Internet of Things 26 (2024) 101160

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.sciencedirect.com/journal/internet-of-things

Taxonomy and software architecture for real-time context-aware collaborative smart environments

Adrian Bazan-Muñoz [a], Guadalupe Ortiz [a,*], Juan C. Augusto [b], Alfonso Garcia-de-Prado [a]

[a] *UCASE Software Engineering Group, University of Cádiz, Puerto Real, Cádiz, Spain*
[b] *Research Group on Development of Intelligent Environments, Middlesex University, London, UK*

A R T I C L E I N F O

Keywords:
Taxonomy
Context awareness
Internet of Things
Smart environment
Complex event processing

A B S T R A C T

The widespread of Internet of Things (IoT) and the price reduction and ubiquity of telecommunications has led to the emergence of smart environments where devices are becoming increasingly smarter and everything is connected and from which society aims to benefit. The data obtained from IoT is rapidly processed in various domains for the achievement of smart cities and societies. However, in many cases, applications are not contextualized by using data from outside the domain but are only contextualized using data from the domain itself, missing the opportunity for further contextualization. The lack of common criteria for the integration of data from different application domains is one of the main reasons that significantly hinders the integration of third-party data into real-time processing and decision-making systems and thus, the context awareness of developed applications. Although the use of several taxonomies and ontologies for context awareness in various application domains have been proposed, in many cases they are highly domain specific and/or difficult to integrate with other systems, which makes it challenging to facilitate data sharing between different systems and their processing to achieve enhanced context awareness. We aim to contribute to the addressing of these limitations through a reusable and extensible multi-domain taxonomy targeted to collaborative IoT and smart environments, which is also automatically integrated into a software architecture with real-time complex event processing technologies. The proposed solution has been illustrated through a case study and performance tests have been carried out in different computing capacity scenarios, showing its feasibility and usefulness.

1. Introduction

In recent years the Internet of Things (IoT) has evolved significantly and, together with the proliferation and affordability of mobile devices and communications, has led to the emergence of multiple applications for Smart Environments [1]. *Smart environments* refer to the multiple application domains where real-time IoT data processing allows for improved decision making through smart applications, be it smart cities, smart health, smart shopping, etc. The vast majority of these applications for smart environments are focused on monitoring data in a particular domain and location, providing smart actions for such a domain or location, but many applications disregard data from other possible contexts. The fact is that there is no consensus nor generally applicable standard on the definition of

* Corresponding author.
E-mail address: guadalupe.ortiz@uca.es (G. Ortiz).

https://doi.org/10.1016/j.iot.2024.101160

Available online 19 March 2024

data specific to a smart environment domain, what makes it difficult to integrate and correlate data from various sources and systems and reuse them in existing processing and decision-making systems. This means that many applications that could certainly benefit from correlating data from other domains do not achieve this goal due to the difficulty of understanding the semantic and syntactic of such third-party data.

Today's societies could benefit from a greater contextualization of their smart services and smart environment applications and therefore from better decision making, if a software architecture that facilitates the integration and correlation of data from various domains were provided. In order to achieve this, it would be necessary to set a criterion for the format to be used to share data from different domains; achieving such a common criterion would be a major step forward towards data democratisation, the Collaborative Internet of Things (C-IoT) [2,3] and the contextualisation and personalisation of services. C-IoT considers IoT at the scope of heterogeneous elements and domains, where sensors, gateways and services potentially interoperate at different levels; sensors provide their sensed data; gateways add intelligence to them and take actions or report information at a higher level; and services use the information provided by the gateway to improve people's quality of life or business processes. This leads to collaborative smart environments, in the sense that (1) they can benefit from the correlation of data from various heterogeneous domains to contextualise and improve decision making in smart environments in a given domain, (2) that data can come from sensors or gateways indistinctly, which further enriches the context awareness, as well as the result of data correlation can be integrated with different services that can improve contextualisation or support actions suggested by decision making. However, to achieve this collaboration, it is essential to first reach some agreement on the definition and format of the data exchanged by the IoT and in the smart environments. Some taxonomies and ontologies were proposed for the IoT and intelligent applications domains over the last years. However, they suffer from being too generic, without means to extend them, or too focused on a specific application domain; which makes it difficult to reuse them across domains and to seamlessly integrate data from various domains into applications.

Furthermore, it is not only a question of common criteria for the format of the data to be shared between domains. There is also a need of software architectures which facilitate acquiring, processing, and correlating such data and making real-time improved decisions from them, as well as providing explanations to decisions made based on the detected situations of interest [4].

After analyzing the advantages and disadvantages of ontologies and taxonomies, we firmly believe that the use of a taxonomy together with Complex Event Processing (CEP) technologies provides an efficient solution to the existing gap. Although the use of taxonomies does not offer the reasoning provided by the ontology rules, we consider that a taxonomy can be used in conjunction with the reasoning provided by the complementary use of CEP rules. This joint use will facilitate the integration of the system with other existing systems in the IoT and smart environment domains, as well as the processing of heterogeneous data coming from these domains. Indeed, there are several proposals that support the use of CEP-based architectures as an effective and efficient alternative for IoT and smart environments [5–7].

For these reasons, in this paper, first, a taxonomy to facilitate the description of IoT and smart environment data in a homogeneous way for different application domains is proposed. This taxonomy can be used both to describe data from the domain in question for which the application is to be developed, as well as to describe data from other domains that can enrich the contextual knowledge of the individual or entity being served. In addition, it also provides the means to describe the actions involved in decision-making in IoT and smart environment domains. It is important to note that the taxonomy provides a set of common attributes for the IoT and smart environment domains but is extensible with other attributes that may be needed in the domain in question. The proposed taxonomy is accompanied by a JSON structure that will facilitate the description of the IoT data according to the taxonomy in order to facilitate data processing.

Secondly, a software architecture for the acquisition, processing and correlation of domain and contextual data, defined according to the previously mentioned taxonomy, using CEP technologies is provided. CEP allows processing and correlating streaming data and real-time detecting situations of interest in IoT and smart environment domains [5–7]. We will show how CEP can be easily integrated with the proposed taxonomy, to process both input data, whether domain or contextual, and output data, which enables the necessary action to be taken on the basis of the situation of interest detected in the domain in question. The advantage of using a common format for the description of events in different domains and that this format is supported by the processing software architecture, is that events in domain A, if shared, can provide contextual information to domain B, and vice versa. Facilitating the sharing of data between different domains enables knowledge of data from such various domains that help to contextualize situations in a given one, and the data from the latter can help contextualize one of the earliest, thus improving decision making in both domains. For a better understanding, we will exemplify the proposal with a case study and evaluate its performance. Of course, sharing data from different domains will require the awareness of the domain stakeholders and system users [8] —such as a city public and private entities and citizens in a smart city domain— about open data and information sharing, as well as ensuring data security and privacy.

To sum up, the main contributions of this paper are: (1) a taxonomy and JSON based structure for the homogeneous definition of domain and contextual data, as well as of possible actions to be taken in the domain in question depending on the relevant situations detected in it. (2) A software architecture that integrates the data received with the previously indicated format and that processes and correlates them by means of CEP, producing an output also based on the previous taxonomy and structure. (3) The representation of the data in JSON, according to the proposed structure and taxonomy, for a case study, as well as the CEP patterns that will allow processing the data and detecting situations of interest in that case study. Such case study will illustrate how data from one domain can help contextualise other domains and how decision making in various domains is improved by such contextualisation. Finally, (4) an evaluation of the resource consumption and performance of the proposed processing and correlation software architecture in different computing capacity machines, showing its feasibility and usefulness in a variety of scenarios.

The rest of the paper is organized as follows. Section 2 presents the state of the art in which this work is contextualized. Then, Section 3 describes the proposed taxonomy. In Section 4 the provided software architecture is explained in detail, while Section 5 presents an illustrating case study. Then, Section 6 explains the tests to be carried out and subsequently evaluates their results. Finally, the results obtained are discussed in Section 7, subsequently, related work is examined in Section 8 in finally conclusions are outlined in Section 9.

2. State of the art

Context and *context awareness* have been widely discussed over time [8–12] since they are an essential part of intelligent environments by enhancing the interaction between the user and the system. One of the most widespread and accepted definitions of "context" was proposed by Abowd and Dey et al. [10,13], in which "Context is any information that can be used to characterise the situation of an entity". The entity can be a person, a place or almost anything, as long as it is relevant to the interaction between the user and the application. Since the characterisation of an entity's situation depends on the domain and the circumstances, in common practice context is usually handled as a specific feature of the system in question and its goals, which is an obstacle to generalise it, and rarely generalization is shown or discussed in practice.

Over the years, a more precise definition of these terms has been sought, also in the perspective of the current state of the art. Recently, Augusto et al. [8,14] have defined "context" as all information that is relevant to qualify a situation of interest for the users of a system. Derived from this definition is the definition of "context awareness", which is the ability of a system to use context information and adapt its services accordingly, in order to make them more useful for the actors of the system, making use of their preferences or needs. As in the previous definitions, in this one, context awareness is system-specific, implying that what is considered part of the context in one system may not be so in another, with a special focus on the stakeholder.

For instance, one of the areas of greatest application of context awareness nowadays is Ambient Assisted Living [15], in which, through the use of sensors in the person's home, in the person himself and from the environment, the person's daily life can be monitored and contextualised, facilitating his/her life and taking actions when necessary. This is an example in which context awareness support the improvement of human health and quality of life, being of great utility to control different disorders and diseases too.

Although several context-aware software systems have been built, they have repeatedly encountered the difficulty of modeling and describing the context so that it can represent reality and at the same time be understandable by the software that will process it. This difficulty has been solved over the years very frequently with domain-specific or more general taxonomies and ontologies, and, in recent years, with special emphasis on the IoT and smart environments. While an ontology is a formal definition shared between different domains, which makes it possible to describe concepts and relationships in an application domain; a taxonomy is less formal and provides a systematic organisation of objects or concepts into hierarchical categories, based on shared characteristics to efficiently classify and retrieve information.

In general, the proposals we find on taxonomy and ontology, either search for the creation of a structure as simple as possible, which allows the most general use of them; or design the taxonomies and ontologies for a particular domain, in which the use of very large, domain-specific structures is required.

Within the specific taxonomies and ontologies we find several proposals such as Mosqueira-Rey et al. and Wang et al. [16,17], where the taxonomies and ontologies defined are for the particular domain of product usability, and home domain, respectively. As a consequence of the specificity, the proposed taxonomies and ontologies are very detailed structures which include the specific attributes of such domains. This has a clear advantage when targeting a single domain, but at the same time it requires to create a new structure for each domain of interest.

As previously mentioned, there are some proposals that aim to provide more general taxonomies and ontologies. For instance both in Kofod-Petersen et al. [18], Anagnostopoulos et al. [19] and Preuveneers et al. [20]; the main aim is providing the taxonomy with versatility and flexibility, without focusing on a particular domain, but rather trying to provide them with mechanisms that allow them to be adapted to multiple application domains. It is important to mention the usefulness of general proposals; a clear example of their use can be found in Chen et al. [21], where a general ontology —CoBrA— was first designed, applicable to several domains. Subsequently, in order to demonstrate how to extend a general ontology into a specific one to be useful in a particular domain, the proposal was extended with SOUPA [22] for supporting specific types for pervasive application domains.

While most of the previously presented proposals refer to ontologies, we have also seen that some of them use a taxonomy to structure context data, without the need of using an ontology, such as those proposed by Mosqueira-Rey et al., Kofod-Petersen et al., Belkadi et al., Zhou et al., and Zimmermann et al. [16,18,23–25]. Both possibilities are equally effective for representing this type of context information. Although ontologies have the advantage of providing a richer description of the relationships between terms and allowing the definition of a set of rules related to the defined terms; they also have certain limitations. These include the few existing options in terms of ontology definition languages, the difficulty in transferring specialised knowledge from texts or domain experts to abstract and efficient conceptual representations and the lack of adequate tools to build them [26,27]. Taxonomies, on the other hand, will allow us to define the context, but not the associated rules. While this may seem a limitation, it has the advantage of allowing us to be more flexible in terms of the language used for the definition of rules associated with the context, with the possibility of using more extensive grammars, as well as adapting to current trends in terms of data description and processing.

On the other hand, there are some emerging standards that try to facilitate event management and event logging in today's systems. Noteworthy is the IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams, approved by IEEE in 2016 [28]. XES provides a hierarchical structure for representing events, is extensible and adaptable to different domains, allows additional attributes to be added to enrich the event information. We found two limitations in XES in terms of the needs of the IoT and smart environment domains; firstly, that most of the sensor and application data producers are in JSON, yet XES proposes the use of XML; and secondly, that we consider it difficult to use the standard in these domains as it defines the data at a high level of abstraction and does not provide the means to easily match common data such as environment, activity or stakeholder with the attributes of the standard. Also worthy of mention is the work being done along the same lines with the proposed Object-Centric Event Data (OCED) [29] and Object-Centric Event Logs (OCELs) [30] standards. Although these consider the JSON format, the standard remains at a very high level of abstraction, which makes its acquisition difficult in smart environment scenarios where there are a series of concepts that appear recurrently in order to contextualise the systems.

Besides, having in mind the rapid evolution of software and hardware, we firmly believe that a general taxonomy would be more beneficial than a domain-specific one: having a standard for data sharing and a common general structure together with the mechanisms to adapt it to the different application domains can be very advantageous for smart environments in general and smart cities and environments in particular. Such a taxonomy should however include the most commonly used terms for context definition, especially in the IoT and smart environment domain.

2.1. Methodology followed to examine ontologies and taxonomies for IoT and smart environments

To know the most commonly used terms and categories, our goal is to examine the most significant existent proposals of taxonomies and/or ontologies for IoT, smart environment and context aware systems.

For this purpose, we have followed the guidelines for systematic literature reviews proposed by Kitchenham [31] for the field of Software Engineering. The aim of applying this procedure is to generate a list of relevant works in our field, filter them to analyse the most significant ones in our domain, extract the relevant data and, finally, analyse them. To do so, the following steps have been followed: determination of the search strategy, definition of the inclusion and exclusion criteria, setting of the quality assessment, data extraction and data analysis.

Concerning the search strategy, many proposals for taxonomies and ontologies for context awareness and smart environments have been made over the years. In this sense, our first decision has been to use Scopus as the scientific database search engine, which guarantees us a certain quality of the articles we find. Although there may of course be other quality articles outside Scopus, it is necessary to limit the search. On the other hand, due to the scope of our proposal, it is evident that terms such as ontology, taxonomy and context aware are mandatory in our search strings. Besides, since context awareness have been widely used along years in the ambient assisted living domain, we have also included this term in the search strings, as we think that the results can provide added value.

In relation to the inclusion criteria, in addition to the previously mentioned terms, we have taken into account the year of publication, considering how the exaggerated growth of IoT in recent years has led to different smart environments than in previous decades. For this reason, we have decided to limit the search to the last 10 years: from 2014 to 2024. In addition, as this is such a changing topic in recent years, we have decided to select from the years 2014 to 2018 only highly cited papers (50 or more citations in Scopus); and from the most recent years (2019 to 2024), all papers regardless of the number of citations. However, 2 papers highly cited by the community will also be included in the analysis, although they date from earlier years, due to the relevance and influence they may have had over the years (see Section 2.2).

Regarding exclusion criteria, papers not indexed by Scopus in the categories *review* or *article*, not written or English or not belonging either to the field of computer science or to the field of engineering were excluded.

