A Smart IoT-Aware System For Crisis Scenario Management

Marina Mongiello, Francesco Nocera, Angelo Parchitelli, Luigi Patrono, Piercosimo Rametta, Luca Riccardi, and Ilaria Sergi

***Abstract*—In most dangerous events, involving many people in large buildings, rescue workers need to intervene in a timely and targeted manner in order to help most number of people and secure the environments without wasting resources. This work presents an Internet of Things(IoT)-based framework, aiming at monitoring environmental parameters in order to alert rescuers when they exceed some alarm thresholds. A hardware infrastructure driven by a software layer adds flexibility and adaptability to the Complex Event Processing engine and to a rule engine-based reflective middleware that manages and analyzes raw data in conjunction with a knowledge base modeling the application domain.**

***Index Terms*—Internet of Things, IoT, Sensors, Fire Alert, Safety, Complex Event Processing, Reflective Middleware.**

I. Introduction

The aim of this work is to provide a framework to support rescue workers (firefighters, first aid workers, civil protection teams, etc.) in case of hazardous events. Particularly, we consider events in which they need to intervene in large environments with different access points and with a large number of people: meaningful examples are schools, hospitals, offices, senior citizens homes, cultural or entertainment places as museums or art galleries. In these cases, indeed, it is necessary to guide the rescuers at the points of the building in a timely manner, where there is a certainty that there are users to help, avoiding waste of resources in environments where there is nobody at the time of the disaster or where the damages are of low magnitude. The achievement of this goal will be possible through the creation of innovative services based on the use of emerging technologies enabling the Internet of Things (IoT) including sensors, mobile devices, apps, Bluetooth Low Energy, Cloud technologies, and the use of embedded devices, as initially investigated in [1]. Within the buildings to be monitored, the installation of low cost devices that can detect the environmental parameters of interest will be planned in each place (room, classroom, etc.) and, if certain thresholds are exceeded, alert the rescuers through a telephone call to one or more emergency numbers. Once alerted, rescuers will be able to readily read all the information about the building and have the specific environment in which they need to intervene. More in detail, the system that will be implemented is made up of the following conceptual components:

- *Ambient monitoring smart box*: A prototype box based on board or microcomputer (e.g. Arduino, Raspberry Pi, etc.) connected to a set of sensors useful for detecting specific environmental parameters (e.g. temperature, smoke, movement, etc.). This device will also have one or more communication interfaces (Wi-Fi, 3G/4G, Bluetooth) for communicating the externally detected data.
- *Back end Server:* It deals with the processing and storage of data detected by each device in order to obtain the magnitudes of interest. The business logic at the basis of this component can determine whether and which data collected should generate an alert to the rescuers. If so, it makes a call to fixed network numbers.
- *Adaptive Middleware and Business Intelligence Platform:* It takes care of storing and processing the data detected by each prototype device. The business logic that can determine whether and which data should generate an alert to the rescuers lies on the platform, it enables a call to fixed or mobile network numbers. In addition, the platform must be able to provide data for other purposes to other services (e.g. energy waste monitoring, etc.) The platform will have to store the floor plans of the buildings through a back end application administrator side of the service.
- *Front end Application:* A reactive Web Application that displays real-time situations inside the building, highlighting the employment status of the different environments involved. This application can be consulted by the rescuers once they receive the alarm so that they can quickly decide in which building environments to intervene.
- *Mobile applications:* The alert triggered by the rule engine inside the back-end server activates mobile applications to rescuers. It is the mobile interface of the front-end application.

Main goals of the framework aim to ensure the safety of public environments and rescuers, hence to protect goods and ensure continuity of service. Middleware behavior depends on a Complex Event Processing (CEP) engine [2] core component of the proposed framework to extract relevant

Manuscript received April 4, 2018; revised April 9, 2018. Date of publication: April 12, 2018.

Marina Mongiello, Francesco Nocera, Angelo Parchitelli and Luca Riccardi are with the Department of Electrical & Information Engineering(DEI), Polytechnic University of Bari, Bari, 70126 Italy, 30332 e-mail:{firstname.lastname}@poliba.it

Luigi Patrono, Piercosimo Rametta and Ilaria Sergi are with the Department of Innovation Engineering, University of Salento, Lecce, 73100 Italy, e-mail: {firstname.lastname}@unisalento.it

Digital Object Identifier (DOI): 10.24138/jcomss.v14i1.533

knowledge from the domain model. In recent years, CEP over event streams has become an increasingly important tool to extract relevant situational knowledge from real-time or "near real-time"distributed systems. The concept of CEP was introduced by David Luckham in his seminal work [2] as a "defined set of tools and techniques for analyzing and controlling the complex series of interrelated events that drive modern distributed Information Systems (IS)". Several CEP systems have been developed in the last few years, each one proposing a different processing model. Currently, the most popular are: StreamBase CEP[1], Esper[2], Apache Flink[3]. The framework validation has been mainly focused on an experimental proof-of-concept which allowed to verify all components (hardware/software) involved; in particular let us consider a Smart City environment and requirements of safety of people. The validation has been performed to promote safe communities and aimed at providing a good support for rescue operations in case of danger. Suppose to consider environmental parameters such as air quality, temperature and humidity, smoke as variables to be monitored. We have worked to build scenarios in a domestic controlled environment. The test phase, lasting 10 days, has allowed to analyze data from sensors in veritable installations in a way to offer ideas for the next evolutionary steps. On the other side the raw data product from all sensors are exposed with the Open Data paradigm. This big data is a transformation in the knowledge and governance of cities through the creation of a data deluge that seeks to provide much more sophisticated, wider-scale, finer-grained, real-time understanding and control of urbanity [3]. Big data are:

- high volume of raw data (e.g. terabyte of data);
- diverse in variety (structured or unstructured);
- temporally and spatially referenced;
- fine-grained in resolution, aiming to be as detailed as possible, and uniquely indexical in identification.

