Journal of Computational Science 53 (2021) 101409

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

IA-CPS: Intelligent architecture for cyber-physical systems management

Henry Duque Gómez [a], Jose Garcia-Rodriguez [a,*], Jorge Azorin-Lopez [a], David Tomás [b], Andres Fuster-Guillo [a], Higinio Mora-Mora [a]

[a] Computer Technology Department, University of Alicante, Spain
[b] Department of Software and Computing Systems, University of Alicante, Spain

ARTICLE INFO

Keywords:
Cyber-physical systems
Internet of things
Service-oriented architecture
Event-driven architecture
Complex event processing
Microservices architecture

ABSTRACT

Nowadays, the emergence of large-scale and highly distributed cyber-physical systems (CPSs) in applications including Internet of things (IoT), cloud computing, mobility, Big Data, and sensor networks involves that architecture models have to work in an open and highly dynamic world. This fact increasingly highlights the importance of designing real-time and intelligent CPSs that are capable of decision making. In this paper, an Intelligent Architecture model for CPS management (IA-CPS) based on online processing is proposed for this purpose. Specifically, it can manage simple and complex events based on a service-oriented architecture and directed by an event-driven architecture, using event processing technology as Complex Event Processing (CEP). The novelty of our approach relies on the fact that the proposed architecture is service-oriented, which models the functionalities of the event-driven system. This gives us the possibility to offer a flexible service catalog, allowing us to connect to the system on any kind of device and interact in different scenarios. The model has been applied to two use cases: processing images from video surveillance cameras, and processing of consumption data captured by water and energy sensors installed in end-user environments.

1. Introduction

Cyber-physical systems (CPS) are devices that integrate computing, storage, and communication capabilities to control and interact with a process in the physical world. They are interconnected with each other and to the virtual world by global digital networks [1,2]. In other words, a CPS can be considered as a mechanism controlled or monitored by software-based algorithms and linked through the Internet, in which physical and software components are deeply integrated. Moreover, each component operates at different spatial and temporal scales [3]. Nowadays, the emergence of large-scale, highly distributed intelligent CPSs in the framework of the internet of things (IoT), cloud computing, mobility, Big Data, networks of interconnected devices and sensors, involves that software architecture models must work in an open and highly dynamic world [32]. This increasingly highlights the importance of designing real-time and intelligent CPS capable of decision making [4,5].

Broadly, there are three main aspects to classify the structural units for CPS tasks: a component architecture, a service-oriented architecture, and an agent-based architecture model [6]. These structural units have different performances in terms of their adaptability, autonomy, and interoperability properties [6,7]. These non-functional properties were proposed as critical in the challenges identified at the National Science Foundation (NSF) Cyber-physical Systems Summit [8].

To address these challenges, this paper proposes an Intelligent Architecture for CPS management (IA-CPS). This is an architectural model for intelligent processing and managing simple and complex events base on the Event-driven architecture (EDA) and integrated into a Service-oriented architecture (SOA) and microservices (MSA), which evolves to the SOA 2.0 concept [9,10]. EDA allows executing an action when it receives one or more event notifications [9,10], being able to react and make CPS devices interact with the environment through events and be processed by complex event processing (CEP) technology [11,12]. The novelty of our proposal relies on the fact that the proposed architecture is service-oriented, which models the functionalities of the event-driven system. This gives us the possibility to offer a flexible service catalog, allowing us to connect to the system on any kind of device and interact in different IoT and cloud scenarios.

The IA-CPS architecture has been applied to two use cases: the processing and management of images from video surveillance cameras, and the processing of consumption data captured by water and energy sensors installed in end-user environments.

* Corresponding author.
E-mail address: jgr@ua.es (J. Garcia-Rodriguez).

https://doi.org/10.1016/j.jocs.2021.101409
Received 1 April 2021; Received in revised form 17 June 2021; Accepted 18 June 2021
Available online 25 June 2021

The sections of the paper are organized as follows: Section 2 describes the state of the art and background of the software architecture models for real-time CPS tasks; Section 3 details the components and specifications of the IA-CPS architecture to provide intelligence to the systems; Section 4 presents the application use cases of the IA-CPS architecture; finally, the conclusions and future works are presented in Section 5.

2. Related works

The architectures for cyber-physical systems have evolved over time. Consequently, several classifications and taxonomies can be found in the literature with the aim of supporting model-based architectures. The basic architectures present unique characteristics in their functionality. We include in this category the architectures by components, by services, and by agents. These will be analyzed under the level of compliance with the properties of adaptability, autonomy, and interoperability.

This section analyzes the research works aimed to provide real-time CPS tasks, organizing them in three main approaches of architecture and software engineering for CPS mentioned above: components, services, and agents.

2.1. Architecture for component-based CPS

Real-time distributed managed systems such as DREMS (Distribute Real-times Managed Systems) are typical component-based CPS runtime support models. It is conceived for distributed and mobile scenarios, e.g., groups of satellites or swarms of unmanned aerial vehicles (UAVs) [13]. The DREMS architecture is composed of two subsystems: design-time development platform and run-time support platform [6, 13]. In the CPS component model in DREMS, two non-functional elements are highlighted that are as important as the main functional characteristics of a system: real-time features of CPS components, and independent and decoupled CPS components.

The RTCCM (Real-Time Container Component Model) [14] and the ARINC-653 (ACM - Arinc Component Model) [15] support real-time operations. Functionally, they are software specifications for space and time partitioning in safety-critical real-time air vehicle operating systems [6,14,15].

On the other hand, the independence of the components and the fact that they are not coupled to the system play a fundamental role in a model for CPS based on components. For example, DEECo (Dependable Emergent Ensembles of Components) [16] is a model aimed at working on the difficulties of large-scale distributed CPS-like dynamics or autonomy. A component system such as DEECo is characterized by its great independence. Critical technical problems in this type of model usually exist in aspects such as the implementation and deployment of CPS tasks based on components, or the reconfiguration of CPS tasks.

The construction of complex CPS systems is implemented through the composition of simple blocks (components), previously developed independently of the application in which they will be used [15]. In fact, the separation between interface and implementation proposed for CPS tasks [16] allows a component to be conceived as a black box that encapsulates services. In this way, it is not necessary to know its internal details to use it. It is only necessary to characterize its interface.

Furthermore, the CPS that operate in dynamic and unlimited environments are composed of multiple communication networks, controllers, sensors, and actuators that involve constant and dynamic changes given the behavior of the physical scenario in which they act. That is why aspects such as the reconfiguration of CPS tasks are very important and acquires greater relevance in those implementations based on components [17].

2.2. Architecture for service-based CPS

Some researchers have designed and implemented frameworks for real-time support of service-based CPS containing all the characteristics of the SOA [18]. It is worth highlighting the work of Martin et al. [19] who designed the OWL-S model (Web ontology Language for services), which is able to develop context- and resource-sensitive CPS services. Huang et al. [20] extended the model to provide the Context Sensitive Service Model (CSSM). This service model, based on the ontology of physical entities, is context-sensitive. The physical entities are organized hierarchically according to their relationships. In terms of context, it introduces two new constraints: the precondition of context and the effect of context. These are incorporated and treated as a traditional complement to the constraints of service provision [20].

Another important model is Physicalnet [21]. It is a generic framework for the management and programming of embedded and distributed sensor and actuator resources in a multi-user and multi-network environment. It was designed and implemented under a four-layered service-oriented architecture [21,22]: service provider, gateway, negotiator, and application level.

Finally, it is important to highlight RI-MACS (Radically Innovative Mechatronics and Advanced Control Systems) [23,24]. It is a model designed as an industrial automation proposal based on SOA and Web services. It is used in systems of new factories that automate processes under CPS concepts, which require real-time responses [23,24]. This framework presents a hardware and software infrastructure for industrial automation, which takes advantage of open technologies such as SOA, Ethernet-based communications, and real-time technologies. The implemented architecture is supported in communications that add network capacities of TCP/IP protocols with real-time traffic management [23,24].

2.3. Architecture for agent-based CPS

Several architectures designed for agent models use the Java library framework for the development of a set of agents called JADE (Java Agent Development Framework). The objective of JADE is to simplify the implementation of multi-agent systems through middleware that complies with the specifications of the Foundation for Intelligent Physical Agents (FIPA). It is aimed at defining standards for the interaction of agents [25].