Considering the above restrictions and the scope of our proposal, we have used the following search strings contained in the title, abstract or keywords:

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND ("ambient assisted living")) AND PUBYEAR > 2013 AND PUBYEAR < 2019 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND ("ambient assisted living")) AND PUBYEAR > 2018 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND (ontology OR ontologies)) AND PUBYEAR > 2013 AND PUBYEAR < 2019 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND (ontology OR ontologies)) AND PUBYEAR > 2018 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND (taxonomy OR taxonomies)) AND PUBYEAR > 2013 AND PUBYEAR < 2019 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

TITLE-ABS-KEY ((context-aware OR "context aware" OR context-awarenes OR "context awareness") AND (taxonomy OR taxonomies)) AND PUBYEAR > 2018 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA, "COMP") OR LIMIT-TO (SUBJAREA, "ENGI")) AND (LIMIT-TO (DOCTYPE, "re") OR LIMIT-TO (DOCTYPE, "ar")) AND (LIMIT-TO (LANGUAGE, "English"))

For quality assessment of the proposals retrieved from the search, we propose a two-step strategy. Firstly, proposals that do not propose a new taxonomy, ontology or structure for the classification of data in IoT and/or intelligent environments will be discarded. Secondly, since for contextualisation in any IoT and ambient intelligence scenario we consider it necessary to at least take into account the stakeholder's environment or activity, proposals that do not present the concepts *activity* or *environment* in their taxonomy,

Table 1
Comparison of common categories/features in already existent ontologies/taxonomies to the ones proposed in this approach.

Approach	Approach categories	Approach features	Approach attributes	Proposed categories	Proposed features	Proposed attributes
Forkan et al. [32]	Person	–	Age	Stakeholders	Features of Interest	–
		–	Sex			
		–	Name			
		–	Weight			
		–	Height			
	Place	–	Location	–	Location	–
		–	Longitude			
		–	Latitude			
	Environment	–	Temperature	Environment	Environmental conditions	–
		–	Humidity			
		–	Light			
		–	Noise			
		–	DateTime			
	Device	–	–	–	Device	–
Forkan et al. [36]	Activity	–	Start-time	Activity	Time	–
		–	End-time			
		–	High level activity		–	–
Gu et al. [37]	Computational Entity	Service	–	Technological Infrastructure	–	–
		Application	–		–	–
		Device	–		Device	–
		Network	–		–	–
		Agent	–		–	–
	Location	IndoorSpace	–	–	Location	IndoorPlace
		OutdoorSpace	–			OutdoorPlace
	Person	–	–	Stakeholders	–	–
	Activity	DeducedActivity	.	Activity	–	InferredActivity
		ScheduledActivity	.		–	ScheduledActivity
Ouissem et al. [38]	Technology	Learning Technology	Device	Technological Infrastructure	Device	–
			Network		–	–
	Activity	Learning Activity	–	Activity	–	–
	User	Learner	–	Stakeholders	–	–
	Environment	learning Environment	Location	Environment	Location	–
			Time		Time	–
Aguilar et al. [39]	User	–	UserProfile	Stakeholders	–	–
		–	Role		–	–
		–	Schedule		–	–
	Activity	Domain	–	Activity	–	–
	Time	–	Instant	–	Time	Instant
		–	Interval			Interval
	Device	–	–	Technological Infrastructure	–	–
	Services	–	ServiceProfile	–	–	–
		–	QoS		–	–
	Location	–	Indoor		Location	IndoorPlace
		–	Outdoor			OutdoorPlace
		Environment	Temperature	Environment	Environmental Conditions	–
			Illumination			–
	External Context	–	Software	–	–	–
		–	Hardware		–	–
Elkady et al. [40]	Activity	–	Status	Activity	–	–
		–	End_time		–	–
		–	Start_time		–	–
		Compound Activity	–		–	–
		Simple Activity	–		–	–
	Context	Temperature	–	Environment	–	–
		Location	–		Location	–
		Date	–		Time	–
		Time	–			–
	Entity	–	name	Stakeholders	Features of Interest	–
		Person	–		–	–
		Object	–		–	–
		Application	–		–	–
	Sensor	–	ID	–	Sensor	SerialNumber
		–	Type			–
		–	value			Value

(*continued on next page*)

Table 1 (*continued*)

Approach	Approach categories	Approach features	Approach attributes	Proposed categories	Proposed features	Proposed attributes
Pradeep et al. [33]	Entity	LivingEntity	–	Stakeholders	–	–
		ComputationalEntity	–		–	–
		VirtualEntity	–		–	–
		NonLivingEntity	–		–	–
	Activity	Internal	–	Activity	–	–
		External	–		–	–
	Event	–	–	–	–	–
	Goal	–	–	–	–	–
	State	–	–	–	–	–
	Relationships	–	–	–	–	–
	Location	–	–	–	Location	–
	Time	–	–	–	Time	–
Chen et al. [21]	Place	–	–	–	Location	–
	Agent	–	–	Stakeholders	–	–
	Location	–	–	–	Location	–
	Activity	–	–	Activity	–	–
Ngankam et al. [41]	Person	Profile	–	Stakeholders	–	–
		Preference	–		–	–
		Characteristic	–		–	–
	Activity	Personal	Hygiene	Activity	–	–
		Community	Service		–	–
		Residential	Manage		–	–
	Assistance	Audio	–	–	–	–
		Light	–		–	–
		Comfort	–		–	–
		Safety	–		–	–
	Device	Controller	–	–	Device	–
		Actuator	–			–
		Sensor	–			–
	Task	Subtask	–	–	–	–
		Action			–	–
		Operator	–		–	–
		Condition	Pre-Condition		–	–
			Post-Condition		–	–
	Home	PhysicalObject	–	–	–	–
		PhysicalPlace	–		–	–
Hasanov et al. [42]	User	User profile	–	Stakeholders	–	–
		Social customs	–		–	–
		Emotions and interest	–		–	–
		Cognitive abilities	–		–	–
		Learning style	–		–	–
	Environmental	Temperature	–	Environment	Environmental Conditions	–
		Humidity	–			–
		Luminosity	–			–
		Noise level	–			–
	Technical	User interface	–	–	–	–
		Operating system	–		–	–
		Network bandwidth	–		–	–
		CPU-occupancy	–		–	–
		Battery	–		–	–
		Memory and screen size	–		–	–
	Spatio temporal	Time	–	Environment	Time	–
		Location	–		Location	–
		Task duration	–		Time	Interval
	Pedagogical	Learning objectives	–	–	–	–
		Pedagogical strategy	–		–	–
		Learning activity	–		–	–
		Tools and learning resources	–		–	–
Yilmaz et al. [34]	Location	–	–	–	Location	–
	Weather	–	–	–	Environmental Conditions	–
	Person	Personal Profile	–	Stakeholders	Features of Interest	–
	Computational Entity	Agen	–	–	–	–
		Smart Phone	–		–	–
		Computational Profile	–		–	–
	Social	–	–	–	–	–
	Time	–	–	–	Time	–

(continued on next page)

Table 1 (*continued*)

Approach	Approach categories	Approach features	Approach attributes	Proposed categories	Proposed features	Proposed attributes
	Device	–	–	–	Device	–
	Activity	–	–	Activity	–	–
	Movement	–	–	–	–	–
	Zone	–	–	–	–	–
Degha et al. [35]	Appliance	Device_Consume Energy	–	–	–	–
		Device_Produce Energy	–		–	–
	Time	Time_Temporal	–	–	Time	–
	Source	Energy-Source	–	–	–	–
		Gas-Source	–		–	–
	Bill	–	–	–	–	–
	Policy	–	–	–	–	–
	Action	Command-Action	–	–	–	–
		Notification-Action	–		–	–
		Turn-On	–		–	–
		Turn-Off	–		–	–
		Update-Value	–		–	–
	Building	–	–	–	–	–
	Calendar	–	–	–	–	–
	Network	–	–	–	–	–
	Environment	–	–	Environment	–	–
	Place	IndoorPlace	–	–	Location	IndoorPlace
		OutDoorPlace	–			OutdoorPlace
	Vehicle	–	–	–	–	–
	Actor	Group	–	–	–	–
		Individual	–		–	–
	Behaviours	–	–	–	–	–
	Activity	Inferred-Activity	–	Activity	–	InferredActivity
		Scheduled-Activity	–		–	ScheduledActivity
	Outcomes	–	–	–	–	–
	Service	Service_Type	–	–	–	–
		Service_Grounding	–		–	–
		Service_model	–		–	–
	Event	–	–	–	–	–
	Location	–	–	–	Location	–
	Profile	Human_Profile	–	–	–	–
		Service_Profile	–		–	–
		Policy_Profile	–		–	–
		Site_Profile	–		–	–
		Building_Profile	–		–	–
	Data-Object	–	–	–	–	–
	Requirement	–	–	–	–	–
El-Bouroumi et al. [43]	ContextOntology	User	–	Stakeholders	–	–
		Device	–	–	Device	–
		Environment	–	Environment	–	–
	DomainOntology	–	–	–	–	–
	ProcessOntology	ProcessElement	Service	–	–	–
			Event		–	–
			Gateway		–	–

ontology or structure will be dismissed.

In the data extraction stage, we have obtained the main categories, features and/or attributes presented by the taxonomy, ontology or structure from each of the selected papers; and we have arranged them in a table to facilitate their subsequent analysis. Finally, in the analysis stage, we have identified the most common and most relevant categories/features/attributes in the taxonomies, ontologies and structures for IoT and smart environments, in order to be in a position to then formulate our proposal. Both stages are explained in more detail in the following subsection.

2.2. Results of the followed search and analysis strategy

The search process was conducted in January 2024. Following the search strategy, 232 proposals passed the filter of the above-mentioned inclusion and exclusion criteria (available in supplementary material). Once the search was completed, the selection process began. For this purpose, we proceeded to read the different papers, seeking first to discard those proposals that did not provide a new taxonomy, ontology, or structure for the classification of data in the IoT and Smart Environments. After this discarding process, a total of 135 proposals remained. Subsequently, proposals that did not present the concepts of *activity* or *environment* in their taxonomy, ontology or structure were discarded. After this discarding process and without taking into account the repetitions of the proposals in different searches, a total of 29 proposals were left.

All the data obtained at the data extraction stage can be found in the supplementary material. Although the subsequent analysis has been done over all 29 proposals, we have included a representative selection of these in the manuscript, as the results with this selection are sufficient to illustrate our proposal: proposals that are too domain-specific and those that are already redundant with respect to others selected for inclusion in the table, can be found in the supplementary material, but not in this manuscript, as they do not provide additional information relevant to the analysis.

In Table 1 we can see the study carried out on the selected works in which a classification of data is provided by means of an ontology or a taxonomy. In this table we can see a first column with the reference of the publication. A second column, named *Approach Categories* indicates the different categories proposed in the mentioned approach in the case of grouping the features by categories. The third column, named *Approach Features*, includes the feature terms proposed by this approach, whether or not they have been grouped by category. And the fourth column, named *Approach Attributes*, provides the attributes proposed by this approach, grouped or not by categories. The remaining columns —*Proposed Categories, Proposed Features* and *Proposed Attributes*—, will be explained and discussed in Section 3. As can be seen in Table 1, multiple features covering very diverse aspects can be found in the literature, from a person's age and gender, to his/her activity, location, time, a device or weather conditions. All the features represented in the table are frequently found in the literature —although in this table we have only included the most significant proposals—. In addition, as we see in the table, in some proposals, features are grouped by categories; some examples of categories are *person, place, environment, technology* and *activity*. Among them we can highlight the existence of very detailed proposals, such as [32], with 3 categories composed of various attributes, although only focused on the ambient assisted living domain. Other proposals address broader domains such as [21] or [33], without using categories and with too generic features. There are also, for example, other domain-specific proposals, focused on dementia in the elderly and on energy saving, respectively, which provide which provide numerous categories and features [34,35]. These two proposals are so specific and extensive that their full list of attributes have not been included in Table 1; in our opinion and in view of other existing approaches, they do not cover all the necessary, but general enough, features for the IoT and smart environment domains. Based on this study and through a more detailed analysis of the categories and features in this table, in Section 3 we propose a taxonomy with the aim of covering a broad spectrum of IoT and smart environment domains. To this end, the taxonomy provides a set of categories and features generic enough to be easily adopted in different systems, but at the same time trying to provide the main features that recurrently appear in one and other domains.

3. Taxonomy

In this section we present the proposed taxonomy for the definition of domain and contextual data, as well as for the definition of possible actions to be taken depending on the relevant situations detected in a given domain. For this purpose, as mentioned before, we have previously examined most relevant existent proposals of taxonomies and/or ontologies for IoT, smart environment and context aware systems, to see which are the most commonly used categories and feature terms, as shown in the first four columns of Table 1.

The fifth, sixth and seventh columns of Table 1 refer to the taxonomy proposed in this paper. The fifth column, *Proposed Categories*, shows the main categories that we envisage the taxonomy should contain, since, as we will explain in the next section, four relevant

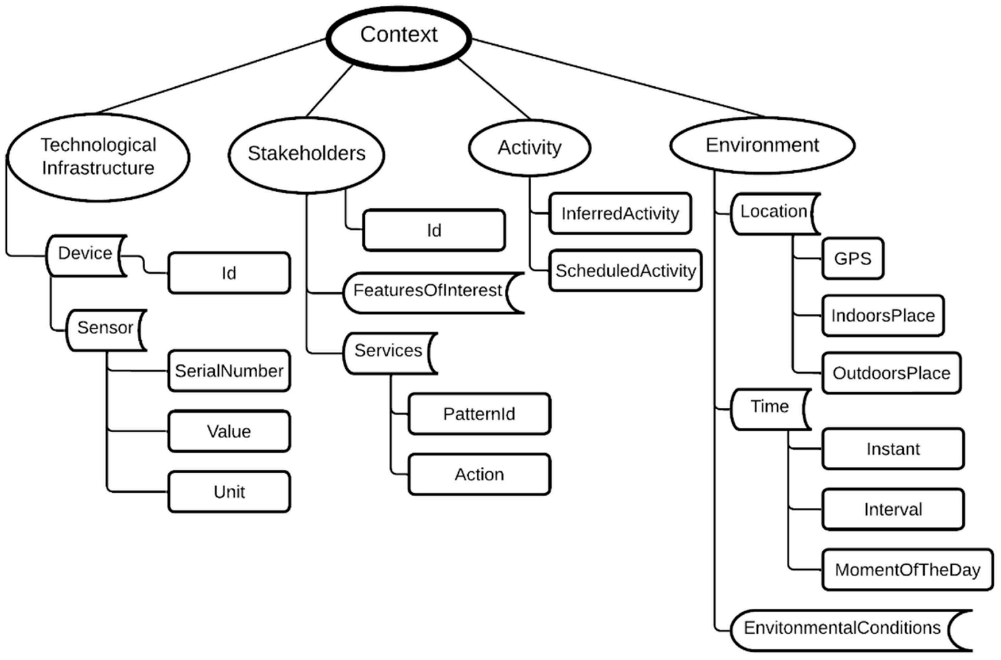

Fig. 1. Taxonomy for context information and basic structure for the domain and context events.

categories have been recurrently found in the IoT and smart environment domains. The sixth column, *Proposed Features*, indicates the proposed features for each of the categories, which will allow us to group and define the various attributes proposed in the last column, *Proposed Attributes,* in a more structured way. Note that we have only included in this column those attributes that we have included in our taxonomy and that already appeared in the literature study. But then we will see in the taxonomy that we have added some more attributes that are not in the table, but that we consider essential for most application domains. This way, we have structured the taxonomy in *categories, features* and *attributes* to better cover and group the attributes, features and categories found in the literature that we consider relevant in IoT and smart environment domains, so that they can be more easily reused in multiple domains and facilitate interoperability between different systems. For this purpose, note that we have, for example, unified terms from different proposals to facilitate the acceptance of our taxonomy by finding a middle ground between all the proposals studied, as well as added new terms where necessary. If the row is blank, it means that either we have found the feature too specific or not relevant enough and therefore it is not represented in our taxonomy.

As a result of the study of other proposals and according with the needs we identified in IoT and smart environment applications, we propose to classify the data into four main categories (ovals) with their own features (second or third level rectangles with concave right side) and attributes (rectangles). Such categories are: *Technological Infrastructure, Stakeholders, Activity*, and *Environment*. Fig. 1 shows the representation of the taxonomy.