Many Smart City governments use real-time analytics to manage aspects of how a city functions and is regulated. The variety of data is large: from the flow of traffic to adjust traffic light sequences and speed limits and to automatically administer penalties for traffic violations [3] to environmental conditions might be collated from a sensor network distributed throughout the city, for measuring air pollution, water levels or seismic activity. Many local governments use management systems to log public engagement with their services and to monitor whether staff have dealt with any issues. Big data comes from sensors, devices, video/audio, networks, log files, transactional applications, web, and social media - much of it generated in real time and in a very large scale. In order to get more interaction with the maintenance staff working in a building, the system was able to process the raw data coming from the sensors spread in all rooms and getting on how to reduce the electricity consumption through a decision support algorithm that can help and advise on the right consumption curve. The remainder of the paper is structured as follows: in the next section we introduce background issues concerning monitoring system in IoT scenario and CEP architectures. Then we formalize a prototyping framework and software architecture. Experiments and results are discussed at the end. After a validation of the framework, conclusion and future work close the paper.

II. Related Work

A. Hardware solutions for Fire Monitoring Systems

In this section, a state of art of fire monitoring systems, including both commercial solutions and research works, is reported. Several commercial solutions provide fire monitoring systems or combined fire alarm and emergency communication systems[4]. However, the main weak point of such systems is their obtrusiveness due to wired connections among components; this aspect must be considered at the design time of the building, and it implies a quite high cost of setup and management for the system. Moreover, it hinders the adoption of the system in already existing buildings. Fire monitoring systems have been one of the application fields of Wireless Sensor Networks (WSN) in recent years, helping to overcome wiring-related issues by exploiting their wireless peculiarities [4]. In [5], a smart fire monitoring system is reported, it is composed of (i) a fire detection trigger module, which transmits the smoke and temperature parameters to (ii) a control module, which analyzes the information coming from the detector and transmits the fire information to (iii) a monitoring center module. Finally, it is responsible for monitoring the whole system and making decision about alarms and interventions. In [6], a centralized wireless fire control system Bosch Fire Monitoring System[5] , using WSN technology is developed for the scenario of a university campus. The system connects the five buildings in the campus with a central control room by using ZigBee technology[6] . In case of fire in any monitored building, the event is promptly notified to the control room and proper countermeasures can be taken. In [7], a novel solution for building fire monitoring system is proposed. It includes a ZigBee-WiFi gateway which transforms ZigBee network into WiFi network. In addition, the system identifies the place where the fire occurred, and it notifies this information to the handheld terminal of the building security personnel. In this way, they can intervene quickly and more precisely. In recent years, many research works have addressed the topic of fire monitoring systems by using an IoT-aware approach based on embedded and Cloud systems. The SCUBA project [8], among a wide range of functionalities, provides also a cloud-based service for buildings evacuation during a fire emergency and a proof of concept of the Rescue Worker Interface (RWI). It is a cloud-based service and mobile application that provides real-time monitoring and control capabilities in a unified view of the emergency. Many research studies are

[1]https://www.tibco.com/streaming-analytics [last access February 2018]
[2]http://www.espertech.com/esper/ [last access February 2018]
[3]http://flink.apache.org [last access February 2018]

[4]In-Building Mass Notification Systems, http://www.cooperindustries.com/content/dam/public/safety/notification/Resources/Brochures/Inbuilding%20MNS%20Brochure.pdf [last access February 2018]
[5]http://resource.boschsecurity.com/documents/FSM_Commercial_Brochure_enUS_1218466443.pdf [last access February 2018]
[6]ZigBee Alliance, http://www.zigbee.org/ [last access February 2018]

based on fast-prototyping boards like the Arduino Platform, to further lower system setup and deployment costs. In [9], a monitoring system for fire detection using Arduino microcontroller is presented. All the data taken from smoke sensor and camera connected to the Arduino are sent wirelessly to data monitoring system for alert visualization. In [10], a Home Based Fire Monitoring and Warning System using Arduino Uno R3 is presented, which can also quickly alert the building owner via GSM communication. The previous analysis highlights that only few projects provide a visual map of the building with a clear indication of the real-time status of each room. Furthermore, even less projects contemplate a direct connection with rescue workers and a support for the definition of the intervention strategy in case of emergency.