Providing real-time support for CPS is a major challenge. Many models that provide tasks for CPS have time constraints and some low-level control tasks can be only executed on dedicated hardware. Among the framework for agent-based CPS, the following guarantee the real-time CPS: the Holonic Agent Model (HLA) [26] and the Rainbow Model (RM) [27]. The HLA is a multi-agent platform composed of three main modules: High-Level Control Module (HLC), Low-Level Control Module (LLC), and the Control Interface. The RM is a platform that allows the development of relatively easy applications for smart cities. It consists of three layers, designed to make calculations nearing the physical part: physical layer; distributed middleware, and level cloud.

3. Intelligent architecture for CPS management (IA-CPS)

The hybrid architectures are built from the basic models incorporating aspects that try to improve the shortcomings of the basic architectures (by components, services, and agents). With this approach, our proposal is composed of an event-driven architecture and those oriented to services and microservices, incorporating the best properties of hybrid architectures. The intelligent software architectures and the properties of adaptability, autonomy, and interoperability were proposed as critical to the challenges identified at the Cyber-Physical Systems Summit at the National Science Foundation NSF.

In this section, we focus on the need for an intelligent, scalable, flexible, and highly distributed, adaptable, autonomous, and

interoperable architecture in the context of IoT, cloud computing, mobility, Big Data, and interconnected sensor networks.

The requirement to design and implement intelligent architectures imposes major challenges that exceed the capabilities of the software developed so far, since there are no properly intelligent software models that adapt to changes in the physical scenario and make decisions in real-time.

To address the intelligence requirements outlined above, we propose an intelligent architecture as a reference framework, designed around the implementation of event-driven services and microservices with the functional ability to process complex events that can be configured, parametrized, and addressed, designing patterns that acquire intelligence. These patterns are designed and implemented according to a previous analysis of the behavioral routines of objects or people in a given scenario.

Intelligence relies on the way to infer new and more complex events, with more semantic and ontological meaning, from the sum of simple events. This makes it possible to implement relevant behavior patterns detected by sensor devices (video surveillance cameras, visual sensors, proximity sensors, etc.) installed in physical scenarios.

The specification of the proposed architecture is oriented to the learning of patterns so that from this learning, it can analyze the series of events and derive conclusions from them. This allows the system to react online and improve decision-making. In our case, intelligence is also provided by the software, since the intelligence designed through the patterns is recursive in the sum of capabilities of the different behaviors that can acquire an event exposed through a microservice. In this way, intelligence is due to the joint interaction of the components in each ecosystem in which the microservice acts (cyber-physical systems, the cloud, neural networks, artificial intelligence, robotics, etc.). Also, intelligence is given in the way services compete for the underlying information resources in sensor devices and the needs of competition for the infrastructure of the IoT environment and the cloud.

3.1. Architecture overview

The technologies for the specified AI-CPS architecture can be very varied and their application depends to a large extent on the logic of the use cases to be implemented. In any case, although the technologies are very changeable, this paper discusses, with the aim of completeness, a range of technologies that can be used for the implementation of AI-CPS. These technologies revolve around providing answers to the hybrid architecture between SOA services, MSA microservices and EDA CEP event-driven architectures. The technologies also revolve around the fulfilment of the best performance for the detailed use cases.

The approach of our IA-CPS proposal is based on the fact that microservice-based architectures (MSAs) are a subset of service-based architectures [28]. Furthermore, microservice architectures are not necessarily event-driven. While in an event-driven architecture (EDA), an agent is triggered by the arrival of an event [28], in microservices architecture an event can be triggered by an explicit call from a procedure. This implies clear differences. However, we have addressed both approaches to design a hybrid architecture taking advantage of the capabilities of them. Our proposal provides the necessary data consistency and reactivity to events of interest that are processed online to make intelligent decisions. Similarly, events are part of the communication model between microservices with the construction of a reactive model or the composition of capabilities through choreography and orchestration. The overall architecture shown in Fig. 1 complies with the paradigm and fundamental principles of an event-driven MSA and event-driven EDA service-oriented architecture, which is considered the state-of-the-art for managing and processing event streams. The proposed framework can process event streams from different sources that generate complex events.

Fig. 2 details the proposed architecture for the processing of events captured by the sensor devices, which are devices treated as asynchronous sensor services. The central axis of the IA-CPS architecture is

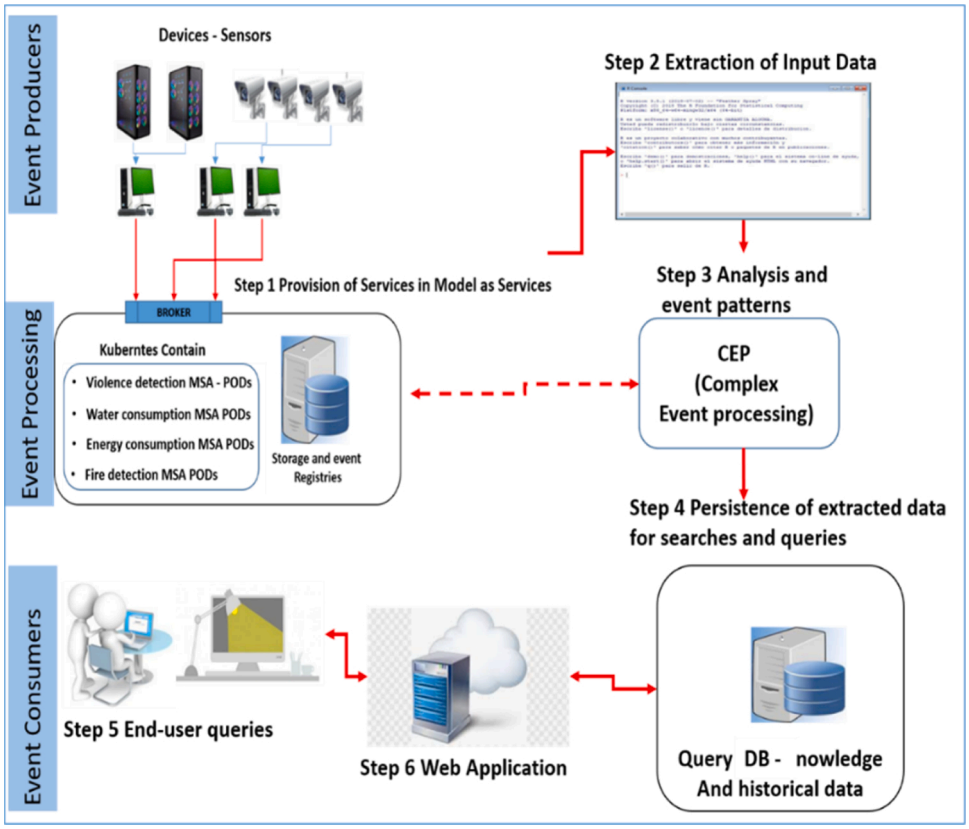

Fig. 1. Architecture overview EDA – MSA.

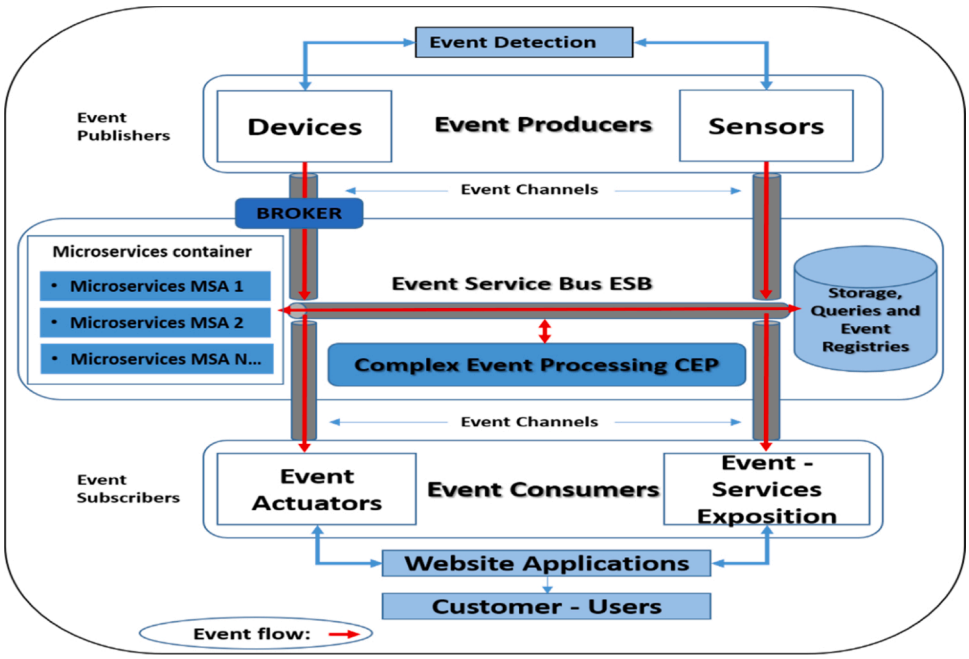

Fig. 2. System components and event flow.