In the following sub-sections, we will explain each of the categories, features and attributes of the taxonomy proposed and analyse the rationale for them, based on the study of previous existing work in Table 1.

3.1. Technological infrastructure

The first category within the taxonomy is *Technology Infrastructure*. Here we classify the hardware that performs the different measurements.

If we analyse Table 1 we can see how these data are recurrently classified almost always as *device*. In some cases, *device* is introduced as a feature of *technology*. As this way of classifying these data appears so frequently, we decided to use the *technological infrastructure* category as the synonym of the *technology* category and to include the *device* feature within it. In accordance with other proposals in which *sensors* are mentioned as a separate feature, with their own attributes such as *id, type* or *value*, we decided to provide the possibility of including several *sensors* for every *device* in the taxonomy. Therefore, we decided that all hardware can be considered as a device, which has a unique identifier (*id*). In addition, as previously said, each device contains one or more sensors. Each sensor has a serial number (*serialNumber*) that identifies it, as well as a measurement *value* and the *unit* in which the measurement is performed.

3.2. Stakeholders

The second category of the taxonomy is *Stakeholders*. Here we classify the individuals or organisations that make use of the system.

If we analyse Table 1, we can see how this information is recurrently classified in different ways, whether *person, user* or *entity*. In this sense, we decided to establish a synonymous name for all of them that better represents what we currently understand as context. For this reason, we decided to name this category *stakeholders*. Within this category we use a unique identifier for each user. While some proposals perfectly distinguish the attributes *age, height, weight, name*, or *gender* within the stakeholder, we decided that, since the aim of our taxonomy is to be general enough for different domains, we will not make use of these attributes. Instead, we propose to use a feature called *features of interest,* which describes personal or organisational information relevant to the system in question, and which will be extendable by the end users in order to adapt the taxonomy to the needs of their domain.

Finally, we included a new feature named *services*, which describes the actions to be performed by the system when a certain event pattern is detected. In those examined proposals which provided an ontology, this feature was not necessary since it was dealt by the ontology rules. In our case, the feature *services* is necessary to describe the actions to be performed by the system when a certain event pattern is detected by the CEP engine. These services will contain an identifier of the pattern (*PatternId*) that should be detected to activate the service for the stakeholder. It also includes the *action*, which is the formal description of the action to be performed by the system. In one of the proposals examined, the term *service* was used, but it did not refer to the same concept, but to the description of the web service implemented in a smart classroom, such as a web service for predicting student results.

3.3. Activity

The third category of the taxonomy is *Activity*. Here we classify the activities performed by the stakeholder. If we analyse Table 1, we can see how all the proposals agree on classifying these data as activity, so we decided to call them in the same way. Although some proposals include times within this category, we decided not to introduce these fields within this category, as we consider it more appropriate, like other proposals, to indicate the times in the *time* feature. In our case, the *Activity* category will contain two attributes, as suggested by some of the proposals analysed. *InferredActivity* and *ScheduledActivity,* indicate whether the activity is inferred by some device, or scheduled.

3.4. Environment

The last category of the taxonomy is *Environment*. Here we classify the environment in which the activity takes place. If we analyse Table 1, we can find repeated terms in it. On the one hand, many proposals refer to *time* and *location*, but do not mention aspects of the environment such as *temperature, humidity*, etc. On the other hand, those that do mention these aspects of the environment create a main class, called *context* or *environment*. For this reason, we considered that, to encapsulate all the information in this category, we need a main category named *Environment*, which must have two features named *Time* and *Location*, as well as a third one named *EnvironmentalConditions*. *EnvironmentalConditions* which contains all the information on aspects such as humidity, temperature or noise that affect the development of the activity. Regarding *Location*, it describes the specific place where an activity takes place by means of *GPS* coordinates, by means of the *IndoorPlace*, or by means of the *OutdoorPlace* establishing a location. With regards to the *Time* in which the activity takes place, the *instant* in which it takes place, the *time interval* or the *time of day* might be provided.

4. Software architecture

Once we have a taxonomy that allows us to represent the context, it is necessary to have a software architecture that allows us to receive the data according to the taxonomy in an interoperable format, as well as to process them, and to launch the appropriate actions according to the detected contextualized situations. In particular, in this paper we propose to represent the taxonomy data by using a JSON structure, widely used in the field of IoT and smart environments [44], and to process the data by using CEP, a technique that has proven its usefulness in these scenarios [3,45]. Of course, the taxonomy could be integrated with other data processing and/or analysis technologies, such as machine learning (ML) ones, or even with several technologies at once; this will be discussed further in Section 7.

In this section, firstly, we introduce CEP main concepts to facilitate the understanding of the architecture proposed, then, the three types of events we have considered in our system —domain events, domain context events and external context events— are described. Afterwards, the software architecture proposed and the constraints in the definition of data according to the taxonomy are explained.

4.1. Introduction to complex event processing

CEP [46] is a paradigm that allows capturing, receiving, analysing and correlating large amounts of data in real time from different application domains with the aim of detecting relevant situations.

The data that we capture, and that will later be processed by the system, are called simple events. These simple events can come from information systems or they can be generated by IoT devices. The processing of simple events through the use of CEP allows us to detect different situations of interest, which are called complex events.

To detect these complex events in a given context, we will have had to define a set of event patterns beforehand. These event patterns will analyse simple incoming events, looking for specific conditions that the incoming events must fulfil, as well correlating with each other over time, to detect situations of interest; such situations of interest, when detected, will be called complex events. To put into practice all the concepts explained above, we need a software that allows us to capture the incoming simple events, analyse whether they meet some of the event patterns in real time, and create output complex events. This software is a CEP engine, in which the predefined event patterns will be deployed. Once the events are deployed in the CEP engine, it will receive a stream of simple input events and provide the output of complex events detected based on these patterns.

It is important to note that each CEP engine has its own Event Processing Language (EPL) for the definition of the event patterns. In this work we decided to use the Esper CEP engine [47], due to its high performance [48,49], the versatility of operators provided by Esper EPL pattern definition language and its technological maturity.

To become familiar with this CEP language, which will be used in later sections, we need to know the basic clauses and functioning. In order to define a pattern through an EPL statement, a *schema* or event type must first be declared, which will allow to receive simple events with a specific name and format. To create a schema, we must use the *create schema* clause, followed by the name of the schema. After that, we must indicate the name of the attributes and their type.

On the other hand, Pattern EPL statements usually start with the *insert into* clause, which is an optional clause that allows us to insert the result of the pattern into a data stream, with the name we specify below, making it available for use in other event patterns or as output from the CEP engine. After this clause we find the first mandatory clause of every EPL event pattern: the *select* clause allows us to select which attributes of the event pattern we want to display. After the select clause, we find the *from* clause, which is also mandatory in all EPL patterns, and allows us to indicate from which incoming data flow the information to be analysed in the event pattern will be obtained. Finally, we find the *where* clause, which can be used for joining and correlating event streams. In the following lines we can see an example of schema and event pattern:

(1) *create schema InputEvent (id String, value int);*
(2) *insert into Test*
(3) *select id, value*
(4) *from InputEvent*
(5) *where value>10;*

First of all, we must create the schema in line (1). In this case it is called *InputEvent*, and contains two attributes, a String *id* and an int *value*. After this we start the definition of the event pattern in line (2). In this event pattern, line (3), we select the *id* and *value* from simple incoming *InputEvent* events (line (4)) for those events with the *value* attribute greater than 10 (line (5)), and, if the condition is met, we insert the output data (*id* and *value*) into a data stream called *Test*.

4.2. Event types

Within our software architecture we can distinguish three types of events —domain events, domain context events and external context events—, which are explained below.

Firstly, *domain events*, are those events that pertain to an activity or occur within a specific domain of interest. For example, let's suppose that our domain is the house of an elderly person; a domain event would be any information we receive from the sensors in his/her house, such as whether a door is opened or closed; and those events from activities performed by the person, such as eating, sleeping, and so on.

Secondly, *domain context events* are those events raised by the interaction of a stakeholder with domains other than his/her own. For example, let's suppose that the elderly person in the previous example leaves his/her house, and goes to a bin to throw away the rubbish. This interaction with the bin would trigger a simple event that would be a domain context event.

Finally, *external context events* are global events, i.e., context events that do not pertain to a specific stakeholder. For example, external context events would be events such as outdoors weather conditions, or critical events such as earthquakes, etc.

This classification of event types is done to facilitate data sharing policies. We expect that domain events will not be shared with other domains, in order to preserve the privacy of domain-specific data, domain context events will be shared if they are considered relevant to other domains and external context events will be publicly available. The entity who is providing the data will decide whether an event should be categorized as a *private* context of such a domain event, domain context event or external context event in terms of sharing. This classification does not influence the rest of the system's behaviour, only on how these data will be shared with other domains.

4.3. Architecture

In the proposed software architecture, shown in Fig. 2, we can find the following components:

- Firstly, there are three types of events explained in Section 4.2: domain events, domain context events, and external context events.
- Secondly, there is a CEP engine, in which an event schema and the different event patterns to be analysed are deployed.
- Thirdly, there are two semi-static databases. This means that the information stored in these databases will remain static for a period of time, but at a certain point in time, it may be necessary to update it. One of them stores the persistent context, which is all the semi-static information regarding the stakeholder such as the *FeaturesOfInterest*. The second semi-static database stores the *Actions* that the *Services* perform.
- Fourthly, two REST services are included, which are the interface to inject information into the databases.
- Fifthly, the actions and the persistent context to be stored in the databases are also represented. This information will be sent to the system, as explained in Section 5.2, via a JSON structure.

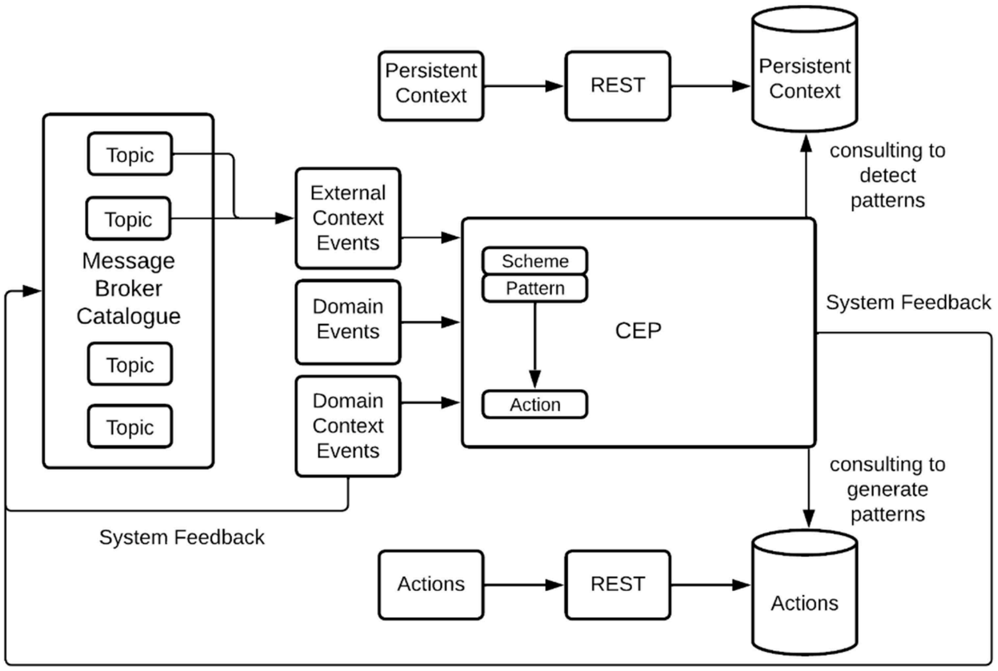

Fig. 2. Architecture proposed for the domain and context data sharing and processing.

- Finally, there is a catalogue, which consists of a messaging broker with different message queues in it, i.e., topics, to which the different users can subscribe.

The proposed architecture has the following functioning:

- Firstly, a series of domain events will be raised in a system domain; they will be sent to the CEP engine for processing and will not be shared with other parties.
- Secondly, a series of context events will be raised. These context events can be of two types. On the one hand, domain context events will be produced by the interaction of the system stakeholder with other domains, i.e., they belong to a stakeholder. These context events will be sent to the system CEP engine for processing and correlation. At the same time, depending on the domain to which the context events belong, (a) they will either be sent to the appropriate topics within the catalogue, and/or (b) the CEP engine will process them and feed them back to the topics within the catalogue. For example, if a user of our domain interacts with a bank to make a card payment, this information would be a domain context event since it is the interaction of the stakeholder with an external domain. This event would be used by our CEP engine, for instance, to detect if it could be a fraudulent payment, and at the same time sent to the bank, to be analysed, studied or stored within the domain of its system. Another example would occur if a user of our domain interacts with a rubbish bin, when approaching to throw the rubbish away, (a) on the one hand, the information of the bin where he/she throw it would be sent to the appropriate topic of the catalogue, to be used by the rubbish management company. (b) On the other hand, this information would be used by our CEP engine, for example, to check if the person has gone to the usual bin. From this fact we can draw the conclusion, for example, that the person may have become disorientated, in which case, we would notify the caregiver, or that the usual container was full, in which case we send this information to another topic in the catalogue to feed back to the waste management company. Finally, we have external context events [50], which will be produced without the interaction of a stakeholder. These events will be sent to the CEP engine for their processing if the stakeholder system is subscribed to any of the topics in the catalogue.
- Thirdly, two tables are available in the database for persistent context. On the one hand, we have a table with semi-static information of domain context events, such as the usual rubbish bin to which the stakeholder goes, in order to detect anomalous behaviour of the stakeholder. On the other hand, we have a table with the stakeholder's features of interest. The instances of this table can be updated if the features of interest of the stakeholder are modified, for example in case a person changes age or a disease is detected. This database will be queried by the CEP engine to be correlated in the pattern where necessary as a context reasoning engine.
- Fourthly, the semi-static actions database will be consulted by the CEP engine to generate actions when a pattern is met. The instances in this table can be updated if the actions required by stakeholders change. For example, if the caregiver of an older person changes, the contact details should be updated.
- Fifthly, the CEP will process the different domain and contextual events received by means of the different event patterns deployed in it. When necessary, it will perform queries to the different databases to collect relevant information required to check the conditions set in the patterns, or to generate the appropriate information for the actions to be taken by the system according to the detected patterns. Finally, useful information generated by some event patterns will be sent to the appropriate topics within the catalogue to feed back into the system.
- Sixthly, the actions and persistent context JSON files, which follow the taxonomy structure, are injected in their respective databases through the invocation of the two available REST services.
- Finally, the topics in the catalogue will store information from different domains to be used as context events in other domains. In turn, the different topics can be fed by the domain context events, and by the complex events detected in the CEP engine. Within the catalogue there is a topic only for the external context. Stakeholders can subscribe to topics that are relevant and useful to them. In addition, the catalogue should have a mechanism for conflict management, security and control of data quality; such a mechanism is out of the scope of this paper and will be dealt with in the future.

4.4. *Constraints on use*

It is important to mention that the CEP engine in the architecture can receive any type of event, but such an event will be discarded by the CEP engine if it does not meet any of the conditions set in the event patterns. Indeed, the CEP engine can receive simple events which only include information of some parts of the taxonomy. For example, we could receive a simple event including a *Device* with its *Sensor Id, value* and *unit*, and a *Stakeholders Id*, with the rest of the attributes having a null value. However, it would not make sense to receive an event with only one value for one of all the possible attributes of the structure, such as just a *Device* with an *Id*, and the rest of the attributes as null.

Therefore, we set that the minimum amount of information that a domain event or a domain context event must contain to be considered by the processing architecture is a level 1 element of the structure, i.e., *TechnologicalInfrastructure, Activity* or *Environment*. Besides, it must contain an identifier or value for one of its attributes, always accompanied by at least one *Stakeholders* element with its identifier, which allows to make the information useful for the stakeholder in question. For example, if an event contains *TechnologicalInfrastructure* and *Stakeholders* with an *Id*, it could be used in an event pattern to check that a device is running. For an *Activity* and an *Environment*, at least one value would be needed, as long as they were accompanied with an identifier in the *Stakeholders* element.