B. *Middleware for IoT*

In this paragraph we present the state of the art of existing approaches for Middleware for IoT and reflective middleware. Middleware for IoT is a very active research area. A huge amount of work is available in the literature about IoT middleware and the multiple challenges they face [11], [12], [13]. Interesting and structured surveys are in [14], [15], [16], [17]. In order to build distributed systems, software engineers have to know which middleware is available, which one is best suited to the problems at hand, and how middleware can be used in the architecture, design and implementation of distributed systems. In [16] the authors categorize existing middleware according to design approaches: event-based, service-oriented, agent-based, tuple-space, VM-based, database-oriented, and application-specific. The paper analyzes several existing middleware which are reviewed and summarized w.r.t. their supported functional, nonfunctional, and architectural requirements. According to the taxonomy described in this paper, in our work we consider some functional requirements - scalability, reliability, availability and adaptability - and architectural requirements - adaptiveness, context awareness and programming abstractions - for IoT middleware. A survey of reflective middleware for IoT is in [18]. The survey addresses a broad range of techniques, methods, models, functionalities, systems, applications, and middleware solutions related to context awareness and IoT. The paper analyzes, compares and consolidates past research work. One of the first approaches is in [19] that presents an architecture for reflective middleware based on a multi-model approach. Through a number of working examples, they demonstrate that the approach can support introspection, and fine- and coarse- grained adaptation of the resource management framework. More recent relevant approaches of reflective middleware are proposed in [13], [20], where an ontology-based IoT middleware is implemented, and in [21] that proposes SOAR (SOA with Reflection). The paper [22] presents a chemical reaction-inspired computational model using the concepts of graphs and reflection, which attempts to address the complexities associated with the visualization, modelling, interaction, analysis and abstraction of information in the IoT. The work [22] presents Internetware which consists of a set of autonomous software entities distributed over the Internet, together with a set of connectors to enable collaborations among these entities in various ways. To support on-demand collaboration, Internetware middleware employs an RSA and reflection mechanisms on its own application server. Qin et al. in the paper [23] extend the Multinetwork INformation Architecture (MINA), a reflective (self-observing and adapting via an embodied Observe-Analyze-Adapt loop) middleware with a layered IoT SDN controller.

III. Prototype Framework

The role of the environmental monitoring device is to collect raw data related to ambient parameters inside the building. As its main functionality, it periodically measures the value of some environmental parameters by using the on-board sensors. Then, the collected data are transmitted to the Back-end Server with the aim to evaluate if some alarms or interventions need to be notified and managed. The design of the proposed device has considered several aspects of the addressed scenario. In this context, in fact, the proper set of sensors to be used depends on the physical properties that must be sampled, such as the air temperature and humidity, the presence of flames, the presence of smoke and toxic gases, the presence of people. Similarly, the choice of the communication interfaces depends upon several characteristics of the building where the system is installed, in terms of typology (museum, school, hospital, etc.), topology (number of floors), surface, structural materials, etc. Even the ability to operate with batteries and in low-power mode must be considered, and both are desirable features, along with the scalability and the low cost. This last requirement is important in this context in order to foster the adoption of the proposed solution even in case of existing buildings, when only little investments are possible. For this reason, it is worth noting that the design of the environmental monitoring device is based on a "best-effort" approach. Considering the cost constraints, in fact, and the critical working conditions during the fire, it is not possible to guarantee that the monitoring device keeps working also during the fire event. The important thing is that it can track the ambient status (and communicate it to the Back-end Server) at least until a moment before it stops working due to fire. In this way, operators can have a snapshot of the situation at least when the event occurred and, if the node survived, also in real-time. The hardware has the structure described in Figure 1 and detailed in the following paragraphs. All previous considerations led to the definition of a modular hardware equipment for the environmental monitoring device, which can be adequately based on fast-prototyping boards, like Arduino[7]. If more computing and/or communication capacities were needed, microcomputers represent a valid alternative. Going into details, the Arduino Platform provides a wide set of prototyping boards, each one having well-defined characteristics and application fields. The Arduino MKR1000[8] directly includes a Wi-Fi module, a 32 bit low power 48 MHz ARM

[7]The Arduino Platform, https://www.arduino.cc/ [last access February 2018]

[8]Arduino MKR1000, https://www.arduino.cc/en/Main/ArduinoMKR1000[last access February 2018]

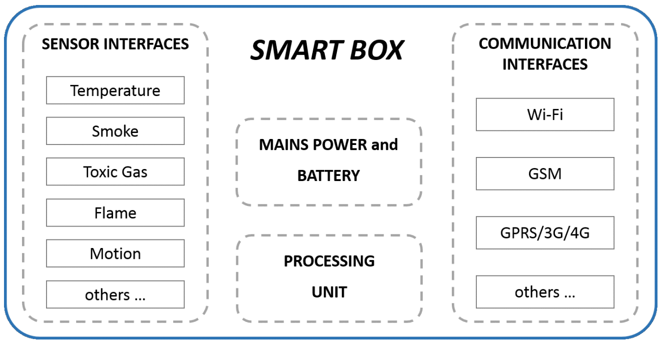


Fig. 1. Block diagram of the Smart box.

MCU, 32 KB SRAM, 8 Digital I/O pins, 7 analog inputs, 1 I2C port and it can also work with a Li-Po battery. In cases where more computing power and a greater number of analog/digital inputs are needed, the Arduino DUE[9] is more suitable, since it provides a 32bit low power ARM Cortex-M3 CPU running at 84 MHz, 54 digital I/O pins (12 of which can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), one USB OTG capable connection, and 2 DACs.