composed of three main layers:

- Event producer layer. In the case of microservices, an event producer is responsible of generating semantic and ontological information and attending to the information captured by the environment's sensors. In short, it is responsible of perceiving and adapting the information from the environment. The flow of information goes through the event channels.
- Event processing layer. It is related to the analysis of the information captured by the sensors based on the logic of the microservice chosen by the end user. A database is implemented to store information for querying and reporting. Specifically, this is the Event Service Bus (ESB), which also hosts the cluster of containers that host microservices. It also contains the Complex Event Processing (CEP) [29–31] that is in charge of integrating the inputs and outputs of the data flows in the information sequences that enter the system, in addition to discovering and processing complex events, among others, related to semantics and ontologies.
- Event consumers layer. The event consumer deals with the management of events to generate response actions within the system itself and to external entities. In this layer the users of the system have an important role to play.

The novelty of our proposal relies on the fact that the architecture is service-oriented, which models the functionalities of the event-driven system. This gives us the possibility to offer a flexible service catalog, allowing us to connect to the system any kind of devices and interact in different IoT and cloud scenarios, such as smart cities [28,31]. However, the implementation of a service from the catalog will depend on the application area, which would require a prior analysis of the particularities of the new process.

3.2. Architecture specification

The specified modules are decoupled, managing special data processing when it comes to processing complex events captured by sensor devices. The information coming from the sensors is translated into events. Therefore, the interaction of the modules can occur in different periods and there is only a cause-and-effect relationship between the event emitted and the event consumed.

The proposed architecture supports a system that allows the management of open and technology-independent services in its operating environment. The logic of the architecture is managed around a broker that coordinates communications through messages, the microservices (the MSA), all based on event-driven logic (the EDA), and intelligent analysis employing patterns (the CEP).

The processing flow is not unique, since it depends on the logic implemented in the microservices and on the rule patterns configured in the CEP. The connection between modules is guaranteed by allowing a module to consume events from another module. This is subjacent to the interoperability and decoupling provided by the microservices.

3.3. Environment perception

In the proposed architecture, event sensors are responsible for sensing and obtaining information from the environment. The set of sensors in the scenario are the input to the event processing system. The broker extracts the information and reads the data streams from the sensor devices or some data storage (message queues or storage repositories) for subsequent delivery to the microservices.

3.3.1. Event detection

The higher layer modules (producers) publish messages to the middle layer modules in charge of processing. They have the logic to determine whether an event is simple or complex (the CEP). Also, they detect important state changes in the analyzed data frame, which could generate one or several important events for the system, triggering the operation of all the architecture modules, until the event is finally consumed by the lower layers (consumers) and reported for action. Events can also be shared with third-party applications to be processed under another platform or system, as explained above, which confers the model as Software as a Service (SaaS).

Our proposed architecture is a sensor event reaction model because event-driven architectures are event-reactive and with on-line processing. This characteristic is provided since they are asynchronous architectures. It must be considered that events are not always a single information entity; many events are configured from the combination of others.

3.3.2. Event producer module

From the events producer components, event information is obtained that must be processed and correlated to determine situations defined by the specific requirements of the system where it is to be implemented. It should be noted that different types of event-producing devices could be connected to the system. This is one of the advantages of implementing microservices to extract information from sensors in a standardized way. It is an important issue for the design and implementation of this module. Not all sensors generate events in the same format and must be transformed to deliver them to the event channel.

3.3.3. Logging of data sequences

The logging of the data captured by the sensors is performed by the action of the broker and the microservices (Fig. 3, step 1), which generate a log message queue, from which the information and data flows are extracted and sent to the CEP engine to analyze the events. The registration process in the message queue is the trigger that sets into action the modules that make up the proposed processing architecture (Fig. 3 step 2). The sending of information flows from the microservices to the event processing module is composed of two sub-processes:

- Presentation of metadata to the server. This metadata includes attributes such as sensor ID, geographic location, text description, date of registration, and time of registration.
- Sending the sequence of data (Fig. 3, step 3) through the microservices. Such information can be exposed to client applications, i.e., users of the particular systems instantiated from the architecture or third-party applications in the cloud, as we explained above. The information flow is stored and archived in databases for queries by system users (Fig. 3, step 4).

If the logging of the data captured by the sensors is successful, it is available in the system to be analyzed by the CEP engine, which has the logic to determine whether the event is simple or complex and what patterns it should implement and recognize. An advantage of our microservice-oriented and event-driven architecture is that sequences of event information can concurrently undergo the process of logging on multiple servers, sending the data to cloud structures for processing and analysis. This procedure could serve to share the event sequences to several interested institutions, also to store the information in other places, and to comply with aspects of continuity, flexibility, availability, and security of the information.

3.4. Information processing

In a microservices architecture managed by an event-driven architecture, the ESB module, which contains the microservices and the CEP event processor, corresponds to the main module. It has the full intelligence of the system through the autonomy, scalability, flexibility, and portability provided by the microservices, in service of the configuration of CEP event patterns that infer complex events with greater semantic and ontological meaning.

3.4.1. Communication channel

The event channel is the data communication channel, which allows linking the different modules of the proposed architecture. It delivers the sequences of events captured from the sensors through the publication of messages (event publisher) so that the processing and consumption modules can subscribe and consume events (event subscribers).

3.4.2. Event processing module

This module is composed of the microservices catalog, which will be provided by the particular system where it is implemented (Fig. 3, step 1) and the complex event processing module CEP (Fig. 3, step 3). The logic of dispatch, exception, update, and event detection is managed in this module. It also checks for errors that may occur in the parameters that unify an event. When an event message is received at the bus level, the publishing services (Fig. 3, step 5) intercept it and distribute it among the various event channels, making it available to consumers or to the subscription services component.

The event processor handles the parameterization and combinations of multiple event producers, i.e., multiple sensors connected to the system. It is important to highlight the transparent communication, the light, and stateless management of the event extracted by the broker and passed to the microservices for subsequent forwarding to the complex event processing module, in addition to the implementation through the

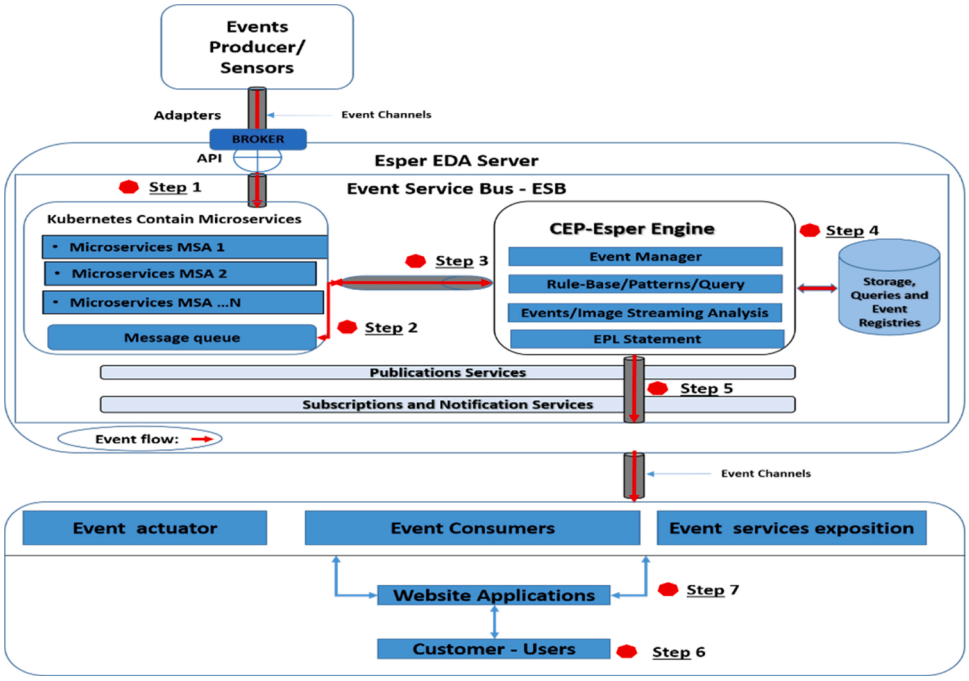

Fig. 3. System modules and data processing flow.

patterns that are intelligent features of the system and a critical aspect to process information flows and data provided by sensors.

