Stack of Services for Context-Aware Systems: An Internet-Of-Things System Design Approach

Quang-Duy Nguyen, Catherine Roussey, Patrick Bellot, Jean-Pierre Chanet

Stack of Services for Context-Aware Systems: An Internet-Of-Things System Design Approach

Quang-Duy NGUYEN
Research Laboratory LTCI
Télécom-Paris, Institut Polytechnique de Paris
91120, Palaiseau, France
quanguyen@telecom-paris.fr

Catherine ROUSSEY
Research Unit TSCF
INRAE, Université Clermont Auvergne
63170, Aubière, France
catherine.roussey@inrae.fr

Patrick BELLOT
Research Laboratory LTCI
Télécom-Paris, Institut Polytechnique de Paris
91120, Palaiseau, France
patrick.bellot@telecom-paris.fr

Jean-Pierre CHANET
Research Unit TSCF
INRAE, Université Clermont Auvergne
63170, Aubière, France
jean-pierre.chanet@inrae.fr

***Abstract*—The Internet of Things is an ideal world in which all computing devices from all over the world connect and exchange data through the Internet. This new scenario demands context-aware systems to evolve with new characteristics; thus, brings new challenges for system developers in system development. While addressing these challenges, this paper presents a system design approach based on a stack of 16 services specialized for context-aware systems. The approach enables system developers to focus more on services than hardware and software components. The case study of a smart irrigation context-aware system, also presented in this paper, is an example of using this design approach in practice.**

***Index Terms*—IoT, Stack of Services, Context-Aware System, System Design, Smart Irrigation**

I. Introduction

The earlier 21st century has witnessed a significant number of Internet-of-Things contributions in both the public and private sectors. The Internet of Things (IoT) is defined as a scenario in which *"people and things are connected anytime, anyplace, with anything and anyone, ideally using any path/network and any services"* [1]. This term was first coined in 1999 by Kevin Ashton [2]; however, it has been widely adopted due to the exponential growth of computing devices and networking technologies in the past decades. The ideal world of the IoT has become more and more realistic, with a foreseen picture of billions of internet-connected computing devices to connect through many reliable and high-speed internet connection [3], [4].

As a consequence, the IoT demands information systems evolving with new characteristics to adapt to the new scenario. In the scope of our research, this paper focuses on the context-aware system (CAS), a kind of information system that reacts adequately based on context. A typical CAS is composed of three modules: (1) a module to monitor the environment such as a wireless sensor network (WSN); (2) a module to process, analyze, and stock data; and (3) an actionable module. While building a CAS in the IoT, system developers must consider three new characteristics. First, a CAS in the IoT can connect to the Internet and exchange data with external resources. Second, a CAS in the IoT can work in a highly distributed environment. That means the system is divided into several parts located in different local networks. These parts are reassembled logically through the Internet. Third, a CAS can be upgraded frequently with new hardware and software components. This characteristic relies on the fact that the IoT market by hardware and software grows quite fast.

While considering the above characteristics, this paper proposes a new design approach based on a stack of 16 services specialized for CASs. Note that the proposal is developed from the thesis of Quang-Duy NGUYEN [5]. It allows system developers to focus on services instead of hardware and software components. The idea is to view a CAS as a set of services of which each service is *"a logical representation of a set of activities that have specified outcomes, is self-contained and is a black box to consumers of the services"* [6]. Note that a consumer can be a device, a person, or another service. One service can access the Internet and connect with other services. This design approach has three advantages according to the three new characteristics of CASs. First, it proposes services that allow a CAS to exchange data with external resources. Second, each service is self-contained and independent: it can work with the other services from a distance through the Internet. Third, the design relies on services, and hardware and software components are replaceable.

The rest of this paper is organized as follows. Section II presents the concept of the CAS. Next, Section III focuses on the main contribution of this paper: the stack of services for context-aware systems (SS-CAS) and the principles of the design approach. The case study in Section IV is an example of how to use the SS-CAS. Section V compares this design approach with other design approaches. Finally, Chapter VI is a brief conclusion, to summarize the contents of this paper and to open a discussion.