On the other hand, we set that the minimum amount of information that an external context event must contain to be considered by the processing architecture is a level 1 element of the structure, i.e., *TechnologicalInfrastructure, Activity* or *Environment*. In this case it should not be accompanied by a stakeholder identifier, since external context events do not belong to any stakeholder. For example, if an event contains *TechnologicalInfrastructure* or *Environment* with the *value* attribute of the temperature measurement or with the *EnvironmentalConditions*, it could be used in an event pattern to recommend a stakeholder not to go outside under certain circumstances.

All these events are expected to be defined in a JSON structure according to the taxonomy previously described, preserving the hierarchy and being nested as defined in it. The JSON structure can be found as supplementary material data set referred in the Supplementary material section.

Section 5 will explain in detail how the correlation of domain events, domain context events, and external context events received from third parties will allow us to detect relevant actions for the stakeholders. As we will see, we assume that third parties will provide the system with the context data in the correct format, according to the structure proposed, and that they accept such data to be processed by the system to improve the services offered to the system users.

It is also taken for granted that the user in question will collaborate; for instance, the user might have an application of the municipality installed on his/her mobile phone, which might automatically connect via Bluetooth to the beacons located at certain points. It should also allow certain user actions to be identified by third parties providing contextual information: in the example given before this would be identifying in which bin a particular person is throwing out the rubbish.

5. Case study

In previous sections the taxonomy to be used to describe the events as well as the architecture proposed to integrate such events from different domains, enhancing correlation and decision making, were explained. In this section we will present a case study to illustrate the proposal. This case study involves the care of elderly people, both inside and outside their home. When the elderly person leaves the home, this requires the integration and correlation of domain-specific data with domain contextual data and external data, such as, as we will see in Section 5.7, interaction with rubbish containers and knowledge of the fullness and location of these containers, interaction with other people outside the home, or specific weather conditions. There could be other examples of interaction with third party systems, for example in terms of erratic or unusual card or bank account charges, correlation of medication taken (measured at home) with symptomatology detected in the hospital, and so forth. We must therefore bear in mind that this involves providing assistance to meet their physical, emotional and social needs, always guaranteeing their well-being inside and outside their home, quality of life and promoting their independence.

We will first explain how we need to extend the taxonomy for this particular case study and which actions and persistent context need to be injected in the databases. Afterwards, the domain, domain context and external context events to be processed in the case study are defined according to the taxonomy. Finally, several patterns for the case study are shown and explained.

5.1. Extending the taxonomy for the case study

As previously introduced, the taxonomy presented in Section 3 attempts to provide a set of common values in the field of IoT and smart environments, but it is also extensible with specific features for the domain in question. In our case study we will need to extend it with some relevant fields for the elderly care domain.

To do so, we must add the following feature of interest attributes that will be relevant for our case study:

- Age. Age of stakeholder.
- Illness. This field indicates if the stakeholder has any kind of disease.
- Mental illness. This field indicates if the stakeholder has any kind of mental illness.

In the service field we must add the following attributes that will be relevant for our case study:

- PatternId. Name of the pattern which will trigger this action.
- ActuatorId. Identifier of the actuator.
- ActuatorAction. Action to be performed by the actuator.
- NotificationId. Identifier of the person to be notified.
- NotificationAction. Action to be notified, i.e., call on the phone, or send a message, etc.

As previously explained, for the events to be understood by the CEP engine, they must be defined in a JSON structure, according to the taxonomy hierarchy. An example of the format of the base structure can be found in Structure.json file (see Supplementary Material section).

5.2. Inserting actions and persistent context data into the databases

To insert the action data and persistent context one, we must as well make use of the previously defined taxonomy in Section 3.

In order to insert the static data in the actions database, we invoke the corresponding method in the REST services providing the JSON for each instance of each of the tables with the necessary information. We can find an example of a JSON with action information for the pattern detecting when a person is taking too much medication in Actions.json (see Supplementary Material section). It defines the action of making a phone call.

The same should be done to insert the persistent context in the corresponding database. For instance, we can find an example of a JSON with information about the pervasive context of the stakeholder in FeaturesOfInterest.json (see Supplementary Material section), where the stakeholder's age and his/her mental illness are defined. This information will allow us to make use of patterns that analyse the stakeholder's behaviour according to his/her age or mental conditions.

5.3. Defining the domain events

As explained in Section 4.2, domain events are events that pertain to activities or events in a domain of interest. Our domain consists of an elderly person's house in which we install a series of sensors to analyse and detect different situations and behaviours related to the individual's health. As this is a first approach to this problem, as we can see in Fig. 3, we simplified our domain, reducing the person's house to a single room with several sensors that are explained below. Please note that this scenario and sensors are simulated.

Firstly, we have a first contact sensor located on the door of the house, which allows us to know when the door is opened and closed. This opening sensor sends a domain event to the CEP engine every time its state changes, being the possible values emitted 0 when the door opens or 1 when it closes.

Secondly, we have a pressure sensor located on the sofa of the house, which lets us know when someone is seated or not. This pressure sensor sends the CEP engine a domain event every time its state changes, being the possible values emitted 1 when someone sits on the sofa and 0 when the user gets up from the sofa.

Thirdly, we have a motion sensor located in a corner of the room so that it can cover the entire surface of the room. This passive infra-red sensor allows us to know when someone is moving inside the house. It sends the CEP engine a domain event every time its status changes, being the possible values emitted 1 when motion is detected and 0 when no motion is detected.

Lastly, we have a pressure sensor in the rubbish bin of the house, in order to know when the rubbish is removed from it. This sensor sends the CEP engine a domain event every time its status changes, being the possible values the pressure level of the rubbish bin.

On the other hand, the house would also have an electronic water meter, which will send the house water consumption to the CEP engine. In addition, the user would have an electronic pill dispenser, which will also send the information to the CEP engine and therefore will permit to keep track of the medication the user took. For the correct functioning of the system, we take as a premise that the user will not try to cheat the system, for example, by taking medication from the pill dispenser and not consuming it.

All users would have to wear a smart bracelet on their wrist, which will be connected to other devices in the network via Bluetooth and it will allow to identify their interactions with other users and infrastructures such as hospitals. This smart wristband will also measure the user's body temperature among other variables. All these data are also sent to the CEP engine.

The data of all these sensors and the domain events that are simulated in our domain will be processed and correlated in conjunction with the domain context events and the external context events the system receive from external companies, which are also simulated, to detect relevant situations for the person in question. For example, together with the actions performed by the stakeholder inside his/her home, which will provide the system with domain events; the system will process and correlate the data from the actions performed outside the domain, for instance throwing out the rubbish, going to the hospital, etc., which would enter the system as domain context events.

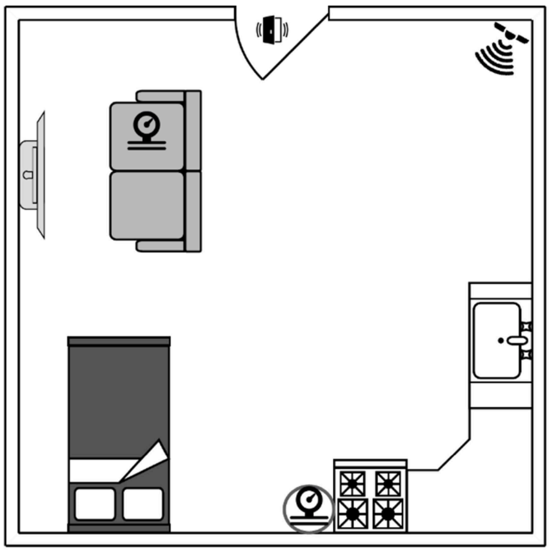

Fig. 3. Distribution of sensors in a house.

It is important to mention that it is understood that all sensors would need to be adapted to submit the information according to the taxonomy format. We assume that every data provider would be submitting the data according to such format; in the future we will investigate additional modules for the architecture to facilitate such task.

An example of a sensor domain event of the sofa pressure sensor can be found in DomainEvent.json (see Supplementary material section).

5.4. Defining the domain context events

As explained in Section 4.2, domain context events are context events produced by a stakeholder's interaction with another domain. This interaction will typically be caused by the user's interaction with a third-party application, or by the interaction of a user's device such as a smart wristband or mobile phone with a third-party detector. An example of a domain context event in our case study would be, for example, a person that makes a payment with a bank card. This payment information would be a domain context event. At the same time, this event may or may not be sent to our system depending on whether it is relevant for analysis or not, and sent to the catalogue, in case it is useful for use in other domains.

Due to the large amount of information that will be managed by the system as a whole, it is necessary that, according to the event patterns we have in our CEP engine, we decide which domain context information is useful for our stakeholder. And on the other hand, if this domain context information is useful for other domains, it would be sent to the catalogue, to be processed by other systems.

As previously said, these events will also follow the structure defined in Section 3. An example of a waste container domain context event can be found in DomainContextEvent.json (see Supplementary material section).

5.5. Defining the external context events

As explained in Section 4.2, external context events are context events that do not pertain to a specific stakeholder. An example of an external context event in our case study would be for example the temperature in the street, pollution in the air etc. These external context events will be available for subscription in the catalogue, to all users. Thus, a person/system could subscribe to the data that is of interest to his/her domain.

These events will also follow the structure defined in Section 3. An example of an external outdoor temperature context event can be found in ExternalContextEvent.json (see Supplementary material section).

5.6. Ensuring data processing security

As the system might handle users' critical personal information such as medical data, it is essential to define the security conditions under which they will be stored, sent and accessed. First, explicit informed consent from the users will be collected, to store, send and access his/her personal data. Furthermore, the data will only be stored for the duration of the contract with the user.

Secondly, appropriate security and confidentiality measures will be implemented to protect the data from unauthorised access, disclosure or misuse. Technical measures such as data encryption, firewalls and intrusion detection systems, and organisational measures such as security policies, employee training and access management.

With regards to data storage, it is worth mentioning that the databases, as well as the messaging broker that hosts the catalogue, may be located on the same device as the CEP engine, or outside it, depending on the company providing the service. The data should be encrypted and have access control limited to the people in charge of their maintenance. For communications between the system, secure communications such as HTTPS will be used. Please note that all these security procedures are out of the scope of this paper [51].

5.7. Defining the complex event patterns

To illustrate the usefulness of the different event types, we have implemented several event patterns in CEP to detect and alert of different situations relevant to user health issues according to our case study, as explained below:

- **SofaSittingDetection**. Although this basic pattern does not generate any alerts for the system, it allows us to explain the basic operation of the following patterns in the case study. It is important to mention that a simple domain, domain context or external context event, may contain information in many of its attributes; but to detect a particular relevant situation, we may only need to obtain and compare some of its variables. In this event pattern, in line (1) the *insert into* clause is used to create an output complex event with the name *SofaSittingDetection*. Then, in line (2) the *select* clause is used to select the attributes that must be kept in the output complex event, in this case a representative name for the incident, the *id* of the *Stakeholders*, and the *value* of the sensor. In later event patterns the *insert into* and *select* clauses will not be explained to simplify the explanations of the event patterns. Specifically, this pattern makes use of a single couch pressure sensor within our domain and is not combined with context events. For this purpose, in line (3) it is checked whether the sensor identifier corresponds to the identifier set for the sofa pressure sensor, which permits knowing that the user is performing some interaction with the sofa. After that, in the same line, it is checked that the value of the sofa pressure sensor is 1, which would indicate that the user has sat on the sofa. For this event pattern the system would not need to make use of more variables of the simple event received.

(1) @public insert into SofaSittingDetection
(2) select 'SofaSittingDetection' as information, Stakeholders.Id as Id,
TechnologicalInfrastructure.Device.Sensor.Value as Value
(3) from DomainEvent (
TechnologicalInfrastructure.Device.Id='Sofa pressure sensor' and
TechnologicalInfrastructure.Device.Sensor.Value=1);

- **LeavingHome**. This event pattern will allow the generation of a complex event, which will be used in subsequent event patterns to identify that the user has left the house. To do this, in line (3) it is first checked whether a domain event, in which the sensor identifier is catalogued as the main door of the house, is received, and then it is verified that the value of this measurement is 0, which would indicate that the door has been opened. Another domain event with the same identifier performed by the same user must then be received, and after that it is checked that the value of the measurement is 1, which would indicate that the door has been closed. Finally, a new domain event must be received, with movement sensor identifier, and a value of 0, which would indicate that the door has been opened and closed and there is no movement inside the house. If there was movement, the door could have been opened and closed, but not exited the house.

  (1) @public insert into LeavingHome
  (2) select a1.Stakeholders.Id as IdStakeholder,
  a2.TechnologicalInfrastructure.Device.Sensor.Value
  as Door, a3.TechnologicalInfrastructure.Device.Sensor.Value
  as Movement
  (3) from pattern [every (
  a1 = DomainEvent(
  a1.TechnologicalInfrastructure.Device.Id='House door' and
  a1.TechnologicalInfrastructure.Device.Sensor.Value=0)
  -> a2 = DomainEvent(a2.Stakeholders.Id=a1.Stakeholders.Id and
  a2.TechnologicalInfrastructure.Device.Id='House door' and
  a2.TechnologicalInfrastructure.Device.Sensor.Value=1)
  -> a3 = DomainEvent(a3.Stakeholders.Id=a1.Stakeholders.Id and
  a3.TechnologicalInfrastructure.Device.Id='Motion sensor' and
  a3.TechnologicalInfrastructure.Device.Sensor.Value=0))];

- **LeavingHouseToThrowOutTheRubbish**. A possible reason for leaving the house may be to go out to throw the rubbish away. The following five event patterns will generate complex events that will be related to each other to detect possible situations such as if a person goes out to throw the rubbish away and suffers from a mental illness and forgets how to get home or gets lost. In this first event pattern, which will be activated when, as shown in line (3), the sensor with the rubbish bin identifier detects a change in its pressure value, being the new value detected 0, indicating that the rubbish has been removed. After detecting this situation, it will be checked if a complex event indicating that the user has left the house is received. This event will indicate that the user has left his house to throw the rubbish away.

  (1) @public insert into LeavingHouseToThrowOutTheRubbish
  (2) select a1.Stakeholders.Id as IdStakeholder,
  a1.TechnologicalInfrastructure.Device.Sensor.Value
  as RubbishEmptying
  (3) from pattern [every (a1 = DomainEvent(
  a1.TechnologicalInfrastructure.Device.Id='Rubbish bin' and
  a1.TechnologicalInfrastructure.Device.Sensor.Value=0 and
  a1.TechnologicalInfrastructure.Device.Sensor.Unit='bar')
  ->a2=LeavingHome(a2.IdStakeholder=a1.Stakeholders.Id))];

- **ThrowRubbish**. Another pattern of this sequence allows for the first time in this use case to make use of context events, using the information received from the rubbish company's system. This is done when it is detected, through the user's smart wristband or the mobile application, that the user is approaching the rubbish bin to throw away the rubbish. To do this in line (3), it is first checked if a complex event indicating that the user has left the house to throw away the rubbish is received. After that, it is checked if a context event is received from the rubbish company, verifying that the user in question is close to the rubbish bin by checking that the name of the outdoor place received is *RubbishBin*. This value for the *OutdoorPlace*attribute would be generated by all rubbish bins in such a way as to unambiguously distinguish that the user is very close to a rubbish bin. This complex event could feed back to the rubbish company as well as to the system.