These characteristics make microservices and CEP an intelligent entity and make it the core of the proposed event-driven architecture. The intelligence underlies the sum of the advantages offered by microservice-oriented and event-driven architectures. That is, our design takes the functional and non-functional characteristics of microservices and adds them to the analysis of patterns that operate under online information processing.

3.4.3. Data flow and processing

The microservices collect the captured data (Fig. 3, step 1) in a messages queue (Fig. 3, step 2). The CEP event manager module receives a request to generate an event for the sequence of information and data collected (Fig. 3, step 3). This request enters the rule-based engine for processing complete events. The queries and patterns are registered from the beginning of the processing in the CEP rule-based engine so that they are executed and act on each of the sequences of events that are continuously coming in. As the CEP engine finds the event sequences that match the predefined queries and patterns, events are identified in event sequences that will be consumed by the lower layers of our design (a graphical user interface or the lower layers of consumers).

The above approach allows processing sequences of data (events) as they arrive, easily scaling them horizontally so that the processing of complex events occurs in parallel to data storage (Fig. 3, step 4) and consumer or visualization layer. This activates all the intelligent processing logic implemented in the engine.

Fig. 3 describes the components, steps, and flow of data stream processing in the ESB, i.e., the interaction between the microservices and the CEP engine. The CEP engine contains the logic of the intelligent patterns that allow the analysis of detection, tracking, and recognition of events in the data sequences extracted by the broker and the microservices from the sensors. The rule base is configured with the intelligence to correlate and recognize detected events in data sequences in different spaces and times.

Once the CEP engine has analyzed the events through the analysis of the designed patterns, it publishes the sequence of events in the publication services layer, which are taken from the subscriptions and notifications services layer by the consumers (Fig. 3, step 5). This allows sensor events to be dispatched according to which consumers are interested.

The inherent intelligence of the CEP system lies in the way it infers new, more complex events with greater semantic meaning from the sum of simple events, which allows it to define patterns of detected meaningful behaviors. This allows the pattern to learn, and from this learning, to analyze the series of events and derive conclusions from them. All the above makes it easier to react, process information online, and improve decision-making.

The CEP engine also stores data streams in the database, both as logs of processed events and as knowledge files for historical data queries for end-customer consumer applications.

3.5. Action in the environment

The consumer has the logic to analyze and understand the event, the security scheme, and the response to such an event that has been processed online, which may be an event of interest to the system. These components react to events received from the ESB. The event consumer allows receiving or subscribing events (event subscriber) that respond to a pattern of behavior that is under analysis and for which the consumer has responsibilities to act and make decisions.

3.5.1. Event consumer module

Once the event is received, the consumer performs the tasks associated with the event (Fig. 3, step 6) to visualize the data and its appropriate analysis. It is necessary to update the processed data from a server that contains the CEP, to understand its values and make decisions. Among others, it is necessary to notify the occurrence of the events to the relevant instances.

The visualization and management of information through the event consumer module is implemented in a web architecture system that offers web services (Fig. 3, step 7), which involves the development of a page in a browser that shows a user interface. This has the logic of interaction of the users with the system and with the microservices they wish to subscribe to.

Fig. 4 shows the activation sequence diagram of the proposed architecture in the steps described in the preceding sections, as well as the data flows through the components of the specified IA-CPS architecture from the service side. In the same way, it shows from the client-side the interaction of the latter for the information query, allowing to observe the sequences to access the data query.

4. IA-CPS applications

The application of the use cases is developed under the framework of the proposed IA-CPS architecture. Specifically, we present an event-driven service architecture for managing video surveillance systems and a microservice-oriented architecture for managing sensors in smart environments connected to the cloud. The most relevant components of the system architecture are described below.

4.1. Use case – Event-driven architecture for video surveillance systems

Video surveillance cameras can describe observations of remote events. They are deployed in cities to monitor people's activity and movements, connected to the network to provide the video to a central location or a server for analysis. They serve as a deterrent, identification, and analysis tool for required events or events catalogued as of interest for a given scenario.

The processing of the images is subject to the patterns designed in the CEP, so that they allow correlating features of an image. Patterns are established that contain sections of the images with the desired features.

It is necessary to perform an intelligent analysis of images and audio to generate automatic alarms that strengthen the response time and improve the knowledge of different variables necessary to make decisions in real-time. As defined in previous sections, our perspective covers the analysis and decision making under on-line processing of the input events. In the following sections, we will present the basis of our research and proposal for an intelligent architecture for IA-CPS cyber-physical systems.

4.1.1. System architecture

The main components of the designed architecture are shown in Fig. 5. We will detail three modules: the video surveillance cameras as event producers, the ESB, which contains the CEP, and the event consumers. The event bus task is integrating the inputs and outputs of the information streams in the video and image sequence. Also, it can discover and process complex events, among others, related to semantics and ontologies. The event consumer manages events to generate response actions within the system itself and to external devices, clients, and users of the system. The information flows through the event channels.

The system design is conceived to produce, detect, and react to a flow of events in which the event producers, processors, and consumers are instantiated. We implemented a publish and subscribe communication system, which has the objective of sending streams of information and data captured by the video surveillance cameras to be received by one or more consumers (previously processed).

The processing flow depends on the logic of the rule patterns configured in the CEP to capture the image and video events that need to be analyzed in the case study. The connection between modules is ensured by allowing one module to consume events from another module through event channels. These are communication or data

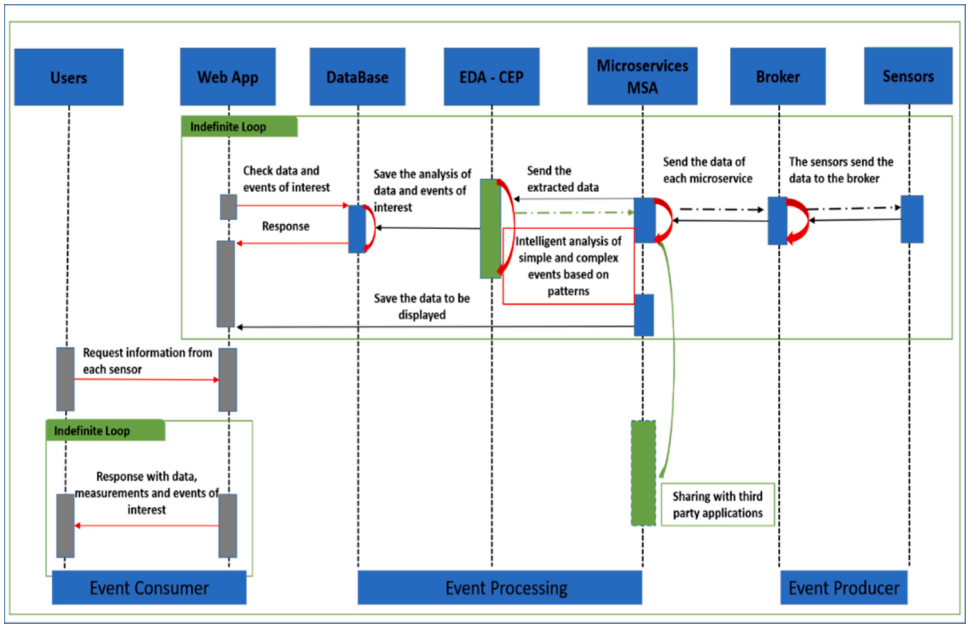

Fig. 4. Sequence diagram of event flow in IA-CPS specification and data query from client-side.