II. Context-Aware System

A CAS *"uses context to provide relevant information and services to users, where the relevancy depends on the users' task"* [7]. Of which, context is the core of a CAS. It refers to as *"a set of entities characterized by their state, plus all information that can help to derive any state changes of these entities"* [8]. In this definition, a state is a quality property that summarizes some information about one entity [9]. The entities and their states are only meaningful in the application which uses them. For example, "outdoor soil moisture" is an entity of an irrigation system, but has no meaning in smart home system. A CAS contains one low-level context and one high-level context. The low-level context is a set of quantitative data; that is to say, it can be a number or any numeric data type. The high-level context is an enrichment of the low-level context with qualitative data. This type of data synthesizes a situation and enables humans to understand a situation to ease the decision quickly. A computing device views a situation as the state of an entity. The qualitative data are more informative but support only logical processing (true, false, comparison).

A CAS is composed of many hardware and software components. They collaborate to do activities and accomplish tasks of the system. Based on the difference between these activities' context, it is possible to group them into four phases: acquisition, modeling, analysis, and exploitation.

Figure 1 illustrates the life cycle of a CAS, in which the pink box represents the scope of the CAS. Note that the actuator device and the WSN are also parts of the CAS. The actuator device works in the exploitation phase, and the WSN works in the acquisition phase. An external system can be a computing device or any system outside the scope of the CAS. When the CAS receives data from an external system, the external system is an external data source. The exchange between the phases in the CAS could be data, low-level context, or high-level context. The CAS receives measured data from WSN or data from external data sources. The CAS can control actuator devices to act on the environment. Humans can retrieve the information provided by the CAS through their users' computing devices. Also, the CAS can send data to external systems. The description of the four phases is as follows.

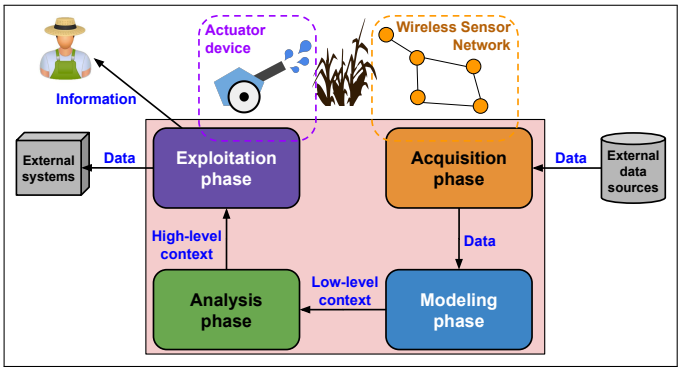

Fig. 1. Life cycle of a context-aware system

- **Acquisition phase** focuses on how CASs retrieve and process measured data from sensor devices or data from external data sources. This paper defines data from the sources located inside a CAS, such as measured data from sensor devices, as internal data. The sources of internal data are also called internal data sources. External data are data from external data sources. Normally, these data are raw: they contain some errors or cannot be used directly. They must be processed to become clean data that the other software components can use. For example, data sent from a sensor device have two parts: the most significant byte (MSB) and the least significant byte (LSB) of an integer number. These two parts of data should be combined to produce the integer number. While the MSB and LSB are raw data, the integer number is clean data.
- **Modeling phase** focuses on how to model and organize clean data into a CAS. A CAS must equip with a data model and data storage in advance. The input of this phase is clean data from the acquisition phase. Note that a CAS should store all of the data; however, not all of them need to be represented in the model. The data able to be represented are part of the low-level context. In CASs, data can be heterogeneous: data from different sources, for different applications, and have different formats. For example, the data format could be text, portable document format (PDF), image, or video. The data model and data storage should be able to deal with the heterogeneity of data. Moreover, this phase should also provide a data extraction activity for some cases, such as extracting numerical data from a PDF document. The output of this phase is the low-level context.
- **Analysis phase** focuses on how to enrich the low-level context to provide the high-level context. The input of this phase is the low-level context from the modeling phase. This phase imitates the human reasoning process: it retrieves the low-level context and reasons them to deduct the high-level context. The high-level context is the material for further decisions. The output of this phase is the high-level context.
- **Exploitation phase** focuses on how the system uses high-level context from the analysis phase to run applications. Three basic types of applications are: (1) providing information to users, (2) sending data to other systems, and (3) making actions. Another feature is the capability that a CAS reconfigures at run-time. This feature enables the system to become adaptive [10].

