5) (2016)

Hasan, R., Hossain, M.M., Khan, R.: Aura: An IoT Based Cloud Infrastructure for Localized Mobile Computation Outsourcing. in Proc. MobileCloud, 2015 (2015)

Miluzzo, E., Cáceres, R., Chen, Y.F.: Vision: mClouds - Computing on Clouds of Mobile Devices. in Proc. ACM MCS., 2012 (2012)

Morand, D., Garcia, I., Lalanda, P.: Autonomic enterprise service bus, ETFA2011, (2011), pp. 1–8. https://doi.org/10.1109/ETFA.2011.6059231

Garcia, G., Pedraza, B., Debbabi, P.L., Hamon, C.: Towards a Service Mediation Framework for Dynamic Applications, 2010 IEEE Asia-Pacific Services Computing Conference, (2010), pp. 3–10. https://doi.org/10.1109/APSCC.2010.90

Fernando, N., Loke, S.W., Rahayu, W.: Dynamic Mobile Cloud Computing: Ad hoc and Opportunistic Job Sharing. in Proc. IEEE UCC., 2011 (2012)

Forschungsunion, Acatech: Recommendations for implementing the strategic initiative industrie 4.0. Technical report, Final report of the Industrie 4.0 Working Group (2013)

Lalanda, P., Morand, D., Chollet, S.: Autonomic mediation middleware for smart manufacturing. IEEE Internet Computing **21**(1) (2017)

Lalanda, P., Vega, G., Cervantes, H., Morand, D.: Architecture and pervasive platform for machine learning services in Industry 4.0. in Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) and other Affiliated Events (PerCom Workshops). (2021)

Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In First IEEE Workshop on Mobile Computing Systems and Applications (WMCSA) (1994)

Dey, A.K., Abowd, G.D.: Context-aware computing applications. In Proc. HUC. Springer (1999)

Becker, C., VanSyckel, S., Schiele, G.: Ubiquitous information technologies and applications. Lecture Notes in Electrical Engineering, 214(1). (2013)


Uncorrected Proof

Kosta, S., Perta, V.C., Stefa, J., Hui, P., Mei, A.: Clone2Clone (C2C): Peer-to-Peer Networking of Smartphones on the Cloud. in Proc. HotCloud. USENIX. (2013)
Jonathan, A., Ryden, M., Oh, K., Chandra, A., Weissman, J.: Nebula: Distributed edge cloud for data intensive computing. IEEE Trans. Parallel Distrib. Syst. **28**(11) (2017)
Edinger, J., Schäfer, D., Krupitzer, C., Raychoudhury, V., Becker, C.: Fault-Avoidance Strategies for Context-Aware Schedulers in Pervasive Computing Systems. in Proceedings of the International Conference on Pervasive Computing and Communications (PerCom). (2017)
Ousterhout, K., Wendell, P., Zaharia, M., Stoica, I.: Sparrow: Distributed, Low Latency Scheduling. in Proceedings of ACM SOSP. (2013)
Flinn, J., Park, S., Satyanarayanan, M.: Balancing Performance, Energy, and Quality in Pervasive Computing. in Proceedings of IEEE ICDCS. (2002)
Cicconetti, C., Conti, M., Passarella, A.: Low-Latency Distributed Computation Offloading for Pervasive Environments. in Proceedings of the International Conference on Pervasive Computing and Communications (PerCom). (2019)
Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. in Proceedings of IEEE INFOCOM. (2012)
Breitbach, M., Edinger, J., Kaupmees, S., Trötsch, H., Krupitzer, C., Becker, C.: Voltaire: Precise EnergyAware Code Offloading Decisions with Machine Learning. in Proceedings of the International Conference on Pervasive Computing and Communications (PerCom). (2021)
Lalanda, P., Hamon, C.: A service-oriented edge platform for cyber-physical systems. CCF Transactions on Pervasive Computing and Interaction **2**(3) (2020)
Becker, C., Handte, M., Schiele, G., Rothermel, K.: PCOM - A Component System for Pervasive Computing. in Proceedings of the International Conference on Pervasive Computing and Communications (PerCom). (2004)
Bourcier, J., Diaconescu, A., Lalanda, P., McCann, J.A.: An autonomic management framework for pervasive home applications. TAAS **6**(1) (2011)
Romero, D., Hermosillo, G., Taherkordi, A., Nzekwa, R., Rouvoy, R., Eliassen, F.: The digihome service-oriented platform. Softw., Pract. Exper. **43**(10) (2013)
Dixon, C., Mahajan, R., Agarwal, S., Brush, A.J.B., Lee, B., Saroiu, S., Bahl, P.: An operating system for the home. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), San Jose, CA, USA. (2012)
Papazoglou, M.: Service-oriented computing: Concepts, characteristics and directions. In 4th International Conference on Web Information Systems Engineering (WISE), Rome, Italy. (2003)
Escoffier, C., Hall, R.S., Lalanda, P.: iPOJO: An extensible service oriented component framework. in Proceedings of the IEEE International Conference on Service Computing (SCC) (2007)
Bardin, J., Lalanda, P., Escoffier, C.: Towards an Automatic Integration of Heterogeneous Services and Devices. in Proceedings of the IEEE International Conference on Asian-Pacific Service Computing (APSCC) (2010)
Roth, F.M., Becker, C., Vega, G., Lalanda, P.: Xware—a customizable interoperability framework for pervasive computing systems. Pervasive and mobile computing **47** (2018)
Chen, Z., Liu, B.: Lifelong machine learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 12, number 3 (2018)
Lalanda, P., Mertz, J., Nunes, I.: Autonomic caching management in industrial smart gateways. in Proceedings of IEEE Industrial Cyber-Physical Systems (ICPS) (2018)
McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, vol. 54. Fort Lauderdale, USA, pp. 1273–1282 (2017)
Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D.S., Khazaeni, Y.: Federated learning with matched averaging. CoRR **abs/2002.06440** (2020) arXiv:2002.06440
Arivazhagan, M.G., Aggarwal, V., Singh, A.K., Choudhary, S.: Federated learning with personalization layers. arXiv:1912.00818v1 (2019)

Uncorrected Proof