In simpler applications, instead, the Arduino UNO[10] is a valid alternative, in that it offers a lower number of analog/digital inputs and a lower CPU speed. This represents one of the most used Arduino boards. Unfortunately, unlike the Arduino MKR1000, on the Arduino UNO and DUE the Wi-Fi module is not included, so the external WiFi Shield[11] must be used. The 3G/4G/GPRS connectivity, instead, can be provided by several solutions, like the Arduino GSM Shield 2[12], 3G/GPRS shield[13], or 4G + GPS Shield for Arduino LE910[14], but with very different costs and functionalities. Finally, if the application's business logic requires a light-weight operating system running directly on the sensor node, the Arduino YUN[15] can be used for this purpose, joining the benefits of an Arduino and a Raspberry PI. Real microcomputers, such as the Raspberry PI 3, can also be used in this context, but they generally offer a limited connectivity, in terms of number of I/O analog/digital pins to connect with sensors and low performances when running with batteries. Finally, according to the requirements of the application scenario considered from time to time, the set of sensors used to sense environmental parameters may include:

- LM35AH or LM35DZ sensors, with the associated signal conditioning circuit, for measuring the air temperature in the range -55°C to 150°C,
- MQ2[16] sensor, directly interfaceable with Arduino boards, for smoke detection,
- MQ7 sensor, to detect other type of toxic gasses, like Carbon Monoxide,
- IR-based low-cost flame sensory[17],
- PIR sensor[18] or a more accurate MEMS Thermal sensor[19] to detect the presence of people in a room.

Figure 2 shows the schema of a possible implementation of the environmental monitoring device, based on the Arduino MKR1000, whose analog and digital input pins are used according to the typology of the attached sensors. The MQ2 smoke sensor and the MQ7 Carbon Monoxide sensor are equipped with a driving circuits with analog outputs, so they have been connected to the A1 and A2 analog inputs of the Arduino. The DHT11 temperature and humidity sensor has been connected to the D1 digital input pin, as well as the PIR sensor (D2 pin). Finally, the flame sensor has been connected to the A3 analog input, in order to adjust its sensitivity to infrared radiation. A signaling system has been also implemented to provide further information to the end user: three LEDs (green, yellow and red) are used to visualize the smoke level in the air and a buzzer is used to communicate a detected alarm in a conventional way. From a programming point of view, an Arduino program (called sketch) is composed of two main parts: the setup, executed at the boot of the program, and the loop, a set of instructions executed in an endless loop. During the setup phase, the following operations are performed:

1) Configuration of the I/O pins;
2) Connection to the Internet through the embedded WiFi module;
3) Check of the reachability of the server;
4) Configuration of the Real Time Clock (RTC);

During the loop phase, instead, all sensors are read in a periodic way (except for the PIR sensor that triggers its state when a presence is detected) and the corresponding data object containing these information is sent to the server. It basically contains:

1) ID of the device;
2) Value of the sensors;
3) Timestamp of the reading.

Figure 3 shows a prototypal implementation of the environmental monitoring device, with a plastic test chamber, to simulate a room in case of fire.

IV. Software Architecture

The software architecture, defined to capture and manage raw data from hardware components (i.e. smart box), is composed of two main layers: the back-end server and the

[9]Arduino DUE, https://store.arduino.cc/arduino-due [last access February 2018]

[10]Arduino UNO, https://store.arduino.cc/arduino-uno-rev3 [last access Feb. 2018]

[11]ArduinoWiFi Shield, https://www.arduino.cc/en/Main/ArduinoWiFiShield [last access Feb. 2018]

[12]Arduino GSM Shield 2, https://www.arduino.cc/en/Main/ArduinoGSMShield [last access Feb. 2018]

[13]Arduino 3G/GPRS shield, https://www.cookinghacks.com/documentation/tutorials/3g-gps-shield-arduino-raspberrypi-tutorial/ [last access Feb. 2018]

[14]4G + GPS Shield for Arduino LE910, https://www.cookinghacks.com/4g-gps-3g-gprs-gsm-gps-lte-wcdma-hspa-shield-for-arduino [last access Feb. 2018]

[15]Arduino YUN, https://www.arduino.cc/en/Main/ArduinoBoardYun [last access Feb. 2018]

[16]MQ Gas sensors, http://playground.arduino.cc/Main/MQGasSensors

[17]Low cost flame sensor, http://www.instructables.com/id/Arduino-Modules-Flame-Sensor/ [last access February 2018]

[18]PIR sensor, http://randomnerdtutorials.com/arduino-with-pir-motionsensor/ [last access Feb. 2018]

[19]MEMS Thermal sensor, https://www.omron.com/ecb/products/sensor/11/d6t.htm [last access Feb. 2018]

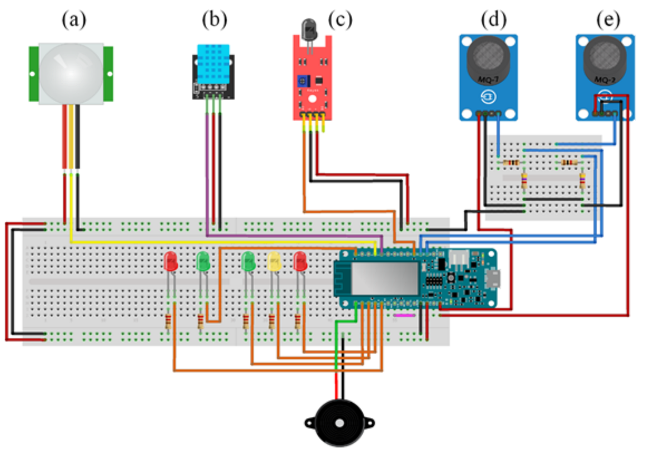


Fig. 2. Implementation schema of an environmental monitoring device based on Arduino MKR1000 board. It includes PIR sensor (a), DHT11 temperature and humidity sensor (b), IR flame sensor (c), MQ2 smoke sensor (d) and MQ7 gas sensor (e).