III. System Design Approach Based on the Stack of Services for Context-Aware Systems

This section describes the new system design approach that supports developing CASs regarding new conditions of the IoT. The first subsection focuses on the SS-CAS, which is the core material of the system design approach. The second subsection presents the principles of the system design approach.

A. Stack of Services for Context-Aware Systems

The SS-CAS is an ordered list of 16 services of CASs. The research team selects these services based on self-experiences and from a study of many available CASs in agriculture [5]. One criterion to choose a service is that it must contribute to the life-cycle of the CAS.

Figure 2 illustrates the SS-CAS. The four rounded rectangle blocks, from bottom to top in the vertical order, correspond to the four phases of the life cycle of CASs, as presented in Figure 1. Each rounded rectangle block contains several services. In a rounded rectangle block, one service Y is on top of a service X means that a new session of the service Y starts when the session of the service X ends and the service Y requires data from the service X. In other words, the service X has the order of execution before the order of execution of the service Y. In each rounded rectangle block, there exist multiple workflows to move between the services inside a block, in one direction from bottom to top. A workflow always starts with the bottom service and moves to the higher service until it reaches the top service and terminates the phase. For example, in the acquisition block, the order of execution from the **source selection** service, crossing the **internal data collection** service, towards the **cleaning** service is a workflow. The description of the 16 services is as follows.

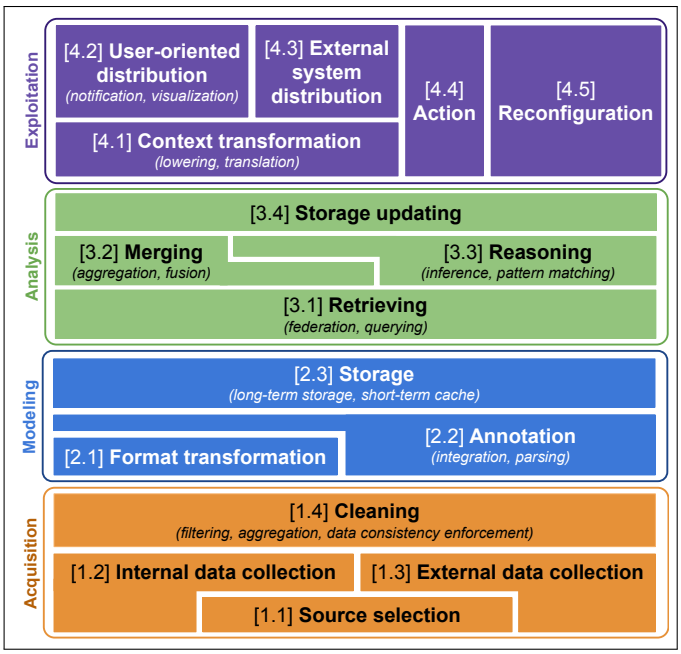

Fig. 2. Stack of services for context-aware systems

[1.1] **Source selection**: The service to register the configuration data of data sources at run-time. By using configuration data, another service can retrieve data from the data sources. The virtual sensor description of the Global Sensor Network middleware (GSN) is an example of configuration data [11].