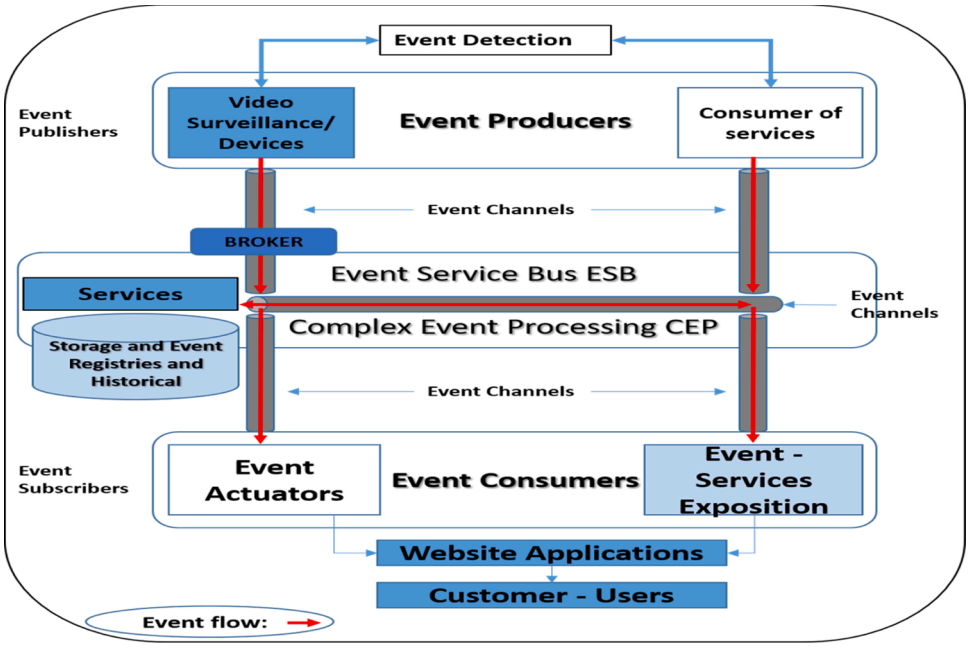

Fig. 5. Event processing flow for the case study.

navigation channels that allow linking the different modules of the proposed architecture. In other words, it delivers the sequences of events captured from the video surveillance cameras (event producers) through the publication of messages (event publishers) so that the processing and consumption modules can subscribe and consume events (event subscribers).

The system implements a RabbitMQ broker and the event channel through the AMQP tool. It is a tool with good characterization for sending messages, routing, and queuing them. It is an open standard that guarantees delivery, order, integrity, uniqueness, and security for transferring messages between event producers and receivers. AMQP is an open, interoperable, and reliable standard. Next, we will detail the flow of events, the sequences of images that navigate through the designed architecture, the functionality of the modules, the interaction between the components, and the information that flows through the architecture model shown in Fig. 6, which encompasses the proposed system in general. It is composed of the event producer module, the services in charge of capturing and recording events, the event bus, which contains the complex event processor CEP, and the event consumer.

4.1.1.1. Event producer module. From these components, event information is obtained that must be processed and correlated to determine critical situations, according to the logic defined in the case study for the video surveillance cameras. It should be noted that different types of event-producing devices could be connected to the system. Here the conversion or transformation of the data collected by the video surveillance cameras in a standardized way is an important issue for the

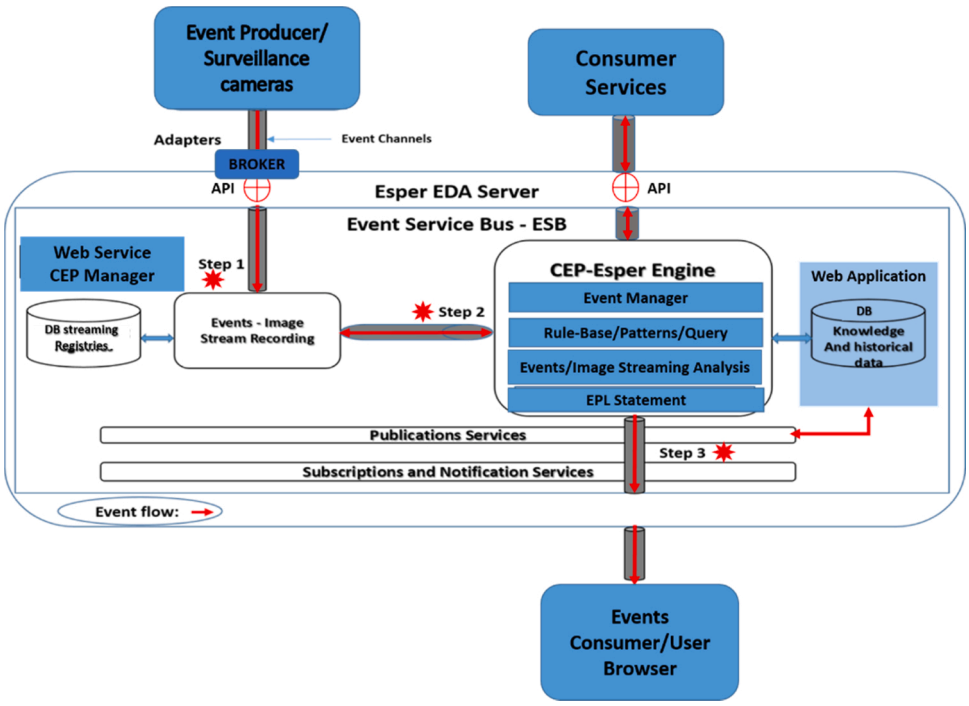

Fig. 6. Image processing components and flow.

design and implementation of this module. Not all cameras generate events in the same format and must be transformed to deliver them to the broker and the event channel. However, we mitigate this risk by using a messaging system and open SOA services.

The tool responsible of sending the data captured by the cameras to the event channel is RabbitMQ (broker). It is an open-source message sending tool for implementing producers and consumers. It fits perfectly with the event processor CEP-Esper, since both use Java classes (Plain Old Java Objects - POJO). RabbitMQ handles the publisher/subscriber scheme, in which messages are distributed and sent from the video surveillance cameras (producers) to all receivers and consumers in the system.

The registration of the images captured by the video surveillance cameras is done through the implementation of the CEP-Manager web service, which is installed on the Esper EDA server. The registration process is the trigger that sets into action the modules that comprise the proposed EDA architecture (Fig. 6, step 1).

By sending the image stream through RabbitMQ to the Esper EDA server, the data streams are exposed to the client applications, i.e., the users of the case study. The video stream is also recorded and archived in the image and video registry database.

Metadata is also submitted to the server. This metadata includes attributes such as: camera identification, video name, geographic location, text description, recording date, and recording time. Similarly, all the information specific to the image is submitted to the Esper EDA server, such as PTZ camera control, time, and location parameters.

If the recording of video captured by surveillance cameras is satisfactory, then, through the CEP-Manager web service, the sequence of images is available in the system to be analyzed by the CEP-Esper engine. It has the logic to determine whether the event is simple or complex and which patterns to implement and recognize (see event processing module).

4.1.1.2. Event processing module. This module manages the logic of event sending, exception, update, and detection. It also checks for errors that may occur in the parameters that unify an event. When we receive an event message at the bus level, the publishing services intercept it and distribute it among the various event channels, making it available to consumers or the subscription services component.

The event processor handles the parameterization and combinations of multiple event producers. That is, multiple video surveillance cameras are connected to the system. We highlight that pattern management is a critical aspect to process the images and videos sent from the video surveillance cameras. This feature makes the CEP-Esper engine an intelligent tool and makes it the core of our event-driven architecture.

The CEP-Esper engine allows defining simple and complex event patterns using Event Processing Languages (EPL). This engine can implement several events in different languages and supports the hierarchical creation of event patterns. Inside this module, we defined the tool that processes the complex events and generates the behavioral patterns, which engage ontological and semantic aspects.

Fig. 6 describes the components and the flow of the image sequence processing in the implemented register service and in the CEP-Esper engine, which belongs to the ESB. Therefore, regarding the flow and processing of images, once the video images captured from the video surveillance cameras are registered in the image sequence registration module (Fig. 6, step 1) described in the previous section, the event manager module of the CEP-Esper engine receives a request to generate an event for the registered image sequence (Fig. 6, step 2). This request enters the engine based on complex event processing rules.