  (1) @public insert into ThrowRubbish
  (2) select 'THROWRUBBISH' as information,

```
a2.Stakeholders as Stakeholders,
a2.TechnologicalInfrastructure.Device.Id as SerialNumberBin
(3) from pattern [every (a1 = LeavingHouseToThrowOutTheRubbish()
-> a2 = DomainContextEvent(a2.Stakeholders.Id= a1.IdStakeholder and
a2.Environment.Location. OutdoorPlace=’RubbishBin’))];
```

- **ThrowRubbishAndGettingLost**. Once the user has thrown the rubbish away, he/she may not return home after a certain period of time because he/she has got lost. This could be useful in cases of users with dementia. To do this, in line (3) it is checked that after receiving a complex event indicating that the rubbish has been thrown away, and after an interval of 1 h, no domain event has been received from the door sensor of the same user’s house with value 0. This would indicate that the door has not been opened and therefore he/she did not return to his/her home. In the case this pattern is detected, an alert is generated. To generate the alert, the action associated to that user for that complex event in the database would be queried in line (4), and it would be returned as information in line (2).

```
(1) @public insert into ThrowRubbishAndGettingLost
(2) select ’LOST’ as information, a2.Stakeholders.Id as IdStakeholder,
notificationId as notificationId,
notificationAction as notificationAction
(3) from pattern [every (a1 = ThrowRubbish -> (timer:interval(1 hour) and
not a2 = DomainEvent(a2.Stakeholders.Id=a1.Stakeholders.Id and
a2.TechnologicalInfrastructure.Device.Id=’House door’ and
a2.TechnologicalInfrastructure.Device.Sensor.Value=0)))],
(4) sql:ContextInformation [’select id_stakeholder, notificationId,
notificationAction from services
where id_stakeholder=${a1.Stakeholders.Id} and
patternId=\"ThrowRubbishAndGettingLost\"’];
```

- **ContainerCheck**. Another possible situation can occur if the user uses a bin that is not their usual bin to waste disposal. This may be due to a possible disorientation, or because the bin was full. To do this, it is first checked that the bin is the usual bin. To do this, a complex event is received in line (2) indicating that the rubbish has been thrown away. After that, the database that stores the usual bin for each user is consulted in line (4). Finally, in line (5), it is checked if the user usual bin matches the bin in which the user has just thrown the rubbish, comparing its serial number.

In line (6) a new event pattern checks whether the user has not gone to the usual bin due to possible disorientation. To do this after receiving a complex event indicating that the user has not gone to the usual bin in line (8), the database containing the features of interest is consulted in line (9), extracting whether the user suffers from a mental illness and which is his/her age. Finally, in line (10) it is checked whether the person has a mental illness or is over a certain age. With this information, it could be presumed that it may be due disorientation.

```
(1) @public insert into ContainerCheck
(2) select ’ContainerCheck’ as information, a1.Stakeholders as Stakeholders
(3) from pattern[every (a1 = ThrowRubbish())],
(4) sql:ContextInformation [’select id_bin from usualbin
where id_stakeholder=${a1.Stakeholders.Id}’]
(5) where a1.SerialNumberBin!=id_bin;
(6) @public insert into PossibleDisorientationRubbish
(7) select ’PossibleDisorientationRubbish’ as information
(8) from pattern[every (a1 = ContainerCheck())],
(9) sql:ContextInformation [’select mentalillness, age from featuresofinterest
where id_stakeholder=${a1.Stakeholders.Id}’]
(10) where mentalillness=true or age>80;
```

- **Loneliness**. Different contexts can be used for this event pattern, as it has been done before. In this case, the user’s smart wristband is used to monitor loneliness. The smart bracelet detects the proximity of other people who also have the smart bracelet, increasing its numerical value as the person spends a certain amount of time with other people, returning the daily values of relationship minutes to the system. To do this, in line (3) a context event is received, in which the sensor identifier is relationship, and it is checked that a value of less than 30 min is received, i.e., which would imply that the user has not been related enough time to other person. After that, it is checked that another event is received consecutively from the same user with the same values, i.e., that he/she has not interacted enough with other person in two days. In this case an alert is generated in the system.

```
(1) @public insert into Loneliness
(2) select ’Loneliness’ as information,
a1.TechnologicalInfrastructure.Device.Sensor.Value as Value,
a1.Stakeholders.Id as Id,
notificationId as notificationId,
notificationAction as notificationAction
(3) from pattern[every (a1 = DomainContextEvent(
a1.TechnologicalInfrastructure.Device.Id='relationship' and
a1.TechnologicalInfrastructure.Device.Sensor.Value<=30)
-> a2 = DomainContextEvent(a2.Stakeholders.Id=a1.Stakeholders.Id and
a2.TechnologicalInfrastructure.Device.Id='relationship' and
a2.TechnologicalInfrastructure.Device.Sensor.Value<=30))],
(4) sql:ContextInformation ['select id_stakeholder, notificationId,
notificationAction from services
where id_stakeholder=${a1.Stakeholders.Id} and
patternId=\"Loneliness \"'];
```

- **HighWater and LowWater**. Because water consumption can also provide information about certain diseases of the user, the following two patterns detect excessive or too low water consumption. This domain information is provided by the smart water meters in the household. For this purpose, the daily water consumption identifier is found out in lines (3) and (6) and the measurements are checked to determine if they are above or below the usual average water consumption.

```
(1) @public insert into HighWater
(2) select 'HighWater' as information,
TechnologicalInfrastructure.Device.Sensor.Value as Value
(3) from DomainContextEvent(
TechnologicalInfrastructure.Device.Id='Daily Water Consumption' and
TechnologicalInfrastructure.Device.Sensor.Unit='Litres' and
TechnologicalInfrastructure.Device.Sensor.Value>133);
(4) @public insert into LowWater
(5) select 'LowWater' as information,
TechnologicalInfrastructure.Device.Sensor.Value as Value
(6) from DomainContextEvent(
TechnologicalInfrastructure.Device.Id='Daily Water Consumption' and
TechnologicalInfrastructure.Device.Sensor.Unit='Litres' and
TechnologicalInfrastructure.Device.Sensor.Value<20);
```

- **AbnormalTemperature**. This event pattern checks that the user's temperature is within normal values, and if not, generates an alert. The temperature is obtained from the user's smart wristband. For this purpose, it is checked if the measurement corresponds to a body temperature in line (3). The temperature is then checked to see if the temperature is above or below the usual body temperature range. If this event pattern is detected, an alert is generated in the system with the information obtained from the database in line (4).

```
(1) @public insert into AbnormalTemperature
(2) select 'AbnormalTemperature' as information,
a1.TechnologicalInfrastructure.Device.Sensor.Value as Value,
notificationId as notificationId,
notificationAction as notificationAction
(3) from pattern[every (a1 = DomainEvent
(TechnologicalInfrastructure.Device.Id='CorporalTemperature' and
TechnologicalInfrastructure.Device.Sensor.Unit='Celsius' and
(a1.TechnologicalInfrastructure.Device.Sensor.Value>37,8 or
a1.TechnologicalInfrastructure.Device.Sensor.Value<36)))],
(4) sql:ContextInformation ['select id_stakeholder, notificationId,
notificationAction from services
where id_stakeholder=${a1.Stakeholders.Id} and
patternId=\"AbnormalTemperature\"'];
```

- **ExcessiveOutdoorTemperature**. This event pattern checks that the outdoor temperature from an external context event is within normal values. The temperature is obtained from an external thermometer in the city. For this purpose, it is checked if the measurement corresponds to a temperature in line (3). The temperature is then checked to see if the temperature is above or below the established temperature range.

```
(1)public insert into ExcessiveOutdoorTemperature
(2) select 'ExcessiveOutdoorTemperature' as information
(3) from pattern[every (a1 = ExternalContextEvent(
(a1.TechnologicalInfrastructure.Device.Sensor.Value>30 or
a1.TechnologicalInfrastructure.Device.Sensor.Value<5)
and a.1TechnologicalInfrastructure.Device.Sensor.Unit='Celsius'))];
```

- **TakingMedication and TakingTooMuchMedication**. These two patterns check that the user has taken the medication and then check whether the user has taken too much medication. To generate this type of domain event, the user can opt for the manual option, entering the information about the medication and amount taken via a mobile app, or make use of electronic pill dispensers or smart inhalers. In either case, these devices will communicate with the system, indicating that the user has interacted with the device to take the medication. First, on line (3), the pill dispenser identifier is checked, which implies that they have taken their prescribed daily dose. Once this complex event has been generated, in line (6) it is checked that no more daily doses have been taken. If two medication complex events are detected in less than a day, an alert is generated.

```
(1) @public insert into TakingMedication
(2) select 'TakingMedication' as information, Stakeholders.Id as Id
(3) from DomainEvent(TechnologicalInfrastructure.Device.Id='PillDispenser');
(4) @public insert into TakingTooMuchMedication
(5) select 'TakingTooMuchMedication' as information,
a1.Id as id_stakeholder,
notificationId as notificationId,
notificationAction as notificationAction
(6) from pattern[every (a1 = TakingMedication()->(timer:interval(1 day) and
a2 = TakingMedication(a1.Id=a2.Id)))],
(7) sql:ContextInformation ['select id_stakeholder, notificationId,
notificationAction from services where id_stakeholder=${a1.Id} and
patternId=\"TakingTooMuchMedication\"'];
```

- **Sendentarism**. This pattern checks that the user has been sitting for more than 2 consecutive hours, generating an alert for inactivity in due case. For this purpose, in line (3) it is checked that the activity carried out by the user is sitting, which is deduced by the wrist band. If such event is not followed by a walking activity in 2 h, the pattern is matched.

```
(1) @public insert into Sedentarism
(2) select 'Sedentarism' as information, a1.Stakeholders.Id as Id
(3) from pattern[every a1 = DomainEvent(a1.Activity.InferredActivity='sitting')
-> not a2 = DomainEvent(
a2.Stakeholders.Id=a1.Stakeholders.Id and
a2.Activity.InferredActivity='walking') where timer:within(2 hour)];
```

6. Evaluation

In order to evaluate the performance of the system proposed we have developed an implementation of the software architecture proposed and tested it with a set of patterns related to the case study, as explained below.

Fig. 4, represents the data flows in our implementation. The flow is as follows: (1) the external data source sends the input data to RabbitMQ, in our case we use a synthetic data emulator, (2) Esper receives the data by being subscribed to a RabbitMQ queue, (3) Esper sends the detected complex events to a new RabbitMQ, and (4) the external destination machines subscribe to the new RabbitMQ queue.

The same software architecture, shown in Fig. 4, have been tested in different machines, according to 3 different possible scenarios. First of all, we assume that the CEP engine is located in a private home (from now on Configuration 1), for which we use a low-capacity

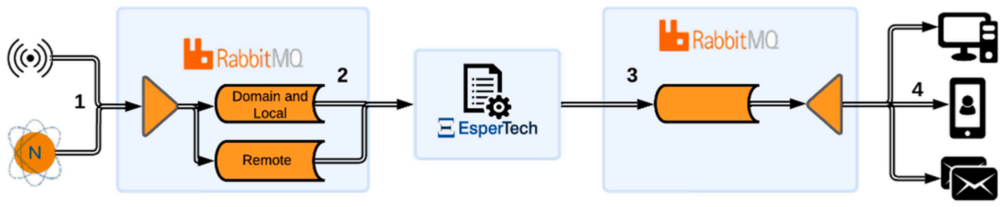

Fig. 4. Software architecture implemented for evaluation.

device, specifically a Raspberry Pi. Secondly, we assume that the CEP engine is located in a housing estate, group of houses or residence (from now on Configuration 2), for this purpose we make use of a device with intermediate capabilities, namely a PC. Thirdly, we assume that the CEP engine is located in a city (from now on Configuration 3), it could be used for a big residential complex or senior resort, or even by a company that manages several living resorts spread over one or more cities. In this case, we make use of a high-capacity device, specifically a server. In the following sections we present the results of the tests carried out on the three different configurations.

Please note that when evaluating the performance of the system it does not matter if the event being processed is from the domain, domain context or external context, since the system always receives them through a messaging queue. This is why a single source of data submitting data to several message queues has been used for the performance tests. All the data for the tests have been emulated. In the RabbitMQ data entry device we have two message queues: one for emulated external context messages, and one for emulated domain and domain context events.

Besides, although we are using a separate computer machine for each component of the system, all components could be on the same machine.

6.1. Computer resources

The following common computer resources were used for the evaluation tests in the 3 configurations:

- A server machine with an Intel Xeon Silver 4110 processor, and 32GB of Ram. This machine was used as broker for CEP engine input events; thus, we have a RabbitMQ instance deployed in it.
- A PC with an Intel i3 3220T and 4GB of RAM. This machine was used as broker for CEP engine output events; thus, we have other RabbitMQ instance deployed in it.
- A PC with an Intel i3 3220T and 4GB of RAM. This machine was used as the external source to send data to the inbound queue; we have emulated all the data with nITROGEN [52] synthetic data emulator.
- All tests were performed within the University network.

Then, the CEP engines were deployed in different machines according to three scenarios described before:

- A Raspberry Pi with a Cortex-A53 processor and 909 MB of RAM was used to host the CEP engine in Configuration 1.
- A PC with an Inter i7 3770 and 7,7GB of RAM was used to host the CEP engine in Configuration 2.
- A server machine with and Intel Xeon Silver 4210R and 32GB of RAM was used to host the CEP engine in Configuration 3.

6.2. Methods

The tests consisted of deploying a series of event patterns in the CEP engine for each of the three configurations described above. For each configuration we deployed all event patterns included in PatternsTests.docx (See Supplementary Material section) at the same time and we progressively increased the number of events sent to the CEP engine per second. Please note that the set of patterns deployed is larger than those explained in the manuscript and it is expected that they are quite resource-consuming. We started with 72 events/s, and continued with 720, 7200 and 14400 events/s, until we reached the maximum number of events per second supported by the CEP engine during the test time. Once the maximum number of events supported by this configuration was reached, no higher event rates were tested.

During the tests, we measured the following performance indicators:

- CPU usage: percentage of CPU usage of the CEP engine process during the test run.
- RAM usage: Megabytes (MB) of memory usage of the CEP engine process during test execution.
- Latency: average time taken by the CEP engine to process each incoming event in milliseconds (ms). To measure this variable, we used a different type of incoming events, so as not to alter the operation of the rest of the event patterns. These events are called *DummyEvents*, and were interspersed among the other input events to measure the processing times during the tests without altering the system. Note that the system is not measuring times of less than 1 ms; so when the event is detected immediately it takes a time of 0 ms and when it takes longer it takes a time of 1 ms (or more millisecond, if it is the case). The proportion of times it takes 1 ms will determine the difference in the mean times obtained in each test.
- Throughput time: total time taken by the engine to process all incoming events in minutes (min). It allows us to identify when the system is overloaded because it could not process all the data instantly, but needed more time.
- Total complex event detected: total number of complex events detected during the test.

All tests were performed for 11 min. If a configuration shows a throughput time longer than 11 min, this would mean that this configuration could not properly deal with such event input rate, since it took too much time to process the data.

A synthetic data emulator called nITROGEN [52] was used to generate the input data. nITROGEN allows us to emulate a set of data, and ensure that a specific percentage of events meet the conditions of the event patterns deployed for the tests. That is, the input data will be homogeneous for all event rates and would be reproducible. Particularly, for the emulated data, 5.5 out of 100 events meet the conditions of the event patterns. For this purpose, we defined 4 groups of events. As explained below, in addition to one group that is included for better tracking of time measurements in the evaluation of tests, there will be two groups generated with a lower frequency of events, which will result in complex events being detected, and there will be another group with a higher frequency, which will not result in complex events. The purpose of this is to achieve high numbers of events in the system but to keep controlled the percentage of simple events that lead to complex events. Thus, group one contains the domain events and context domain events that would trigger the detection of various complex events (for example, a particular combination of three simple *DomainEvent* in which it is first sensed that a door has been opened, then that it has been closed, and finally that no movement is detected in the house, would result in a *LeavingHome* complex event). This group consists of ten different event types and would have an event generation frequency of one event of each type per millisecond at the lowest event generation rate. Group two comprises the external context events that result in several complex events being generated (for example a simple event *ExternalContextEvent* containing the street temperature could, if the conditions are met, trigger the event pattern *ExcessiveOutdoorTemperature*). This group consists of one event type, and it has an event generation frequency of one event per millisecond at the lowest event generation rate. Group three contains the domain events and context domain events that would not result in the detection of complex events. This group consists of five different event types, and it has an event generation rate of ten events of each type per millisecond at the lowest event generation rate, with the objective of overloading the system. Group four contains events that will support timing measurements during system testing. In this case, one event per millisecond is generated at the lowest event generation rate.