[1.2] **Internal data collection**: The service to collect data from internal data sources. For example, measured data are collected from a sensor device. This service works differently depending on the communication models: pull model (request/response) and push model (publish/subscribe). In the pull model, the procedure of sending requests and waiting for measured data arrives is repeated. In the push model, the service subscribes to a source only one time, but it receives new measured data as soon as they are generated.

[1.3] **External data collection**: The service to collect data from external data sources. For example, a national weather station that sends forecast data is an external data source. The working procedure of this service also depends on the communication model of pull or push. Different from the internal data collection phase, the external data collection always needs the agreement of the external systems for the data. Moreover, the exchange message between systems must be encapsulated under some networking protocol standards.

[1.4] **Cleaning**: The service to guarantee the correctness of the input data. Three possible actions in the data treatment are ***data consistency enforcement***, ***filtering***, and ***aggregation***. ***Data consistency enforcement*** replaces error data with correct data. It is also named numerical data consistency enforcement [12]. ***Filtering*** selects relevant data and removes irrelevant one. ***Aggregation*** uses basic operators such as addition and subtraction, to produce more accurate data.

[2.1] **Format transformation**: The service that the format of the data is transformed into another format. For example, a text is extracted from a PDF document and transformed into an XML file.

[2.2] **Annotation**: The service to interpret data using a specific schema. The data after this interpretation becomes a part of the low-level context. Two possible actions in the annotation service are ***parsing*** and ***integration***. ***Parsing*** is the particular case of ***integration***: the cleaned data from the acquisition phase are sufficient to be represented in the model. For example, a message from a sensor device contains not only the value of measurement but also the relevant metadata such as the unit of measurement and the date-time of the measurement. They are enough to be interpreted in the schema. ***Integration*** is when the cleaned data needs to be combined with other data stored in the system to be interpreted.

[2.3] **Storage**: The service to store data in different storages. Two cases are: to store the data into long term storage or short term cache. Data stored in long term storage is persistent over time. In contrast, short term data is stored in a temporary cache for the short term use.

[3.1] **Retrieving**: The service that retrieves low-level context data stored in the system. This service is called ***federation*** when the retrieved data are from more than one database. It is called ***querying*** when the retrieved data are from one database.

[3.2] **Merging**: The service that uses aggregation operators or fusion to combine several types of low-level context data to produce new low-level context data. This service is called ***aggregation*** in the case that the input is only numerical data. ***Aggregation*** operates calculations such as sum, average, min, and max. Otherwise, this service is called ***fusion*** in the case that the types of input are heterogeneous. An example is the fusion of an imagery map representing farmland and the

coordinates of sensor nodes located in the farmland. This fusion results in a new map with multiple points, of which each point corresponds to the position of a sensor device.

[3.3] **Reasoning**: The service that uses reasoning techniques to produce new high-level context data. Many reasoning techniques are available in computer science; however, the two techniques that are considered are ***inference*** and ***pattern matching***. This service is called ***inference*** when the system uses a knowledge base to deduct new data. The knowledge base comprises a facts base and a set of rules. The facts base stores low-level and high-level context data of the system. Rules could be interdependent: some rules can only work based on the results derived from firing other rules. An inference made by an inference engine is triggered by the user or by the schedule of an application. This service is called ***pattern matching*** when the system possesses a set of patterns. Patterns are defined by domain experts or by users. The pattern detection engine observes the input and produces the corresponding output when the input is exactly matching with the pattern. Different from a rule, a pattern is independent of other patterns. One pattern itself contains enough data to produce a decision (high-level context data) from the input.

[3.4] **Storage updating**: The service that updates the new data produced after the analysis phase into the storage.

[4.1] **Context transformation**: The service that the high-level context data received from the analysis phase is transformed into another format. The two possible actions are ***lowering*** and ***translation***. The service is called ***lowering*** when high-level context data is transformed into low-level context data. The service is called ***translation*** when data is translated into a human-readable format.