Queries and patterns are registered from the beginning of the processing in the rule-based CEP-Esper engine, to be executed and applied on each of the image sequences that are continuously coming in. If the CEP-Esper engine finds image sequences that match the predefined queries and patterns, events are identified in image sequences which will be consumed by the lower layers of our design, i.e., a graphical user interface or the lower layers of consumers.

The above approach allows us to process image sequences (events) as they arrive and easily scale them horizontally. In this way, the processing of complex events occurs in parallel to the historical data storage and the consumer or visualization layer. This activates all the intelligent processing logic implemented in the engine.

The CEP-Esper engine contains the logic of intelligent patterns that allows the analysis of detection, tracking, and recognition of people in the image sequences captured by video surveillance cameras. The rule base has the knowledge to correlate and recognize people detected in

different image sequences in different spaces and times.

For the case study, cases of abnormal behavior, such as sudden changes of rhythm, must be detected in people sharing public or private space. The configuration of patterns for the detection, tracking, and behavior of people, and the configuration of abnormal behavior parameters are considered. These algorithms implement the patterns in the CEP-Esper engine. Once the engine has analyzed the events or image sequences, it determines if there is abnormal behavior through the analysis of the stated patterns. In that case, it publishes the image sequence (events) in the publication services layer. These are taken from the subscriptions and notifications services layer by consumers (Fig. 6, step 3), allowing us to dispatch image events according to which consumers are interested.

The intelligence of the system lies in the design of the information capture and registration services, and the implementation of rules in the CEP. In this way, CEP infers new, more complex events with greater semantic meaning from the sum of simple events, which allows defining meaningful patterns of behavior detected in the image sequences that are registered in the system. This enables the pattern to learn and, from this learning, to analyze the series of events and derive conclusions from them. The CEP-Esper engine also stores the image sequences in the knowledge database, as records of processed events, and as knowledge archives and historical data queries.

4.1.1.3. Event consumer module. The consumer has the logic for handling the event, the security and response scheme under the online processing of an event treated as of interest to the system. These components react to events received from the Esper EDA server, where the ESB is instantiated. The event consumer allows receiving or subscribing to events (event subscriber) that are of interest according to the defined logic and for which the consumer has responsibilities to assume.

After receiving the event, the consumer fulfills the tasks associated with the event that are related to the visualization of the data and its correct analysis. An update of the data processed from the Esper EDA server is necessary to understand the values and to make decisions. These include notifying the relevant authorities of the occurrence of events of interest.

The visualization and management of the information, through the event consumer module for the video surveillance system, can be implemented with web services, which involves the development of a browser page that exposes a user interface. This has all the logic of user interaction with the system.

In terms of its functionalities and navigation considerations, the system must provide searching and querying of image sequences and information by various criteria. For example, using the registration date of an image to the system or the name/description of that image. It should also be possible to view only live video streams, i.e., images that are being generated by the video surveillance cameras and entered into the system for online processing. Another functionality would be to view archived image sequences in the registry database, which is implemented by the CEP-Manager web service, or in the knowledge database, which is implemented by the CEP-Esper engine.

4.2. Use case – Microservice-oriented architecture for sensor management in intelligent and cloud-connected environments

This approach leverages microservice-based architectures with publish-and-subscribe messaging-oriented protocols, which implement a Messages Queuing Telemetry Transport (MQTT) broker. The proposed IA-CPS intelligent architecture uses a cloud platform to expose services in the IoT environment. It could be used by utility companies and customers to monitor the energy and drinking water consumption records in a domestic environment using pre-installed sensing devices. The monitoring of the consumption records can be performed on a web server accessed by authorized users from a user-oriented web interface application. Moreover, this architecture supports other sensor types covering different application areas.

4.2.1. System architecture

The IA-CPS reference architecture complies with the paradigm and fundamental principles of a MSA and EDA, considered as state-of-the-art technologies to manage and process information in IoT and cloud environments. The proposed framework can process the information captured from a network of sensors. In our case, these are sensor nodes connected to indoor meters in water supply connection of homes, shopping malls, or industries, which deliver water and energy consumption to the users.

The intelligence lies in the mapping of variables from various alphanumeric data delivered by sensors, and from the addition of simple events to form a complex event. This allows us to implement relevant behavioral patterns detected by the sensor devices installed in water and energy meters according to customer's requirements. The specification of the proposed architecture is oriented to the learning of the patterns in the CEP, so that from this learning it analyses the series of events and derives conclusions from them. All this allows the system to react online and improve decision making.

The microservices architecture embedded in IA-CPS extracts information and data captured by sensors that measure customer's water and energy consumption. The smart analysis is performed using a complex event-processing module, which generates multivariate analysis patterns of different water and energy consumptions: general consumption, peak hours, upstream and downstream consumption, cross-referencing consumption variables by stratum, by zone, by contracted service costs, service quality, consumption in cubic volumes of water and kilowatts of energy, etc.

In the following sections, we will detail the modules that compose the architecture and describe the pipeline.

4.2.1.1. Event producer module. The Message Queuing Telemetry Transport (MQTT) client system corresponds to the first module of the IA-CPS framework and is the first layer of the architecture of a CPS system interacting in the IoT. This corresponds to the perception or production layer. It allows obtaining data from the physical or virtual world using sensors managed by microservices.

4.2.1.2. Communication protocol. Once the sensors that will deliver information to the system are known, it is necessary to know the communications protocol that will send the information within the proposed architecture, that is, to the data processing module which is in the node or server layer where the database persistence takes place. This transport protocol is MQTT, which is an ISO standard (ISO/IEC PRF 20,922) based on the publish-subscribe paradigm for message passing.

To implement the protocol, we will need a node or central layer that runs a message broker. Next, we will describe the MQTT Broker Node.

4.2.1.3. Communication and transport module. One of the main features of MQTT communications is the broker, which is the system that will receive the messages sent by the clients in a system with a publication and subscription pattern. Among several MQTT brokers, we have chosen Mosquitto as it is open-source and multiplatform. The MQTT broker is a translation and data storage server. The role of the broker is to be an intermediary node whose function is to receive all messages from event-producing clients (see client-client node) and then redirect them to the relevant destination consumers or clients. In our case, for a broker, a client is an event producer or sensor node, but it can also be a consumer. This depends on the message flow direction, with which the broker, designed to receive and send messages containing the consumption data captured by the sensor, interacts. The sensor nodes of our system are in the IoT environment.

Since the IA-CPS reference architecture is based on microservices,

the broker is the API gateway layer, which is the interface in charge of publicly exposing the microservices. An API gateway is the piece in charge of unifying the publication of APIs to be consumed by other applications or by developers. It is an intermediary system that provides a REST or WebSocket API interface to act as a router from a single-entry point, the API gateway, to a group of microservices and/or third-party APIs. In our architecture, the API gateway distributes the information provided by a microservice, which can be information generated by water and energy sensors, or information from the intelligent analysis of the CEP complex event processor and the management by the end-users of the system.

4.2.1.4. Event processing module. An MQTT client is hosted on this server. This client is subscribed to the topics in which the data of measurements of interest made by the sensors are sent from the event producer MQTT client node. That is, to transfer the data collected from the sensors (MQTT client node) located in the MQTT broker node, it is necessary to implement an MQTT client in the MQTT server node (where the CEP engine is instantiated) to expose the data to the end clients. It also has the logic to store the information captured from the sensors in a database defined for the architecture.

Likewise, the CEP is instantiated in this module, which has defined the rules and intelligent action patterns, according to the variation or not of the event flows captured from the sensors. The microservices architecture embedded in IA-CPS extracts the information and data captured by sensors that measure the water and energy consumption of customers in a domestic environment, and the intelligent analysis is performed by the complex event processing module.

Also, this server hosts the database and the final client files that form the graphical interface web application, accessible to the users of the system. The objective of this node is to store and query the data published by the sensors and consumed by the broker for further user queries. End-users can access the web application through computers or smartphones.