In addition, some additional tests have been carried out to analyse how the complexity of the JSON structure and the dependences among event patterns affects the performance of the system. In these tests, the same performance indicators have been measured and the tests have been performed for the same duration, i.e., 11 min, with an input rate that would overload the system.

These tests were performed on Configuration 2, with 3 event patterns. The first event pattern makes use of a schema without attribute nesting and only a comparison is made on one of its attributes. A second event pattern uses a schema with attribute nesting; specifically, in this event pattern, a comparison is made on an attribute nested on a fifth level. On the other hand, another event pattern uses a scheme without nesting attributes and performs a comparison on attributes, but nesting complex events (using complex events that depend on other complex events). Specifically, it performs a comparison on the nesting of five complex events. Tests have only been done on configuration 2, but it is expected that the results would be proportional in the other configurations according to the system resources.

6.3. Results

In this section we present the results obtained in all the performed tests explained in the previous section. Firstly, we explain the results of the tests performed for Configurations 1, 2 and 3, with all the event patterns in the supplementary material deployed and with a duration of 11 min. We started by evaluating the performance from an input event rate of 72 events/s, increasing this rate until we reached 14 400 events/s. Once the limit of events per second is reached for a configuration, higher input rates will no longer be tested for such a configuration, which is why those cells appear in Table 2 with the value (-).

In Table 2, we can find the results of the tests performed for the different configurations. In it, we can see how the three configurations successfully exceed the input event rate of 72 events/s (throughput time of 11 min), but with different performance data. The main difference between the three configurations refers to the memory usage: Configuration 3 has a memory usage of 702.4 MB, Configuration 2 a memory usage of 549.7 MB and Configuration 1 a memory usage of 313.9 MB. We can see a connection between these data and the capabilities of each configuration, since despite having lower memory consumption, as we will see below, this does not imply that the performance of that configuration will be better. With respect to CPU usage, we can see how Configuration 3 presents the best percentage of CPU usage with 0.18 %, while Configuration 2 presents a CPU usage of 0.54 % and Configuration 1 an usage of 4.3 %. With respect to the processing time of the Esper CEP engine, Configuration 1 is the one that presents the lowest performance with an average latency of 0.229 ms, while the other two configurations present similar values, being 0.034 ms for Configuration 2 and 0.035 ms for Configuration 3. Finally, we must mention the total number of complex events detected, in which for

Table 2
Results of the performance tests.

Incoming rate (events/s)	**Configuration**	**Throughput time (min)**	**Memory usage (MB)**	**CPU usage (%)**	**Processing time (ms) (Latency)**	**Total complex events detected**
72	Configuration 1	11	313.9	4.3	0.229	2 725
	Configuration 2	11	549.7	0.54	0.034	2 677
	Configuration 3	11	702.4	0.18	0.035	2 659
720	Configuration 1	12:53	329.4	16.77	0.11	25 206
	Configuration 2	11	592.6	3.29	0.029	25 051
	Configuration 3	11	729.2	0.95	0.025	25 357
7 200	Configuration 1	–	–	–	–	–
	Configuration 2	20:19	690.5	7.68	0.0129	255 825
	Configuration 3	11	2 969.6	3.89	0.009	251 113
14 400	Configuration 1	–	–	–	–	–
	Configuration 2	–	–	–	–	–
	Configuration 3	16:22	4 198.4	3.85	0.006	492 374

Configuration 1 we obtain 2 725 complex events, 2 677 complex events in Configuration 2, and 2 659 complex events in Configuration 3. Such slightly difference in values in this indicator is due to the fact that, despite using the same configuration for the emulator, we cannot control the exact order in which all the events reach the system. Since many of the event patterns depend on the event order of arrival, this results in different values of complex output events, but the computation required is equivalent for the three configurations.

Concerning the input rate of 720 events/s, we can see that Configuration 1 fails the test, with a throughput time of more than 12 min. For this input rate we find similar values to those obtained for the previous input rate: the highest memory consumption was reached by Configuration 3 with 729.2 MB, followed by Configuration 2 with 592.6 MB and finally Configuration 1 with 329.4 MB. With respect to CPU usage, we can see that there is an overload in Configuration 1 with a CPU usage of 16.77 %, followed by Configuration 2 with a CPU usage of 3.29 %; the best result is obtained in Configuration 3 with a CPU usage of 0.95 %. With respect to the processing time of the Esper CEP engine, the worst result is found for Configuration 1 with an average latency of 0.11 ms, followed by the other two configurations with similar values, with an average latency of 0.025 ms.

With respect to the input rate of 7 200 events/s we can see how Configuration 2 fails the test with a throughput time of 20 min. In terms of memory usage, Configuration 3 required 2969.6 MB, a much higher amount of memory than the Configuration 2 usage of

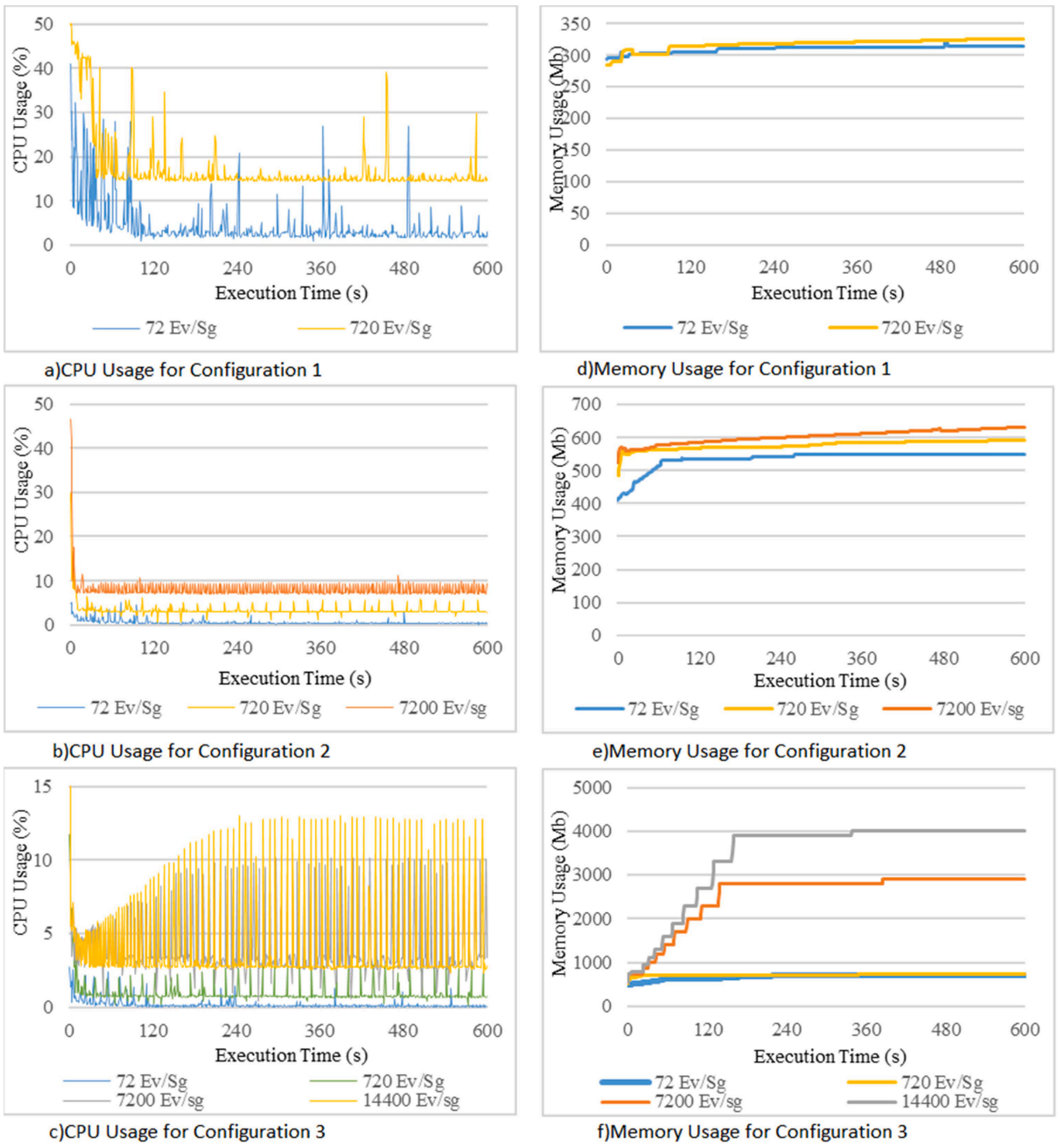

Fig. 5. CPU and memory usage graphics.

690.5 MB. In terms of CPU usage, Configuration 3 had a CPU usage of 3.89 % and Configuration 2 a CPU usage of 7.69 %. If we analyse the processing time of the Esper CEP engine, we can see that Configuration 2 has a higher average latency of 0.0129 ms, while Configuration 3 has an average latency of 0.009 ms. Finally, with respect to the total number of complex events detected, Configuration 2 detects 255 825 and Configuration 3 251 113.

Finally, only Configuration 3 was tested for the input rate of 14 400 events/s, but it failed to pass it, with a throughput time of 16 min. It had a memory consumption of 4 198.4 MB, a CPU usage of 3.85 %, an average latency of 0.006 ms and a total number of 492 374 complex events detected.

To better visualise the results, Fig. 5 shows the data for all the input rates for each configuration individually. Specifically, Fig. 5a), b), and c) represent the average CPU consumption for the different input rates of each configuration. We can clearly observe how the CPU consumptions remain stable for each of the configurations according to their technical limitations, but as we will see below, clearly influenced by the available memory in each device.

In Fig. 5d), e), and f), the memory consumption in MB for each configuration and input rates are represented. As can be seen, the memory consumption remains very stable in all the tests performed. A system crash occurs in Configurations 1 and 2 at low input rates, as the maximum free memory is quickly reached in these configurations. The available memory clearly affects the processing capacity of the system, because depending on the memory capacity, the system will be able to process a larger number of event patterns at the same time. As previously said, the tests were done with several patterns expected to consume plenty of resources, especially because they might be a long time in memory waiting for the arrival of a particular event. For this reason it is always advisable to evaluate the resource needs of our system depending on the patterns we use. This question is out of the scope of this paper, but previous work on performance studies of various Esper CEP operators can be consulted [3,49].

Table 3 shows the results of the tests carried out to analyse how the complexity of the JSON structure and the dependences among event patterns affect the system performance. In this test we exposed the system to a high rate of input events in order to overload it and analyse the results according to the complexity of the event pattern used in each test. It is worth mentioning that none of the 3 tests finished within the 11 min of testing. As we can see, the results are in stark contrast between the three tests. The test that obtained the best processing time was the one without attribute nesting or dependant complex events, obtaining a processing time of 0.0078 ms on average. While the test in which five attributes were nested obtained a processing time of 0.0089 ms and the test with five dependant complex events obtained a processing time of 0.4902 ms. Despite all the tests being performed on the same configuration, we also get very contrasting results in terms of percentage of CPU usage and memory usage. The test without nesting had the worst data for memory usage, 1331.2 MB, and CPU usage, 3.91 %. It was followed by the five-attribute nesting test with a memory usage of 1126.4 MB and a CPU usage of 3.22 %. While the five complex event nesting test, which had the worst processing time by far, had the best memory usage of 734.3 Mb and CPU usage of 1.95 %.

It is worth noting that the significant difference between the performance of the patterns without complex event nesting and the pattern with complex event nesting is mainly due to the fact that when a simple event arrives and triggers the activation of the various complex events, for each simple event that arrives, five event patterns are activated. The activation of these event patterns entails that the system cannot process any more simple events at the same time, which means that by processing fewer simple events at the same time, the reserved memory and CPU usage is much lower, but nevertheless the processing time increases exponentially. Likewise, the difference in performance results between the event pattern without nesting and the event pattern that nests attributes is due to the same reason just explained, since the second event pattern requires more time to process each single event, it can process fewer individual events.

7. Discussion

This section will analyse the results presented in Section 6, assessing in which cases each of the configurations used for the different tests would be useful. We also discuss some limitations and advantages in the use of the taxonomy and software architecture proposed, emphasizing the benefits of the contributions in this paper.

7.1. Discussion of the evaluation results

Regarding Configuration 1, i.e., the configuration in which the CEP engine is deployed in a Raspberry Pi, we can see that with the current number of event patterns deployed, it is hardly not able to correctly handle 720 events/s. It is not a high input rate, however, it is perfectly suitable for the number of events that can take place in an individual home, even more taking into account, as we have

Table 3
Results of the complexity tests.

Incoming rate (events/s)	Pattern	Memory usage (MB)	CPU usage (%)	Processing time (ms) (Latency)
10000	No nesting	1 331.2	3.91	0.0078
	Nesting of 5 attributes	1 126.4	3.22	0.0089
	Nesting of 5 complex events	734.3	1.95	0.4902

already said, that the tested patterns have a high resource consumption. Assuming that Configuration 1 correctly processes 72 events/s, and that not all sensors generate information per second, but rather generate information when their state changes, it could efficiently support a home with more than 72 sensors installed in it.

Secondly, with regard to Configuration 2, i.e., the configuration in which the CEP engine is on a PC, we can see how the system is able to correctly process 720 events/s with the current number of event patterns deployed in it. A configuration with a CEP engine of this capacity would be well suited to the needs of a private housing estate, residence or group of houses with approximately 10 properties and about 72 sensors per house, efficiently processing all the flow generated. Again, depending of the set of patterns used, the consume can be lower or higher and can be adapted to the needs of the system in question.

Thirdly, concerning Configuration 3, i.e., the configuration in which the CEP engine is running on a high-capacity server, we can see how it is perfectly capable of correctly processing 7 200 events/s with the current number of event patterns deployed in it. A configuration with a CEP engine of this capacity would be well suited to the needs of neighbourhoods or sections of a city with approximately 100 properties and about 72 sensors per property sending information every second. We have to take into account that normally the sensors are not sending information every second, but they send the information in a more spaced way. Therefore, the system would properly work for a greater number of properties; moreover, the conditions to be monitored at the city level probably would not depend on so many individual sensors but on a particular set of local sensors.

After analysing the tests of the three configurations, we must pay special mention to the data obtained in the system complexity tests. From the results obtained, we can see how the design of the event patterns, as well as the data structure, has a great importance in the performance of the system. From the results obtained, we can estimate that the system processing time increases by 2.896 % for each attribute that is nested in the structure, while the system processing time is increased by 1235.792 %. for each dependant complex event that is nested. Therefore, based on these results, we must pay attention to a good design of the event patterns and to the data structure of our system since they are of vital importance for its performance.

7.2. Advantages and limitation

It is important to mention that the system is horizontally scalable making use of additional machines with the CEP engine, but there is a limitation: the system can be scaled as long as the event patterns that are analysed in each CEP engine do not have dependencies between them. As an example, in our case study, in order to scale the system horizontally without this limitation, it would be necessary to ensure that each CEP engine always receives data from the same users. Otherwise, there would be a loss of system integrity.