[4.2] **User-oriented distribution**: The service to provide information in a human-readable format to users. An example of the human-readable format is an HTML web page. Two possible actions of the user-oriented distribution service are ***notification*** and ***visualization***. ***Notification*** uses the push model: as soon as receiving new data from the context transformation service, a message is sent to the user. ***Visualization*** uses the pull model: when an user's device requests for information, a response message is sent to the user.

[4.3] **External system distribution**: The service to encapsulate data from the context transformation service using networking protocol standards. Then, the encapsulated data are sent to external systems. The working procedure of this service depends also on the model of pull or push.

[4.4] **Action**: The service to control an actuator device. An actuator can be a simple device such as a open/close water valve or a complex device as a robotic arm.

[4.5] **Reconfiguration**: The service to automatically reconfigure the working schedule of the other services according to the changes of context.

B. System Design Principles

Each service in the SS-CAS has four related elements as follows. Figure 3 illustrates these elements and put them in different layers.

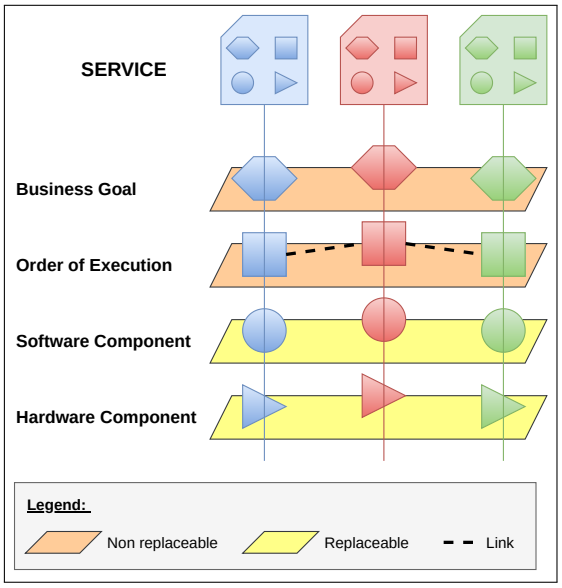

Fig. 3. Four elements of services

- **Business goal** is the goal of a service, that is, to execute some activities to transform an input into an expected output. A business goal of a service always associates with the service. The name of the service briefly describes its business goal. For example, the business goal of the internal data collection service is to collect data from internal sources such as from sensor devices.
- **Order of execution** of a service is the information about which services occur before and after it. This information is necessary to guarantee the workflow of a system. For example, the cleaning service occurs after the internal data collection service. Note that Figure 2 shows not only the services but also the order of execution.
- **Software component** is one or several software programs that run a service to accomplish its business goal. The software component of a service is replaceable. For example, the internal data collection service run by an application built upon Zephyr[1] can be replaced by another application build upon Linux.
- **Hardware component** is one or several computing devices that run a service to accomplish its business goal. The hardware component of a service is replaceable. For example, the internal data collection service run by an Arduino equipped with a Watermark[2] probe can be replaced by a industrial-grade weather station such as Vantage Pro 2 equipped with the same Watermark probe.

Before starting the system design, it is necessary to have a list of system requirements. As usual, this list is the result of the system specification. Design a CAS using the SS-CAS includes three steps.

1) System developers choose appropriate services in the SS-CAS based on the list of requirements. It is necessary

[1]Zephyr is a real-time operating system for embedded systems.
[2]Watermark is a soil moisture sensor widely used in agriculture.

to respect the order of execution of the chosen services.
2) System developers use the chosen services to outline all possibles solutions with their hardware and software component resources. It depends on the conditions of the project that the hardware resources or software resources are considered in the first place.
3) Choose one best solution for the system implementation. It is necessary to save the other solutions as a backup.