4.2.1.5. Event consumer module. In Fig. 7, step 4, users have access to information through an interface where they can register and log in, with a navigation menu for the required services of each sensor. It enables users to interact with the sensor microservice of their choice (water or energy service). Its most relevant tasks are to analyze the sensor data and to manage the registry of sensors and devices and models that can be executed by creating a catalog of elements and programs of the model. In general, service information can be obtained from the users and sensors.

Data and information streams can be searched and queried by the registration date of an event, the name, or the description of the microservice of energy and water consumption.

Users will be able to capture information on consumption during peak hours, rise and fall of consumption, cross-referencing of consumption variables by stratum, by zone, by contracted service costs, quality of service, consumption in cubic volumes of water and kilowatts of energy, hours of service failure due to damage, maintenance, and discount equivalents, etc. The observation of captured consumption data can be done by interacting with the cloud web server discussed in the previous section. They can also access the system from smart mobile devices.

Also, this architecture allows connecting other types of sensors and extract the required information to cover other application areas such as telemedicine (blood pressure sensors, accelerometer, gyroscope, sugar level, or panic button), fire detection (temperature sensors, smoke level, or luminosity), etc.

4.2.1.6. Information processing flow. The information processing flow is depicted in Fig. 7. It shows a general overview of the proposed architecture and how system components interact between them. This interaction is done through messages that carry the input information captured in the sensor nodes. A topic (messaging protocol and language of microservices) corresponds to a sensor and a publish is a message from a given sensor, containing the data it has previously captured. When a client subscribes to a message, it is because it consumes the

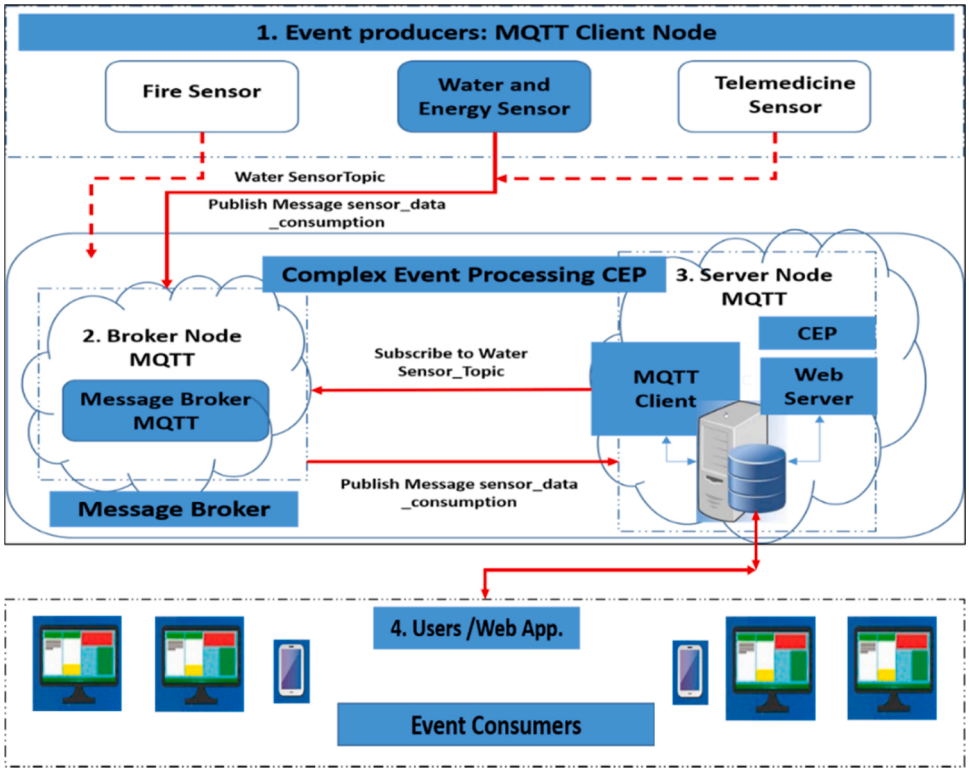

Fig. 7. General architecture of the proposed system.

published microservice delivering the information in that message and storing it in a database of the MQTT server node. This is how users, accessing the MQTT server node via a web application, see the information.

As defined above, the message broker is implemented using the open-source Mosquitto. It receives the publication and subscription requests from the components described in the proposed general architecture. These components can be concrete microservices or functionalities for web and mobile applications, among others. Mosquitto also performs the role of API gateway, i.e., interface that publicly exposes the microservices. It is responsible for distributing the information provided by a given microservice, such as the information generated by the water and energy sensors, or information obtained from the analysis and management of the end-users.

The information flow and data generated by sensors are addressed in all the nodes through a request to which clients subscribe to consume the exposed information. Next, we describe the information flow that activates the system of the proposed architecture (see Fig. 7).

- The event producer – MQTT client node at step 1 of the proposed architecture, captures data from the sensor and publishes a message with consumption statistics. Moreover, it sends a topic identifying the sensor type in the IoT. This is sent to the MQTT broker node.
- MQTT broker node at step 2 in the proposed architecture publishes the sent message to the event processing module – MQTT server node. As described above, the MQTT broker has the goal to publish and subscribe messages to and from clients who may need them (publish-subscribe).
- Event processing – MQTT server node at step 3 in the proposed architecture has implemented an MQTT client of the same type as the client described at step 1. This is implemented to be able to subscribe or consume the microservices exposed from the MQTT client and to be able to give a response to the MQTT broker in step 2. Moreover, it also processes the information sent from the event producer – MQTT client node at step 1.

In the MQTT event processing server in step 3, the CEP is instantiated. This module manages the events sending logic, exception, update, and detection. It also verifies errors that may occur in the parameters that unify an event. When we receive an event message at the bus level, the publishing services intercept and distribute it among the various event channels, making it available to consumers or the subscription service components.

The event processor handles the parameterization and combination of multiple event producers, i.e., multiple water and energy consumption sensors connected to the system. We emphasize that pattern management is a critical aspect for processing the data streams sent from the sensors. Likewise, in this module the final client applications are instantiated. A database has been implemented in the server to ensure recording, storing, and updating sensor data which travels through all the components of the architecture in the message format. Also, in this processing module, a web server is implemented to expose the data to the users of the system.

5. Conclusions

Nowadays, the importance of the architecture lies in the fact that the result of this practice will guide the construction of a system (a CPS system in our case). In this context, the concepts of architecture and design are completely related. From this perspective, a generic intelligent architecture IA-CPS (Intelligent Architecture for CPS management) has been proposed as an adaptive, autonomous, and interoperable EDA event-driven architecture for capturing event-driven data streams emitted by different types of sensors installed in the IoT. IA-CPS leverages the properties and principles of SOA, MSA, and CEP, interacting with components of these models.

The CEP module is the core of the proposed architecture, which is based on the definition of patterns that support the intelligent analysis of events in information flows and data captured by networks of sensors. It also performs the online detection and processing of interesting or unusual events of the physical scenario where the CPS and the sensor devices act.

Given the importance that cyber-physical systems acquire in all application domains, from IoT environments to cloud and smart cities, event-oriented architectures and the techniques applied to manage them provide motivations for future works in two areas. First, the detection on complex events in sensor devices subjected to the analysis of online and real-time processing. Secondly, the design of intelligent software that allows the integration to the changing dynamics and the behavior of the environment variables that make up the physical scenario where the CPS interacts.

Authorship statement

All persons who meet authorship criteria are listed as authors, and all authors certify that they have participated sufficiently in the work to take public responsibility for the content, including participation in the concept, design, analysis, writing, or revision of the manuscript. Furthermore, each author certifies that this material or similar material has not been and will not be submitted to or published in any other publication before its appearance in the Journal of Computational Intelligence.

Declaration of Competing Interest

We declare no conflict of interest.

Acknowledgments

This work was funded by the Spanish Government PID2019-104818RB-I00 grant for the MoDeaAS project and TIN2017-89069-R for Tech4Diet project, and PID2020-120213RB-I00 grant, supported by Feder funds.