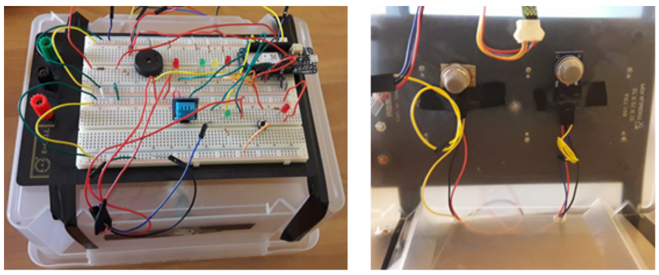


Fig. 3. Prototypal implementation of the environmental monitoring device, with annexed test chamber.

mobile device as shown in Figure 4. The two-layer software is modeled on a rule-based metamodel [13] leaning on an adaptive and reflective middleware interacting with a CEP engine [2]. The resulting software architecture is a flexible, adaptive, and scalable system. The first layer contains a rule-based system that matches the set of formal rules encoding the behavior of the system based on data observed from the devices. It implements the transformation of data and relationships accessed in an ontology that models the domain. We adopt a reflective paradigm [24] for modeling the IoT middleware, by implementing the Reflection design pattern. The main concept in Reflection pattern is the distinction between base level, meta level and the Meta Object Protocol (MOP) [25]. The base level contains the application logic while the meta level manages all the aspects that change independently from the base level. The meta level models runtime changing parts of the application, and creates new applications starting from the base level classes. The Meta Object Protocol enables adaptive changes in the metalevel and in the connections between the base level and the meta objects. A meta object is defined as an object that manipulates, creates, describes, or implements other objects (including itself) [25]. In this way, by using a programming language that supports reflection, we can design a completely configurable and extensible system able to adapt to different operating environments. This enables flexibility of the system since it is possible to run-time change the structure of the objects. A Message Broker component models the Publish/Subscribe mechanism for defining the relationships between messages containing physical sensors information, and actions. A Rule Based system implements the transformation of data and relationships accessed in the ontology that models the domain using a reasoning algorithm. The Adaptor component works as a driver and translates the received command in a real action. Different Adaptors can be placed for each action. The mobile device (components enclosed in the green box in Figure 4) includes five components. The core component is the Complex Event Processing (CEP) engine [25]. In recent years, CEP over event streams has become an increasingly important model to extract relevant situational knowledge from real-time or near real-time distributed systems. Sensor Adapters connect to sensors and translate information obtained from sensors into events format. All formatted events are sent to the Event Stream Management component, which dispatches event streams to other components such as the CEP engine or action handlers or sends the events to the server through publish methods. Pattern Deployment deploys/un-deploys the patterns, which are dispatched by the server to the CEP engine on the mobile device. In order to interact with a user, Action Handler executes actions like playing alarm audio, displaying alerts and/or recommendations on smartphone in case the pre-defined trigger events are detected. The two parts communicate through a publish/subscribe middleware composed by a Distributed Service Bus (DSB), a Google Cloud Messaging service and a Subscription Web Service.

V. Validation of the Framework

The proposed framework has been validated in a controlled environment by using a specific use case: we have used a real domestic scenario. In this simulation, we wanted to experience the passage of information between the various hardware and software components involved. With this, we can come to be able to simulate a real scenario, such as that of a public office as shown in the Figure 5. The main objective of the experimentation is a verification of the entire infrastructure supports the workload generated by monitoring units equipped with sensors capable of detecting environmental parameters such as, temperature, humidity and smoke. With this experiment we choice an Arduino boards with a Wi-Fi controller: temperature and humidity sensors were connected at Arduino boards and programmed to send data on server architecture compatible with Internet communication protocols. The choice of sensors has fallen on sensors with standards that guarantee a high degree of reliability and accuracy of the environmental measure. We use a DHT22 capable of measuring the temperature in a range from -40 to 80°C with $\pm 0.5°$ C accuracy. A validation was carried out in a distributed environment by entering the entire core of the middleware, including the layer of data persistence. For testing purposes, a Linux-based infrastructure was created within a server environment with the following features: 2,5 GHz Intel Core i5 with 16 GB of RAMs. In