IV. Case Study: Smart Irrigation Context-Aware System of TSCF, INRAE

Smart irrigation CAS is a project of TSCF, INRAE. The smart irrigation CAS automates a manual irrigation method IRRINOV®specialized for maize [13]. Following this method, farmers have to do many farming activities manually: (1) they visit the field; (2) they observe the measured data displayed on a monitor box; (3) they calculate aggregate data from measured data; (4) they estimate the state of the crop; (5) they regulate the irrigation schedule; (6) they cancel an irrigation activity when it rains [14]. The above activities are sometimes quite heavy for farmers, for example, in bad weather conditions. The smart irrigation CAS aims to reduce farmers' work; moreover, it helps farmers to use water sparingly. It is composed of three main modules: (1) an IRRINOV®station is an environment monitor system designed in the IRRINOV®method; (2) a smart irrigation decision support system (DSS) suggests irrigation decisions; (3) a web server produces human-readable information. Figure 4 illustrates the IRRINOV®station.

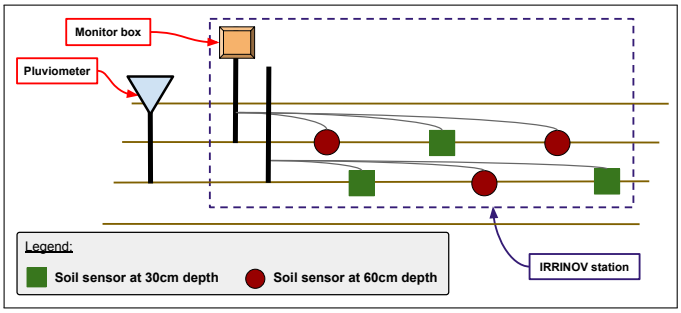

Fig. 4. Prototype of the IRRINOV®station

The methodology used to develop the smart irrigation CAS is mini-waterfall combined with LOT [15]. After the system specification, system developers have a list of requirements. In the system design, they follow the three steps of the SS-CAS.

1) System developers choose 10 services based on the list of requirements. They are: internal data collection, cleaning, annotation, storage, retrieving, merging, reasoning, storage updating, context transformation, and user-oriented distribution.
2) Based on the hardware resources available in the project, which are a work station (server), a Raspberry Pi (gateway), and the IRRINOV®station (sensor device), system developers design two system design solutions: cloud-oriented and fog-oriented, respectively as in Figure 5 and Figure 6. In cloud-oriented system design, most of the services are run by the server. In fog-oriented system design, most of the services are run by the gateway. Then, they select software programs corresponding to the services. For example, the software program Ontogen[3] is used for the reasoning service.
3) System developers choose the first design solution since the gateway in the second solution seems to be overloaded. The second design can be used in the future when the project purchase a high-performance gateway or system developers can reduce resource usage of the software programs.

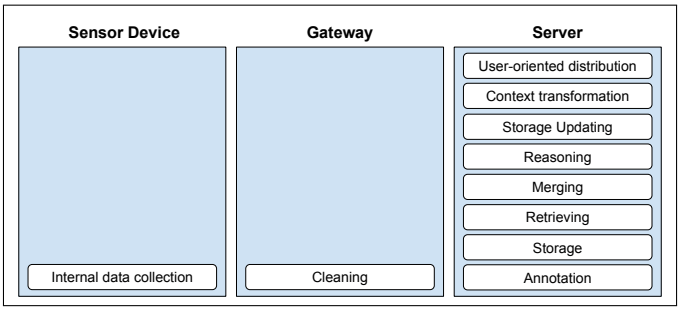

Fig. 5. Cloud-oriented system design solution

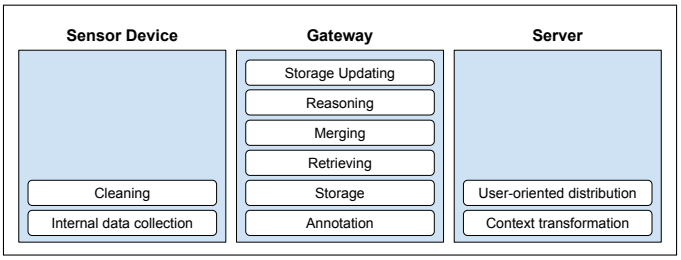

Fig. 6. Fog-oriented system design solution

V. Related Work

The SS-CAS has some aspects in common with other system design approaches in terms of (1) grouping activities into phases, (2) focusing on services, and (3) dividing system into logical blocks.