Revisiting the contributions of this paper, special mention should be made to the facility on the definition of the different types of events for different domains and their integration with the proposed taxonomy, first contribution of this paper. In this way we see that the taxonomy will facilitate the improvement of collaborative smart environments: having a common definition scheme for different types of events from various application domains will allow collaboration between domains by sharing events from each of them in a structure and format that others will be able to understand, and therefore integrate into their systems. Besides, thanks to the use of Esper CEP with a single event schema according to the taxonomy, the three types of events defined are easily defined and integrated with each other. Furthermore, by using a CEP-base software architecture, second contribution of this paper, and due to the use of the same schema for the different domains, we can easily process and correlate information from our domain as well as from other domains in the same event pattern. This fact further facilitates these collaborative environments that can benefit not only from a common taxonomy, but also from a pre-defined architecture that allows data to be shared through message queues and processed, if desired, through the use of CEP, therefore. providing a step forward towards C-IoT. Besides, through the use of the taxonomy field *Service*, the system will be able to carry out the different contextualised actions defined by the interested parties. That is, by using this field, the system can perform actions in response to different situations detected with a wide range of possibilities, in combination with its implementation in the system.

Additionally, we have aimed to make the proposed taxonomy highly reusable in several domains and extensible for specific needs, and we have shown through the study case, the third contribution of this paper, how it can effectively be used for several domains as well as to define both domain and context events.

Finally, through the performance evaluation, the fourth contribution of this paper, we have proven that the system can be used in machines of different capacity and cost, according to the specific needs of each application or stakeholder.

7.3. Using the proposal in conjunction with other processing technologies

As previously mentioned, the proposed architecture includes the use of CEP for the detection of situations of interest. With this solution we intend to provide a self-contained architecture that the developer can use in the domain in question, but of course CEP could be replaced or enhanced by other technologies and, therefore, the proposed architecture can be integrated with other processing technologies that the developer is willing or has the need to use. As an illustration, CEP could be replaced or complemented by ML technologies, which could be used for multiple purposes. For example, ML technologies could be used in place of CEP, to predict situations of interest rather than detect them [53,54]. Or ML techniques could be used together with CEP to detect some situations of interest with CEP and predict others with ML, to correlate ML predictions with complex events detected by CEP aiming to improve system recommendations, to use ML to resolve conflicts in the case that several CEP event detections result in contradictory responses, to improve continuous learning by detecting with ML CEP patterns that need to be improved because they are not detecting relevant situations that have happened or because they have detected irrelevant situations, to use ML to help infer new CEP patterns needed in the system, among others(some examples of such use are shown in [54–59].

In Fig. 6, we see how other processing techniques could be used in the proposed architecture, in the left hand side we see how we could extend the taxonomy to add ML techniques, and in the right hand side we represent, in this case, how CEP and ML techniques could be executed in parallel. Logically, this might entail some minor adjustments into the implementation code of the architecture, which will be explored in our future work.

8. Comparison to related work

We did not find many proposals in terms of context-aware applications for IoT and smart environments that make use of taxonomies; in the following lines we describe the more relevant ones.

There are taxonomy proposals that assume a definition of context that is drastically different from our approach or that is very focused on a specific domain. For instance, Zimmermann et al. [25] propose a 3-part definition of context with the aim of improving the relationship between users and developers. They make use of a classification with categories such as time, location and activity. They also classify non-tangible aspects such as relationships between entities and the influence of other users. Also, context sharing deserves mentioning: they identify "shared contexts" when two entities share certain parts of the context information. They are therefore assuming that all the entities involved belong to the same party and therefore they can provide information about one or more contexts, viewed from different perspectives. In addition, if a group of entities share some parts of the context, they would also share as a group knowledge how this type of context is dealt. Our approach does not assume that the entities sharing context belong to the same party, but provides the means for multiple parties to share information as context. On the other hand, their context definition is very general, and they themselves refer to it as "appearance" of context. Furthermore, this model does not allow for the classification of devices or sensors, what is key in the scope of IoT. Kofod-Petersen et al. [18] proposed the use of Activity Theory to first model the context, and then evaluate it in pervasive situations. For this purpose, they define a taxonomy that corresponds to the basic structure of an activity in Activity Theory and contains information about the acting subject. By focusing on the subject and the activity performed, this classification makes it possible to represent any activity in a simple and complete way. But by being too activity-centred, it neglects aspects of the technological infrastructure that are very relevant for our application domain.

We also find other definitions focused on very specific application domains, such as the one from Belkadi et al. [23] which is mainly focused on describing collaborative situations from multiple views at different organizational levels. In it, a model for collaborative design is put forward with the aim of increasing context awareness. To this end, a framework for collaborative design is defined, in which entities relate to other material entities in a series of collaborative interactions. In addition, a generic situation model is defined for entities and activities, which as an abstraction allows little detail on the entities but defines in a very interesting way the interactions between them. We can also mention the work from Mosqueira-Rey et al. [16] where two product-focused taxonomies are proposed. One taxonomy describes the usability of the product and the other the interaction of the user with the product. As they are specific taxonomies, they allow a great level of detail with aspects such as training and experience in the use of the product in question. However, these two approaches, being focused on very specific domains such as collaborative design and the product respectively, do not allow us to apply the taxonomy to the broader domain of IoT. Also Mountrouidou et al. [60] define a taxonomy for IoT based on generic blocks common in IoT. They present classes such as Device, Communication channels, Mobility, Purpose, Sensor and Actuator. Any IoT device can be represented through this taxonomy. However, as it only allows to classify devices and their communications, this prevents its use for more general domains. Another proposal is the one made by Hemmati et al. [61], in which they analysed the different existing works in many domains of the Internet of Autonomous Things. They highlighted the lack of taxonomies for some of

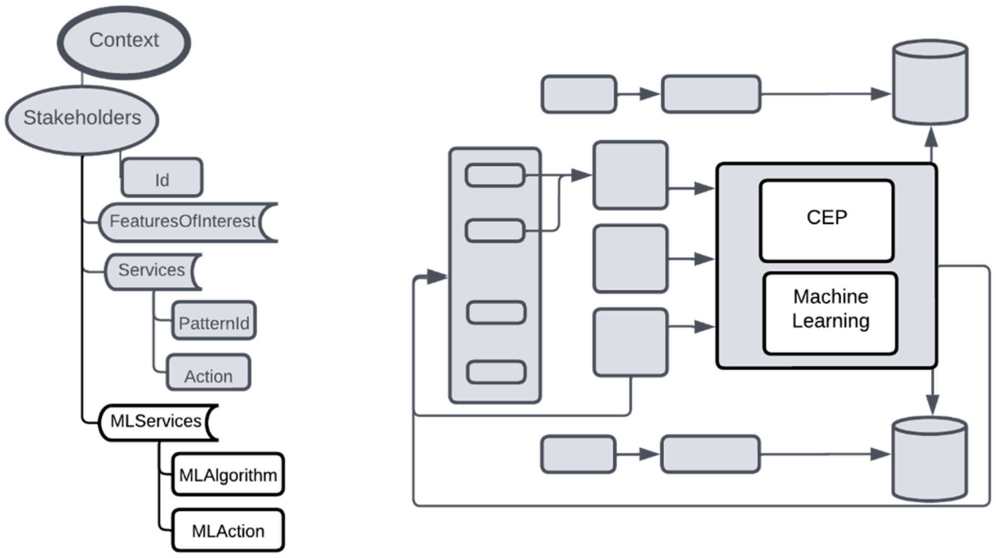

Fig. 6. Taxonomy and architecture enhanced with machine learning.

its different domains. It should be noted that the taxonomies proposed in all these approaches may be of interest in specific domains, but they address individual fields and do not serve as a joint technique for different IoT and smart environments application domains.

On the other hand, it is worth mentioning the proposal by Anagnostopoulos et al. [19], where context awareness is introduced in mobile computing, receiving sensor data from mobile devices and making use of a software architecture for their processing. It indicates many aspects that should be taken into account for modelling mobile computing, and although it does not present a detailed model for data representation, they focus mostly on modelling the relationship of people with mobile devices in a location and activity. If mobile technology is to be used specifically, its use is highly recommended, but this is not our case, as we seek to cover different domains, not just mobile computing. Focused on fog and edge computing, a taxonomy for resource management techniques of artificial intelligence (AI)-based edge and cloud computing is proposed by Iftikhar et al. [62]. The designed taxonomy presents specific classes for AI metrics and methods, as well as for different target applications such as smart cities, healthcare and transportation, in addition to including infrastructure. They make a very interesting analysis and proposal, from which we can extract useful information and possible lines of future work for our taxonomy, making use of AI for certain parts of it. As the application domain is edge and fog computing, its adaptation to our domain is certainly complex; since fields such as Stakeholder and Activity, among others, are not applicable in their domain.

Besides, Zhou et al. [24] provide a proposal closely related to ours: an ambient intelligence system oriented to home care for the elderly. Their taxonomy is composed of classifications very similar to ours, such as sensors and devices, activities or events, and context or situation, which would be equivalent to aspects such as our location and date. Finally, they provide a classification of knowledge and rules, where they define rules in the system to provide assistance according to the situation. This would be equivalent to our approach of event patterns with a CEP engine, where if we detect certain actions we generate certain alerts, but by making use of our *Services* feature, with which we can define custom actions for each user. In addition, for systems that manage large amounts of data in real time, CEP is an option that provides competitive performance [63]. It must also be said that despite being a good proposal, we go further, seeking to be able to process data from different application domains with our software architecture. This allows us to offer the user better personalization and contextualization.

Finally, in some other research areas, such as Business Process Management and Process Mining, context awareness and smart environments have also started to be addressed from the perspective of business processes. Among the works that can be found, it is worth highlighting the proposal by Bertrand et al. [64], where an XML standard is proposed with the aim of integrating IoT ontologies and event logs. The data format consists of lists of data, objects and events. An event can be related to several objects that can be digital or physical or both. All Objects can have a collection of Properties, which can also be digital or physical, and represent contextual parameters of the process. Events, which can be IoT events, process events and context events, are derived from one or more lower levels data inputs or events recorded by a given data source, which can be an information system or a sensor.

We have considered it necessary to analyse whether these approaches meet a number of features relevant to the needs we find in the IoT and smart environments. These features, as already motivated along the paper, are whether they provide a taxonomy or ontology, whether they are applicable to IoT domains, and whether they are specific to a particular IoT-related domain or for general use in the IoT, whether they use rules to analyse information and detect situations of interest, whether they process information in real time, and whether they provide an architecture or framework to support the developer. We have checked the features offered by each of the proposals discussed in the related work; the results are shown in Table 4. The following explains what each column of the table represents:

- *Approach* indicates the approach to be analysed.
- *Tax.* indicates whether a taxonomy is provided.
- *Ont.* indicates whether an ontology is provided.
- *IoT Dom.* indicates whether the approach presents a classification of the IoT domain.
- *Domain* indicates the domain for which the proposal is designed, either more specific or more general.
- Rule indicates whether rules to analyse the information received and detect situations of interest are used.
- Real Time indicates whether real-time decision making is performed.

Table 4
Related work comparative.

Approach	Tax.	Ont.	IoT Dom.	Domain	Rule	Real time	Arch.	Frame.
Zimmermann et al. [25]	×	×	✓	Context aware applications	×	×	×	×
Kofod-Petersen et al. [18]	✓	×	✓	Pervasive computing	×	×	×	×
Belkadi et al. [23]	×	×	×	Collaborative design	×	×	×	✓
Mosqueira-Rey et al. [16]	✓	×	×	Product usability	×	×	×	×
Mountrouidou et al. [60]	✓	×	✓	IoT devices	×	×	✓	✓
Anagnostopoulos et al. [19]	×	×	✓	Mobiles	×	×	×	×
Iftikhar et al. [62]	✓	×	×	Cloud/Edge	×	✓	✓	✓
Zhou et al. [24]	✓	×	✓	AmI	✓	×	✓	✓
Bertrand et al. [64].	×	×	✓	IoT	✓	✓	×	✓
Yilmaz et al. [34].	×	✓	✓	Ambient Assisted Living	✓	✓	✓	×
Degha et al. [35]	×	✓	✓	Smart building and energy efficiency	✓	×	✓	×
Our proposal	✓	×	✓	IoT and Smart Environments	✓	✓	✓	×

- Arch. indicates if the proposal provides a software architecture to give support to the developers in processing and correlating contextual data.
- Frame. indicates if the approach supplies a framework to give support to the developers in processing and correlating contextual data.

As Table 4 shows, most of the works studied present a taxonomy and are applicable to an IoT domain, although some of them for very specific domains, which allows little reusability. However, only one proposal offers rules for analysing the information received, and another one analyses the information in real time. As to whether they present an architecture or a framework, there is some division between those that present neither, and those that present both architecture and framework; while there is one proposal that does not present an architecture but does present a framework. Our proposal provides a taxonomy that is intended to be applicable to any IoT and smart environment domain as well as a software architecture that allows defining the rules to detect situations of interest in real time in the domain in question. In the future we aim to provide a framework with more functionalities that facilitate the use of the taxonomy and the software architecture.

9. Conclusions

Today, IoT ecosystems continuously generate large amounts of data, the exploitation of which for comprehensive use in the context of the smart environment paradigm is hindered by two main challenges. Firstly, the lack of data homogeneity and of taxonomies or ontologies that allow defining data from a wide range of IoT domains, but with adequate detail for each domain. And secondly, the shortage of tools that allow the contextualization of situations by making use of data from various application domains. To meet these challenges, in this paper we have proposed a taxonomy and JSON-based structure for the homogeneous definition of domain and contextual data as well as a software architecture that integrates the data received with such format and that processes and correlates them by means of CEP. This way, several heterogeneous data from different domains can be defined, submitted, processed and correlated in real time through the use of the provided taxonomy and software architecture, without the need of making use of an ontology. Furthermore, through the conjunction of the taxonomy and CEP it has been possible to match the capacity of the inherent contextualised actions of the ontologies. On the other hand, the use of the taxonomy-compliant common JSON structure to define domain and contextual data, has been shown to be suitable to be adapted to different application domains, fostering the collaboration between the different entities of the society. To adhere to the proposed system, it is only necessary to use the format defined in the JSON structure, thus facilitating the integration of multiple existing intelligent systems and environments to make the system more widely accepted.

As future work, the possibility of using an ontology according to the defined taxonomy will be studied to create a system with a higher level of formality. We will also examine the possibility of integrating ML technologies into the taxonomy and architecture, as discussed in Section 7.3. It is also expected to develop a framework to support the whole development process. This framework will make use of a Domain Specific Language to facilitate the work of the employees of the different companies that make use of our system in the task of adapting the data to the taxonomy-compliant JSON structure as well as to facilitate the adoption of the taxonomy for their specific needs. In addition, this framework could be used to easy the development of the event patterns for the system.

Furthermore, it will be important to properly implement different aspects of the security of the system, since the system may deal with different sensitive user data in a shared environment. Therefore, making this data anonymous and secure for everyone who can access the system should be one of the pillars of the future progress of the architecture. Finally, the possibility of testing the architecture and taxonomy in a real domain will be explored.

CRediT authorship contribution statement

Adrian Bazan-Muñoz: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation. **Guadalupe Ortiz:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Funding acquisition, Conceptualization. **Juan C. Augusto:** Writing – review & editing, Supervision, Methodology, Investigation, Conceptualization. **Alfonso Garcia-de-Prado:** Writing – review & editing, Validation, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work was supported by AwESOMe Project PID2021-122215NB-C33 funded by MCIN/ AEI /10.13039/501100011033/ and by

ERDF A way to do Europe and partially by the Ministry of Economic Transformation, Industry, Knowledge and Universities of the Andalusian Regional Government though DECISION project with reference P20_00865. The first author also acknowledges the pre-doctoral program of the University of Cadiz, Spain (2022-006 / PU / EPIF-FPI-CT / CP).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.iot.2024.101160.