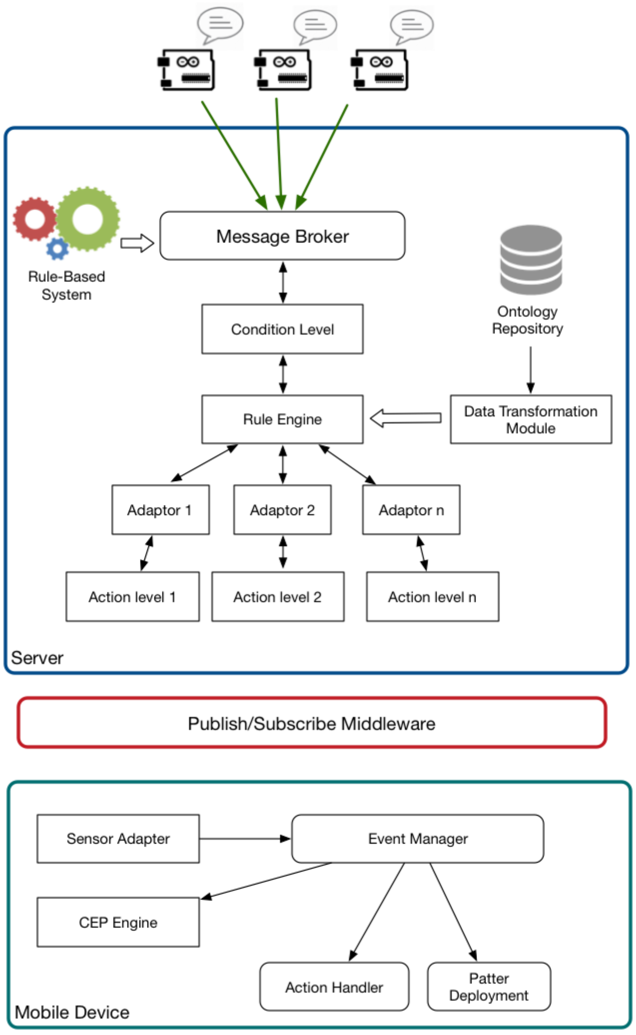


Fig. 4. Overall software architecture.

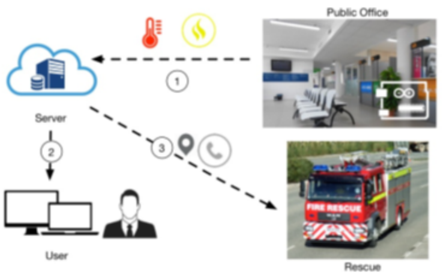


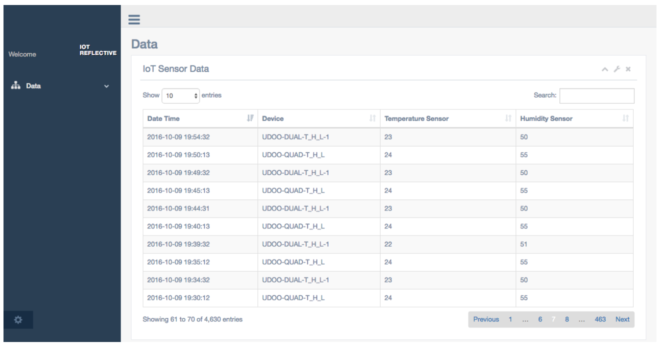
Fig. 5. Validation scenario.

TABLE I
THREE SENSORS MONITORING.

Date	Hour	Device	Temperature ($^{\circ}$C)	Humidity(%)
2017-09-28	14:23	UDOO-DUAL-T_H_L-1	23	51
2017-09-28	14:28	UDOO-DUAL-T_H_L-1	23	51
2017-09-28	14:33	UDOO-DUAL-T_H_L-1	23	51
2017-09-28	14:38	UDOO-DUAL-T_H_L-1	23	51
2017-09-28	14:43	UDOO-DUAL-T_H_L-1	23	50
2017-09-28	14:48	UDOO-DUAL-T_H_L-1	23	50
2017-09-28	14:53	UDOO-DUAL-T_H_L-1	23	50
2017-09-28	14:58	UDOO-DUAL-T_H_L-1	23	51
2017-09-28	15:03	UDOO-DUAL-T_H_L-1	24	50

addition, a virtual machine hosted in the Microsoft Azure Cloud (Ubuntu 16.04, one core, 2 GB of RAMs, 25 GB of HD space), has been devoted to monitoring and checking of the raw data received for a 10 days period. All communications between the entities occur in JSON format, like the following example:
*{ "device":" Arduino_F1_R1",
"sensor":"t_sensor",
"value":23,"timestamp":"1475353576"}*
As shown in the example, the data flow is generating by an Arduino board placed in the room 1 at first floor. The flow of data goes to the Reflective Middleware which is ready to analyze them. In Table 1 we show some data from three boards scattered in three different homes. The system will automatically perform actions according to the values received by the sensors of devices by matching the set of formal rules, considering the values of the monitored variables. For the mobile device, considering the limited computing resources on mobile devices we use the light weight Esper engine as CEP engine. Esper enables rapid development of applications that process large volumes of incoming messages or events, regardless of whether messages are historical or real-time in nature. Esper filters and analyzes events in various ways, and responds to conditions of interest. The CEP engine by matching the set of rules, activates a series of procedures, alerting the intervention of the rescue workers in the exact place on the map where the accident occurred.

For the back end server we choose the following technologies: (i) DeviceHive as IoT middleware; (ii) Redis as a Message Broker, Redis is a NoSQL DBMS and allows a system to translate a message from the messaging protocol of the sender to the recipients messaging protocol; through a decoupling between publishers and subscribers, Redis guarantees a greater scalability and (iii) an Observer, a component that observes rules extracted from the Rule based-systems. In our implementation, the rule based system is a configuration file in JSON format. The system will automatically perform actions according to the values received by the sensors of devices by matching the set of formal rules, considering the values of the monitored variables. The alert triggered by the rule engine inside the back end server activates mobile applications to rescuers. Through a web server the data is displayed in a table through an easily graphic interface (Figure 6). This user interface adopts a responsive paradigm allowing it to adapt to the multiplicity of devices currently on the market. This approach makes web pages render well on a variety of devices and window or screen sizes. A page designed with responsive design adapts the layout to the viewing environment by using fluid, proportion-based grids, flexible images and CSS3 media queries. In this application area this approach promotes the access to the control panel in every situation. Below this, the exposition of date following the Representational stateless transfer (REST) paradigm. A RESTful web services are a way of providing interoperability between systems on the web. REST-compliant Web services allow requesting systems to access and manipulate data representations of IoT resources using a uniform and predefined set of stateless operations. This