References

[1] E.A. Lee, The past, present and future of cyber-physical systems: a focus on models, Sensors 15 (2015) 4837–4869.
[2] J.A. Ringert, B. Rumpe, Andreas Wortmann, Architecture and Behavior Modeling of Cyber-Physical Systems with MontiArcAutomaton. Aachener Informatik-Berichte, Software Engineering, Band 20 (February) (2014) 2015, 27.
[3] E.A. Lee, Cyber physical systems: design challenges. Object oriented Real-time distributed computing (ISORC), in: 2008 11th IEEE International Symposium on, 5-7 May, Orlando, Florida, USA., 2008, pp. 363–369.
[4] C. Perera, C.H. Liu, S. Jayawardena, The emerging internet of things marketplace from an industrial perspective: a survey, EEE Transactions on Emerging Topics in Computing 3 (December) (2015) 585–598.
[5] M. Hamdaqa, L. Tahvildari, Cloud computing uncovered: a research landscape, Adv. Comput. 86 (2012) 41–85.
[6] Y. Sun, G. Yang, X.-S. Zhou, A survey on run-time supporting platforms for cyber physical systems, Front. Inf. Technol. Electron. Eng. 18 (2017) 1458–1478.
[7] L. Monostori, Cyber-physical production systems: roots, expectations and R&D challenges, Procedia Cirp 17 (2014) 9–13.
[8] National Science Foundation, Cyber-physical Systems Summit Report, Missouri, USA April 24-25, 2008, http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf.
[9] Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: MEdit4CEP: A Model-driven Solution for Real-time Decision Making in SOA 2.0. Knowledge-Based Systems.
[10] Service component architecture – unifying SOA and EDA, tech. rep., Fiorano Software Technologies, 2010.
[11] Ollesch, J. Doctoral Symposium: Adaptive Steering of Cyber-Physical Systems with Atomic Complex Event Processing Services. Proceeding DEBS' 16. June 20 - 24.
[12] J. Boubeta-Puig, G. Ortiz, I. Medina-Bulo, A model-driven approach for facilitating user-friendly design of complex event patterns, Expert Syst. Appl. 41 (2) (2014).
[13] T. Levendovszky, A. Dubey, W.R. Otte, et al., Distributed real-time managed systems: a model-driven distributed secure information architecture platform for managed embedded systems, IEEE Softw. 31 (2) (2014) 62–69.
[14] P.L. Martínez, C. Cuevas, J.M. Drake, RT-D&C: deployment specification of real-time component-based applications, Proc. 36th EUROMICRO Conf. on Software Engineering and Advanced Applications (2010) 147–155.

[15] A. Dubey, G. Karsai, N. Mahadevan, A component model for hard real-time systems: CCM with ARINC-653, Softw. Pract. Exper. 41 (12) (2011) 1517–1550.
[16] Bures, T., Gerostathopoulos, I., Hnetynka, P., et al. DEECO: an ensemble-based component system. Proc. 16th ACM Sigsoft Symp. on Component-Based Software Engineering, p.81-90.
[17] P.L. Martínez, L. Barros, J.M. Drake, Design of component-based real-time applications, J. Syst. Softw. 86 (2) (2013) 449–467.
[18] J. Huang, F. Bastani, I.L. Yen, et al., Extending service model to build an effective service composition framework for cyber-physical systems, Proc. IEEE Int. Conf. on Service-Oriented Computing and Applications (2009) 1–8.
[19] D. Martin, M. Paolucci, S. McIlraith, et al., Bringing semantics to web services: the OWL-S approach, in: J. Cardoso, A. Sheth (Eds.), Semantic Web Services and Web Process Composition, Springer-Verlag, Berlin Hei- delberg, 2005, pp. 26–42.
[20] J. Huang, F. Bastani, I.L. Yen, et al., Toward a smart cyber-physical space: a context-sensitive resource-explicit service model, in: Proc. 33rd Annual IEEE Int. Computer Software and Applications Conf., 125, 2009, pp. 122–127.
[21] P.A. Vicaire, E. Hoque, Z. Xie, et al., Bundle: a group-based programming abstraction for cyber-physical systems, IEEE Trans. Ind. Inform. 8 (2) (2012) 379–392.
[22] P.A. Vicaire, E. Hoque, Z. Xie, E. Hoque, J.A. Stankovic. Physicalnet: A Generic Framework for Managing and Programming Across Pervasive Computing Networks. RTAS' 10: Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium April, 2010, pp. 269–278.
[23] "Radically innovative mechatronics and advanced control systems (RIMACS)—Deliverable D1.2—Report on industrial requirements analysis for the next generation automation systems.".
[24] T. Cucinotta, A. Mancina, G.F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, F. Rusina, A real-time service-oriented architecture for industrial automation, IEEE Trans. Industr. Inform. 5 (3) (2009) 267–277.
[25] Java Agent Development Framework (JADE): "an Open Source platform for peer-to-peeragent based applications.
[26] P. Vrba, M. Radaković, M. Obitko, et al., Semantic technologies: latest advances in agent-based manufacturing control systems, Int. J. Prod. Res. 49 (5) (2011) 1483–1496.
[27] A. Giordano, G. Spezzano, A. Vinci, A Smart Platform for Large-scale Cyber-physical Systems, Springer International Publishing, Cham, Switzerland, 2016, pp. 115–134.
[28] J. Boubeta-Puig, G. Ortiz, I. Medina-Bulo, Approaching the Internet of Things through Integrating SOA and Complex Event Processing. IGI Global Book Series Advances in Web Technologies and Engineering (AWTE), IGI Global, 2014 (2014).
[29] D. Luckham, Event Processing for Business: Organizing the Real-Time Enterprise, Wiley, Nueva Jersey, (US, 2011.
[30] B. Sosinsky, Cloud Computing Bible, Wiley, Estados Unidos, 2011. He, M., Zheng, Z., Xue, G., Du, X. Event Driven RFID Based Exhaust Gas Detection Services Oriented System Research. In: 4th International Conference on Wireless Communications, Networking and Mobile Computing. pp. 1–4 (2008).
[31] J. Boubeta-Puig, J. Cubo, A. Nieto, G. Ortiz, E. Pimentel, Proposal for a Device Architectures Services With Event Processing, IGI Global, 2013.
[32] P. Radanliev, D. De Roure, M. Van Kleek, et al., Artificial intelligence in cyber physical systems, AI Soc. (2020), https://doi.org/10.1007/s00146-020-01049-0.

Henry Duque Gómez received his Phd degree with specialization in Cyberphysical Systems Architectures. His research areas of interest include: IoT, CPS, Software architectures, Visual Surveillance, Sensors.

Jose Garcia-Rodriguez received his Ph.D. degree, with specialization in Computer Vision and Neural Networks, from the University of Alicante (Spain). He is currently Full Professor at the Department of Computer Technology of the University of Alicante. His research areas of interest include: computer vision, computational intelligence, machine learning, pattern recognition, robotics, man-machine interfaces, ambient intelligence, computational chemistry, and parallel and multicore architectures.

Jorge Azorin-Lopez is an Associate Professor of Computer Science at the Department of Computer Technology of the University of Alicante. He received the Computer Science Engineer degree, in 2001 and Ph.D. in Computer Science from the University of Alicante, in 2007. His main topics of research are Computer vision: modeling vision systems to: perceive under adverse conditions, real scenes segmentation and labeling and automated visual inspection, and Digital home and Ambient Intelligence.

David Tomás, PHD. is a Lecturer in the Department of Software and Computing Systems at the University of Alicante, Spain. His research interests include information retrieval, knowledge representation, information extraction, log analysis, and machine learning approaches to structured and unstructured data. He is the author of more than seventy scientific publications in international conferences and journals. He has participated in over twenty public and private projects, including EU-funded QALL-ME [FP6 IST-033860], FIRST [FP7-287607] and SAM [FP7-611312] projects.

Andres Fuster-Guillo received the PhD degree in Computer Science at the University of Alicante (Spain) in 2003. He is currently Associate Professor at the Department of Computer Technology of the University of Alicante. His current areas of research interest include 3D computer vision, machine learning, deep learning, ambient intelligence, human activity analysis, and automated visual inspection.

Higinio Mora-Mora received the Ph.D. degree incomputer science from the University of Alicante in 2003. Since 2002, he has been a member of the Faculty of the Computer Technology and Computation Department, University of Alicante,where he is currently an Associate Professor anda Researcher. He also leads the Specialized Processors Architecture Laboratory and has coordi-nated several institutional research projects on newdisruptive technologies. He has been the main researcher and responsible of several R&D contracts with European companies to develop cloud computing technologies.