References

[1] J.C. Augusto, H. Nakashima, H. Aghajan, Ambient intelligence and smart environments: a state of the art, in: H. Nakashima, H. Aghajan, J.C. Augusto (Eds.), Handb. Ambient Intell. Smart Environ., Springer US, Boston, MA, 2010, pp. 3–31, https://doi.org/10.1007/978-0-387-93808-0_1.
[2] F. Behmann, K. Wu, Collaborative Internet of Things (C-IoT): For Future Smart Connected Life and Business, John Wiley and Sons, Inc, Hoboken, 2015.
[3] A. Garcia-de-Prado, G. Ortiz, J. Boubeta-Puig, COLLECT: cOLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things, Expert Syst. Appl. 85 (2017) 231–248, https://doi.org/10.1016/j.eswa.2017.05.034.
[4] H. Chegini, A. Mahanti, A framework of automation on context-aware Internet of Things (IoT) systems, in: Proc. 12th IEEEACM Int. Conf. Util. Cloud Comput. Companion, Association for Computing Machinery, New York, NY, USA, 2019: pp. 157–162. https://doi.org/10.1145/3368235.3368848.
[5] A. Akbar, A. Khan, F. Carrez, K. Moessner, Predictive analytics for complex IoT data streams, IEEE Internet Things J. PP (2017) 1, https://doi.org/10.1109/JIOT.2017.2712672. –1.
[6] G. Ortiz, M. Zouai, O. Kazar, A. Garcia-de-Prado, J. Boubeta-Puig, Atmosphere: context and situational-aware collaborative IoT architecture for edge-fog-cloud computing, Comput. Stand. Interfaces 79 (2022) 103550, https://doi.org/10.1016/j.csi.2021.103550.
[7] T. Domínguez-Bolaño, O. Campos, V. Barral, C.J. Escudero, J.A. García-Naya, An overview of IoT architectures, technologies, and existing open-source projects, Internet Things 20 (2022) 100626, https://doi.org/10.1016/j.iot.2022.100626.
[8] J.C. Augusto, Contexts and context-awareness revisited from an intelligent environments perspective, Appl. Artif. Intell. 0 (2022) 1–32, https://doi.org/10.1080/08839514.2021.2008644.
[9] B. Schilit, N. Adams, R. Want, Context-aware computing applications. 1994 First Workshop Mob, Comput. Syst. Appl., IEEE, Santa Cruz, California, USA, 1994, pp. 85–90, https://doi.org/10.1109/WMCSA.1994.16.
[10] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, P. Steggles, Towards a Better Understanding of Context and Context-Awareness, Springer-Verlag, Karlsruhe, Germany, 1999, pp. 304–307, https://doi.org/10.1007/3-540-48157-5_29.
[11] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Context aware computing for the internet of things: a survey, IEEE Commun. Surv. Tutor. 16 (2014) 414–454, https://doi.org/10.1109/SURV.2013.042313.00197.
[12] J. Augusto, A. Aztiria, D. Kramer, U. Alegre, A survey on the evolution of the notion of context-awareness, Appl. Artif. Intell. 31 (2017) 613–642, https://doi.org/10.1080/08839514.2018.1428490.
[13] A.K. Dey, Understanding and using context, pers, Ubiquitous Comput. 5 (2001) 4–7, https://doi.org/10.1007/s007790170019.
[14] J.C. Augusto, M. Quinde, C. Oguego, J.G. Gimenez Manuel, Context-aware systems architecture (CaSA), Cybern. Syst. 53 (2022) 319–345.
[15] J.G. Gimenez Manuel, J.C. Augusto, J. Stewart, AnAbEL: towards empowering people living with dementia in ambient assisted living, Univers. Access Inf. Soc. 21 (2022) 457–476.
[16] E. Mosqueira-Rey, D. Alonso-Ríos, V. Moret-Bonillo, Usability taxonomy and context-of-use taxonomy for usability analysis, in: 2009 IEEE Int. Conf. Syst. Man Cybern., 2009: pp. 812–817. https://doi.org/10.1109/ICSMC.2009.5346929.
[17] X.H. Wang, D.Q. Zhang, T. Gu, H.K. Pung, Ontology based context modeling and reasoning using OWL, in: IEEE Annu. Conf. Pervasive Comput. Commun. Workshop 2004 Proc. Second, 2004: pp. 18–22. https://doi.org/10.1109/PERCOMW.2004.1276898.
[18] A. Kofod-Petersen, J. Cassens, Using activity theory to model context awareness, in: T.R. Roth-Berghofer, S. Schulz, D.B. Leake (Eds.), Model. Retr. Context, Springer, Berlin, Heidelberg, 2006, pp. 1–17, https://doi.org/10.1007/11740674_1.
[19] C.B. Anagnostopoulos, A. Tsounis, S. Hadjiefthymiades, Context Awareness in Mobile Computing Environments, Wirel. Pers. Commun. 42 (2007) 445–464, https://doi.org/10.1007/s11277-006-9187-6.
[20] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, K. De Bosschere, Towards an Extensible Context Ontology for Ambient Intelligence, in: P. Markopoulos, B. Eggen, E. Aarts, J.L. Crowley (Eds.), Ambient Intell., Springer, Berlin, Heidelberg, 2004, pp. 148–159, https://doi.org/10.1007/978-3-540-30473-9_15.
[21] H. Chen, T. Finin, A. Joshi, An ontology for context-aware pervasive computing environments, Knowl. Eng. Rev. 18 (2003), https://doi.org/10.1017/S0269888904000025.
[22] H. Chen, F. Perich, T. Finin, A. Joshi, SOUPA: standard ontology for ubiquitous and pervasive applications, First Annu. Int. Conf. Mob. Ubiquitous Syst. Netw. Serv. (2004) 258–267, https://doi.org/10.1109/MOBIQ.2004.1331732, 2004 MOBIQUITOUS 2004.
[23] F. Belkadi, E. Bonjour, M. Camargo, N. Troussier, B. Eynard, A situation model to support awareness in collaborative design, Int. J. Hum.-Comput. Stud. 71 (2013) 110–129, https://doi.org/10.1016/j.ijhcs.2012.03.002.
[24] F. Zhou, J.R. Jiao, S. Chen, D. Zhang, A case-driven ambient intelligence system for elderly in-home assistance applications, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 41 (2011) 179–189, https://doi.org/10.1109/TSMCC.2010.2052456.
[25] A. Zimmermann, A. Lorenz, R. Oppermann, An operational definition of context, in: B. Kokinov, D.C. Richardson, T.R. Roth-Berghofer, L. Vieu (Eds.), Model. Using Context, Springer, Berlin, Heidelberg, 2007, pp. 558–571, https://doi.org/10.1007/978-3-540-74255-5_42.
[26] M.L. Caliusco, C. Maidana, M.R. Galli, O. Chiotti, Contextual ontology definition metamodel, in: 4a Jorn. Iberoam. Ing. Softw. E Ing. Conoc. (2004).
[27] I. Durán-Muñoz, M.R. Bautista-Zambrana, Applying ontologies to terminology: advantages and disadvantages, HERMES - J. Lang. Commun. Bus. (2013) 65–77, https://doi.org/10.7146/hjlcb.v26i51.97438.
[28] IEEE Computational Intelligence Society, IEEE standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams, IEEE Std 1849-2023 Revis. IEEE Std 1849-2016 (2023) 1–55, https://doi.org/10.1109/IEEESTD.2023.10267858.
[29] E. Verbeek, OCED Standard - IEEE task force on process mining, (2023). https://www.tf-pm.org/resources/oced-standard (accessed February 1, 2024).
[30] Process and Data Science Group (PADS), Object-centric event Log 2.0, OCEL 20 (2022). https://www.ocel-standard.org/ (accessed February 1, 2024).
[31] B. Kitchenham, Procedures for performing systematic reviews, Keele UK Keele Univ. 33 (2004) 1–26.
[32] A. Forkan, I. Khalil, Z. Tari, CoCaMAAL: a cloud-oriented context-aware middleware in ambient assisted living, Future Gener. Comput. Syst. 35 (2014) 114–127, https://doi.org/10.1016/j.future.2013.07.009.
[33] P. Pradeep, S. Krishnamoorthy, R.K. Pathinarupothi, A.V. Vasilakos, Leveraging context-awareness for Internet of Things ecosystem: representation, organization, and management of context, Comput. Commun. 177 (2021) 33–50, https://doi.org/10.1016/j.comcom.2021.06.004.
[34] Ö. Yilmaz, An ambient assisted living system for dementia patients, Turk. J. Electr. Eng. Comput. Sci. 27 (2019) 2361–2378, https://doi.org/10.3906/elk-1806-124.

[35] H.E. Degha, F.Z. Laallam, B. Said, Intelligent context-awareness system for energy efficiency in smart building based on ontology, Sustain. Comput. Inform. Syst. 21 (2019) 212–233, https://doi.org/10.1016/j.suscom.2019.01.013.

[36] A.R.M. Forkan, I. Khalil, Z. Tari, S. Foufou, A. Bouras, A context-aware approach for long-term behavioural change detection and abnormality prediction in ambient assisted living, Pattern Recognit. 48 (2015) 628–641, https://doi.org/10.1016/j.patcog.2014.07.007.

[37] T. Gu, X. Wang, H. Pung, D. Zhang, An ontology-based context model in intelligent environments, in: Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference. San Diego, California, USA: The Society for Modeling and Simulation International (SCS), 18-21 Jan 2004: pp. 270–275.

[38] B. Ouissem, M. Lamia, M. Hafidi, A proposed ontology-based generic context model for ubiquitous learning, Int. J. Web-Based Learn. Teach. Technol. IJWLTT 16 (2021) 47–64, https://doi.org/10.4018/IJWLTT.20210501.oa4.

[39] J. Aguilar, M. Jerez, T. Rodríguez, CAMeOnto: context awareness meta ontology modeling, Appl. Comput. Inform. 14 (2018) 202–213, https://doi.org/10.1016/j.aci.2017.08.001.

[40] M. Elkady, A. ElKorany, A. Allam, ACAIOT: a framework for adaptable context-aware IoT applications, Int. J. Intell. Eng. Syst. 13 (2020) 271–282, https://doi.org/10.22266/ijies2020.0831.24.

[41] H.K. Ngankam, H. Pigot, S. Giroux, OntoDomus: a semantic model for ambient assisted living system based on smart homes, Electronics (Basel) 11 (2022) 1143, https://doi.org/10.3390/electronics11071143.

[42] A. Hasanov, T.H. Laine, T.-S. Chung, A survey of adaptive context-aware learning environments, J. Ambient Intell. Smart Environ. 11 (2019) 403–428, https://doi.org/10.3233/AIS-190534.

[43] J. El-Bouroumi, H. Guermah, M. Nassar, Enhancing business process modeling with context and ontology, Int. J. Adv. Comput. Sci. Appl. 12 (2021) 373–380, https://doi.org/10.14569/IJACSA.2021.0120942.

[44] European Union, Open EU Datasets | Official Portal for EU Data | data.europa.eu, (2023). https://data.europa.eu/data/datasets?locale=en (accessed February 1, 2024).

[45] A. Garcia-de-Prado, G. Ortiz, J. Boubeta-Puig, CARED-SOA: a context-aware event-driven service-oriented architecture, IEEE Access 5 (2017) 4646–4663, https://doi.org/10.1109/ACCESS.2017.2679338.

[46] D.C. Luckham, Event Processing for Business: Organizing the Real-Time Enterprise, John Wiley & Sons, 2011.

[47] Espertech, Esper documentation, (2023). https://www.espertech.com/esper/ (accessed February 1, 2024).

[48] D. Corral-Plaza, G. Ortiz, I. Medina-Bulo, J. Boubeta-Puig, MEdit4CEP-SP: a model-driven solution to improve decision-making through user-friendly management and real-time processing of heterogeneous data streams, Knowl. Based Syst. 213 (2021) 106682, https://doi.org/10.1016/j.knosys.2020.106682.

[49] G. Ortiz, I. Castillo, A. Garcia-de-Prado, J. Boubeta-Puig, Evaluating a flow-based programming approach as an alternative for developing CEP applications in IoT, IEEE Internet Things J. 9 (2022) 11489–11499, https://doi.org/10.1109/JIOT.2021.3130498.

[50] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, W. Retschitzegger, Context-awareness on mobile devices - the hydrogen approach., in: 2003: p. 292. https://doi.org/10.1109/HICSS.2003.1174831.

[51] I. Cristescu, J. Gimenez Manuel, J. Augusto, Assessing vulnerabilities in IoT-based ambient assisted living systems, in: J. Hernaandez Ramos, A. Skarmeta (Eds.), 64, 81q65, 2020: pp. 94–107. https://doi.org/10.3233/AISE200007.

[52] A. Garcia-De-Prado, nITROGEN: internet of Things RandOm GENerator., (2020). https://ucase.uca.es/nITROGEN/ (accessed February 1, 2024).

[53] J. Augusto, J. Giménez-Manuel, M. Quinde, Ch. Oguego, M. Ali, C. James-Reynolds, A smart environments architecture (Search), Appl. Artif. Intell. 34 (2020) 155–186, https://doi.org/10.1080/08839514.2020.1712778.

[54] G. Fortino, A. Guzzo, M. Ianni, F. Leotta, M. Mecella, Predicting activities of daily living via temporal point processes: approaches and experimental results, Comput. Electr. Eng. 96 (2021) 107567, https://doi.org/10.1016/j.compeleceng.2021.107567.

[55] G. Ortiz, J.A. Caravaca, A. García-de-Prado, F. Chavez de la O, J. Boubeta-Puig, Real-time context-aware microservice architecture for predictive analytics and smart decision-making, IEEE Access 7 (2019) 183177–183194, https://doi.org/10.1109/ACCESS.2019.2960516.

[56] P.C. Cañizares, S. Estévez-Martín, M. Núñez, SINPA: supporting the automation of construction planning, Expert Syst. Appl. 190 (2022) 116149, https://doi.org/10.1016/j.eswa.2021.116149.

[57] E. Brazález, H. Macià, G. Díaz, M. Baeza_Romero, E. Valero, V. Valero, FUME: an air quality decision support system for cities based on CEP technology and fuzzy logic, Appl. Soft Comput. 129 (2022) 109536, https://doi.org/10.1016/j.asoc.2022.109536.

[58] N. Mehdiyev, J. Krumeich, D. Enke, D. Werth, P. Loos, Determination of rule patterns in complex event processing using machine learning techniques, Procedia Comput. Sci. 61 (2015) 395–401, https://doi.org/10.1016/j.procs.2015.09.168.

[59] Y. Sun, G. Li, B. Ning, Automatic rule updating based on machine learning in complex event processing, in: 2020 IEEE 40th Int. Conf. Distrib. Comput. Syst. ICDCS, 2020: pp. 1338–1343. https://doi.org/10.1109/ICDCS47774.2020.00176.

[60] X. Mountrouidou, B. Billings, L. Mejia-Ricart, Not just another Internet of Things taxonomy: a method for validation of taxonomies, Internet Things 6 (2019) 100049, https://doi.org/10.1016/j.iot.2019.03.003.

[61] A. Hemmati, A.M. Rahmani, The internet of autonomous things applications: a taxonomy, technologies, and future directions, Internet Things 20 (2022) 100635, https://doi.org/10.1016/j.iot.2022.100635.

[62] S. Iftikhar, S.S. Gill, C. Song, M. Xu, M.S. Aslanpour, A.N. Toosi, J. Du, H. Wu, S. Ghosh, D. Chowdhury, M. Golec, M. Kumar, A.M. Abdelmoniem, F. Cuadrado, B. Varghese, O. Rana, S. Dustdar, S. Uhlig, AI-based fog and edge computing: a systematic review, taxonomy and future directions, Internet Things 21 (2023) 100674, https://doi.org/10.1016/j.iot.2022.100674.

[63] G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A. Garcia-de-Prado, I. Medina-Bulo, L. Iribarne, A microservice architecture for real-time IoT data processing: a reusable Web of things approach for smart ports, Comput. Stand. Interfaces 81 (2022) 103604, https://doi.org/10.1016/j.csi.2021.103604.

[64] Y. Bertrand, S. Veneruso, F. Leotta, M. Mecella, E. Serral, NICE: the Native IoT-centric event log model for process mining. Lect. Notes Bus. Inf. Process., Springer Verlag, Germany, 2023.