IoT Sensor Data

Show 10 entries Search:

Date Time	Device	Temperature Sensor	Humidity Sensor
2016-10-09 19:54:32	UDOO-DUAL-T_H_L-1	23	50
2016-10-09 19:50:13	UDOO-QUAD-T_H_L	24	55
2016-10-09 19:49:32	UDOO-DUAL-T_H_L-1	23	50
2016-10-09 19:45:13	UDOO-QUAD-T_H_L	24	55
2016-10-09 19:44:31	UDOO-DUAL-T_H_L-1	23	50
2016-10-09 19:40:13	UDOO-QUAD-T_H_L	24	55
2016-10-09 19:39:32	UDOO-DUAL-T_H_L-1	22	51
2016-10-09 19:35:12	UDOO-QUAD-T_H_L	24	55
2016-10-09 19:34:32	UDOO-DUAL-T_H_L-1	23	50
2016-10-09 19:30:12	UDOO-QUAD-T_H_L	24	55

Showing 61 to 70 of 4,630 entries

Previous 1 ... 6 7 8 ... 463 Next

Fig. 6. Web dashboard of the proposed system.

requests use HTTP methods: GET, POST, PUT, DELETE. By using a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running. In fact, this paradigm allows us to expose data in a variety of situation as in the case of Open Data. As proposed in [16] there are several non-functional requirements of IoT middleware. In this work, we will describe and analyze the requirements that middleware implements.

Scalability: An IoT middleware needs to be scalable to accommodate growth in the IoTs network and application or services. Scalability is the capability of a system to increase or decrease the available resources. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources are added, typically hardware resource. In this work, the scalability is achieved via load balancing that divide the data flow through multiple equals node. The node is a Virtual Machine(VM) configured by one or more CPU having a defined amount of RAM. This VM that contains all middleware component. In this case the scalability it is ensured by a load balancer that knows at all times the workload of each node and starting from this data decide to route the packet to a unloaded node. All nodes are equal and contain the same components.

Reliability: The measure of reliability is a probability that a system will produce correct outputs up to some given time t. This measure is enhanced by the system features that help to avoid, detect and repair hardware faults. In this situation the system not deliver the results but detects and, if possible, corrects the corruption. Applying this measure to an IoT middleware this should remain operational for the duration of a process, even in the presence of failures. For our cases the measure of reliability is 100 % without error for the 10 days of trial.

Availability: The measure of availability is connected by the times that the middleware must be available or appear available. If there is a system failure, the recovery must be small enough to achieve the desired availability. The availability and the reliability requirements is connected to ensure the highest fault tolerance require from this application. In our case, the availability measure is 100 % for the 10 days of trial. Next to this requisite there are few architectural requirements that are designed to support application developers. We exposed those that the middleware observes.

Interoperable: This middleware should work with heterogeneous devices, technologies or applications. This important feature allows middleware to work with heterogeneous devices, technologies or application without additional effort from the application or service developer. This term has many definitions we define this property how the ability of a collection of communicating entities to share specified information and operate on that information according to a shared operational semantics in order to achieve a specified purpose in a given context [26]. The main feature of interoperation is a relationship between systems or networks, syntactic, and semantic perspectives each of which must be catered for in an IoT context. This component must be able to exchange data and services. For example, our middleware is interoperable with any IoT board such as Arduino, Raspberry, UDOO, etc.

VI. Conclusion

There are currently available several middlewares, but almost always without documentation and poorly integrable. In this work, we propose a reflective extension of an IoT middleware, designed to be fully configurable and adaptable in different operating environments. The proposed framework allows us to perform automatically actions based on raw data received from the sensor network. In addition, the associated knowledge base allows you to define even rules inside the operating environment. Our implementation captures data from a network of sensors based on the Arduino platform (prototype development platform) by exploiting an embedded system as Raspberry sends all data into the Cloud. We are working to conduct extensive experiments considering also the presence of people, such as schools, hospitals, offices, senior citizens' homes, museums, etc. Additionally, we are committed to extend the set of actions to enable integration and interoperability with multiple and heterogeneous middleware.

References

[1] M. Mongiello, L. Patrono, T. Di Noia, F. Nocera, A. Parchitelli, I. Sergi, and P. Rametta, "A complex event processing based smart aid system for fire and danger management," in *Advances in Sensors and Interfaces (IWASI), 2017 7th IEEE International Workshop on*. IEEE, 2017, pp. 44–49.

[2] D. Luckham, *The power of events*. Addison-Wesley Reading, 2002, vol. 204.

[3] R. Kitchin, "The real-time city? big data and smart urbanism," *GeoJournal*, vol. 79, no. 1, pp. 1–14, 2014.

[4] D. Alessandrelli, L. Mainetti, L. Patrono, G. Pellerano, M. Petracca, and M. L. Stefanizzi, "Implementation and validation of an energy-efficient mac scheduler for wsns by a test bed approach," in *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*. IEEE, 2012, pp. 1–6.

[5] C. Zhang and J. Li, "Fire monitoring system design based on zigbee wireless network technology," in *World Automation Congress (WAC), 2012*. IEEE, 2012, pp. 1–4.