First, MAPE-K, an architecture presented by IBM, also proposes dividing activities of an information system into four phases: monitoring, analyzing, planning, and executing [16], [17]. This architecture is for adaptive systems, in other words, focusing on reconfiguring a system to maintain its robustness and reliability. However, the SS-CAS provides not only the reconfiguration service but also other services to interact with the environment and users. Therefore, it is possible to say that the SS-CAS includes MAPE-K.

Second, service-oriented architecture (SOA) and microservice are two approaches that also promote designing an information system as services [18], [19]. Both of them present

[3]Ontogen is a software program developed by TSCF. It relies on the SWRLAPI Drools Engine library to do reasoning.

principles to define a service in general but ignore the business goal of each service. Therefore, they have fewer constraints in comparison with the SS-CAS. In detail, the SS-CAS provides a list of predefined services and their order of execution to describe the system's workflow. With these detail, the SS-CAS supports better to inexperienced system developers since they can have an idea of which services are available and how to organize services in their systems.

Third, the architectures Agri-IoT and SWAMP represent a system by small logical blocks. Logical blocks in Agri-IoT are software components. Logical blocks in SWAMP are Generic Enablers (GE) [20], [21]. However, these two architectures are different from the SS-CAS in two points. The first difference is that Agri-IoT and SWAMP distribute their logical blocks into architectures of multiple layers in terms of hardware, network, data treatment, and application: they focus on several aspects of an information system. The SS-CAS groups services into four phases and it focus on each service's business goal. The second difference relates to the dependence of a logical block. The authors of Agri-IoT and SWAMP define each of these logical blocks with a determined type of hardware component and/or a specific software program, as presented in their publications. However, hardware and software components related to services in the SS-CAS are non-predefined.

VI. Conclusion

To sum up, this paper presents the SS-CAS, an architecture composed of 16 services available in CASs. By using the SS-CAS, system developers focus more on the services and have more solutions to organize the hardware and software components of their CAS. The design approach using the SS-CAS is at an abstract level; therefore, it is open and flexible. It requires the system developers of each system to further work on the detail, for example, to design the communication between services of their CAS. The case study of the irrigation CAS in TSCF is a practical example of using this design approach. Note that in the scope of this paper, only a part of this case study is presented.

This paper has three points needed to discuss further. First, the SS-CAS is specialized for CASs. A CAS requires the availability of both low-level and high-level contexts. However, many other information systems work only with low-level context. The SS-CAS is inappropriate to these kinds of information systems. They require another stack of services.

Second, the SS-CAS is formed upon limited sources of researches and experience. Therefore, it is unable to conclude that this SS-CAS can cover all CASs in general. It requires further studies on other domains such as in industry, to contribute and update the SS-CAS.

Finally, it could be interesting to have a method to evaluate the effectiveness of different system design solutions after using the SS-CAS. For example, two different systems from two designs solutions of the smart irrigation CAS could give two different performances in work. This point can be the future work of TSCF.

References

[1] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, May 2013.

[2] K. Ashton, "That 'Internet of Things' Thing," June 2009.

[3] A. Zaslavsky and P. P. Jayaraman, "Discovery in the Internet of Things: The Internet of Things (Ubiquity symposium)," *Ubiquity*, vol. 2015, no. October, pp. 1–10, October 2015.

[4] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle, *Vision and challenges for realising the Internet of Things*. Luxembourg: Publications Office of the European Union, April 2010.

[5] Q.-D. Nguyen, "Interoperability and Upgradability Improvement for Context-Aware Systems in Agriculture 4.0," Ph.D. dissertation, Université Clermont Auvergne, Aubière, France, 2020.

[6] ISO/IEC, "Information technology – Reference Architecture for Service Oriented Architecture – Part 1: Terminology and concepts for SOA," ISO/IEC JTC 1/SC 38 Cloud Computing and Distributed Platforms, Switzerland, Technical report ISO/IEC 18384-1:2016, June 2016.

[7] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, HUC '99*. Karlsruhe, Germany: Springer-Verlag London, UK, September 1999, pp. 304–307.

[8] J. Sun, G. De Sousa, C. Roussey, J.-P. Chanet, F. Pinet, and K.-M. Hou, "Intelligent Flood Adaptive Context-aware System: How Wireless Sensors Adapt their Configuration based on Environmental Phenomenon Events," *Sensors & Transducers*, vol. 206, no. 11, pp. 68–81, 2016.

[9] R. Bendadouche, C. Roussey, G. De Sousa, J. Chanet, and K. Hou, "Extension of the Semantic Sensor Network Ontology for Wireless Sensor Networks: The Stimulus-WSNnode-Communication Pattern," in *5th International Workshop on Semantic Sensor Networks in conjunction with the 11th International Semantic Web Conference (ISWC)*, Boston, United States, November 2012, p. 16 p.

[10] C. Efstratiou, "Coordinated Adaptation for Adaptive Context-Aware Applications," Doctor of Philosophy thesis, Lancaster University, 2004.

[11] K. Aberer, M. Hauswirth, and A. Salehi, "A Middleware for Fast and Flexible Sensor Network Deployment," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, ser. VLDB '06, Seoul, Korea, 2006, pp. 1199–1202.

[12] K. Sha and W. Shi, "Consistency-driven Data Quality Management of Networked Sensor Systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 9, pp. 1207–1221, September 2008.

[13] M. Poveda Villalón, Q.-D. Nguyen, C. Roussey, C. de Vaulx, and J.-P. Chanet, "Ontological Requirement Specification for Smart Irrigation Systems: A SOSA/SSN and SAREF Comparison," in *Proceedings of the 9th International Semantic Sensor Networks Workshop, International Semantic Web Conference*, ser. CEUR Workshop Proceedings, vol. 2213, Monterey, United States, October 2018, pp. 1–16.

[14] Arvalis, INRA, and Chambre d'Agriculture, "Guide de l'utilisateur, Carnet de terrain: Piloter l'irrigation avec la méthode IRRINOV," Arvalis, Midy-Pyrénées, France, Technical report 07X04, May 2007.

[15] Q.-D. Nguyen, C. Roussey, M. Poveda-Villalón, C. d. Vaulx, and J.-P. Chanet, "Development Experience of a Context-Aware System for Smart Irrigation Using CASO and IRRIG Ontologies," *Applied Sciences*, vol. 10, no. 5, p. 1803, March 2020.

[16] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, no. 1, pp. 41–50, 2003.

[17] A. Colman, M. Hussein, J. Han, and M. Kapuruge, "Context Aware and Adaptive Systems," in *Context in Computing*, P. Brézillon and A. J. Gonzalez, Eds. New York, NY: Springer New York, 2014, pp. 63–82.

[18] S. Newman, *Building microservices: designing fine-grained systems*. Beijing Sebastopol, CA: O'Reilly Media, 2015.

[19] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. USA: Prentice Hall PTR, 2005.

[20] A. Kamilaris, F. Gao, F. X. Prenafeta-Boldu, and M. I. Ali, "Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, December 2016, pp. 442–447.

[21] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. Salmon Cinotti, R. Filev Maia, and A. Torre Neto, "Smart Water Management Platform: IoT-Based Precision Irrigation for Agriculture," *Sensors*, vol. 19, no. 2, p. 276, January 2019.