[6] C. Meera, P. S. Sairam, S. Sunny, R. Singh, and R. Singh, "Implementation of an incampus fire alarm system using zigbee," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*. IEEE, 2015, pp. 732–737.

[7] L. Yunhong and Q. Meini, "The design of building fire monitoring system based on zigbee-wifi networks," in *Measuring Technology and Mechatronics Automation (ICMTMA), 2016 Eighth International Conference on*. IEEE, 2016, pp. 733–735.

[8] H. M. Poy and B. Duffy, “A cloud-enabled building and fire emergency evacuation application,” *IEEE Cloud Computing*, vol. 1, no. 4, pp. 40–49, 2014.

[9] M. S. A. Azmil, N. Ya’Acob, K. N. Tahar, and S. S. Sarnin, “Wireless fire detection monitoring system for fire and rescue application,” in *Signal Processing & Its Applications (CSPA), 2015 IEEE 11th International Colloquium on*. IEEE, 2015, pp. 84–89.

[10] S. Suresh, S. Yuthika, and G. A. Vardhini, “Home based fire monitoring and warning system,” in *ICT in Business Industry & Government (ICTBIG), International Conference on*. IEEE, 2016, pp. 1–6.

[11] M. A. Chaqfeh and N. Mohamed, “Challenges in middleware solutions for the internet of things,” in *Collaboration Technologies and Systems (CTS), 2012 International Conference on*. IEEE, 2012, pp. 21–26.

[12] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “Role of middleware for internet of things: A study,” *International Journal of Computer Science and Engineering Survey*, vol. 2, no. 3, pp. 94–105, 2011.

[13] M. Mongiello, T. Di Noia, F. Nocera, E. Di Sciascio, and A. Parchitelli, “Context-aware design of reflective middleware in the internet of everything,” in *Federation of International Conferences on Software Technologies: Applications and Foundations*. Springer, 2016, pp. 423–435.

[14] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “A survey of middleware for internet of things,” in *Recent trends in wireless and mobile networks*. Springer, 2011, pp. 288–296.

[15] G. Fersi, “Middleware for internet of things: A study,” in *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*. IEEE, 2015, pp. 230–235.

[16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, “Middleware for internet of things: a survey,” *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[17] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M. A. Gerosa, and A. B. Hamida, “Service-oriented middleware for the future internet: state of the art and research directions,” *Journal of Internet Services and Applications*, vol. 2, no. 1, pp. 23–45, 2011.

[18] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[19] G. S. Blair, F. Costa, G. Coulson, F. Delpiano, H. Duran, B. Dumant, F. Horn, N. Parlavantzas, and J.-B. Stefani, “The design of a resource-aware reflective middleware architecture,” in *International Conference on Metalevel Architectures and Reflection*. Springer, 1999, pp. 115–134.

[20] T. Di Noia, M. Mongiello, F. Nocera, and E. Di Sciascio, “Ontology-based reflective iot middleware-enabled agriculture decision support system.” in *SWAT4LS*, 2016.

[21] G. Huang, X. Liu, and H. Mei, “Soar: towards dependable service-oriented architecture via reflective middleware,” *International Journal of Simulation and Process Modelling*, vol. 3, no. 1-2, pp. 55–65, 2007.

[22] A. Ikram, A. Anjum, R. Hill, N. Antonopoulos, L. Liu, and S. Sotiriadis, “Approaching the internet of things (iot): a modelling, analysis and abstraction framework,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 8, pp. 1966–1984, 2015.

[23] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, “A software defined networking architecture for the internet-of-things,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[24] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, “A system of patterns: Pattern-oriented software architecture,” 1996.

[25] F. Buschmann, K. Henney, and D. Schimdt, *Pattern-oriented Software Architecture: on patterns and pattern language*. John wiley & sons, 2007, vol. 5.

[26] D. Carney, D. Fisher, and P. R. Place, “Topics in interoperability: System-of-systems evolution,” 2005.

Marina Mongiello is an Assistant Professor at Politecnico di Bari, since 2002. Her recent research interests include: Software Engineering and Software Architecture; Formal methods and model checking, Mobile and distributed reasoning and applications, software architecture for self-adaptive and context-aware software. She is associate editor of the *IET Software Journal*, she has served in organizing and in the Program Committee of several International conferences and workshops

Francesco Nocera is a Ph.D. student with particular interests in Software Engineering and Artificial Intelligence fields. Main research topics are: Information Flow Processing (IFP) Systems, Knowledge Representation Systems and Applications for the Semantic Web, Self-adaptive Systems and software architecture design. He holds a master’s degree in Computer Engineering from Polytechnic University of Bari. He has taken part in the organization and in the Program Committees of conferences and workshops. He has served as a reviewer for different scientific journals.

Angelo Parchitelli is a Master student in Computer Science Engineering and he is an IEEE Student Member. He is currently a Research Assistant in the Information Systems Laboratory (SisInfLab), Polytechnic University of Bari. His research interest include Information Flow Processing (IFP) Systems, Self-adaptive Systems and software architectural design, Software Architecture for Internet of Things and microservices architecture.

Luigi Patrono received his MS in Computer Engineering from University of Lecce, Lecce, Italy, in 1999 and PhD in Innovative Materials and Technologies for Satellite Networks from ISUFI-University of Lecce, Lecce, Italy, in 2003. He is an Assistant Professor of Network Design at the University of Salento, Lecce, Italy. His research interests include RFID, the Internet of Things, cloud, wireless sensor networks, and embedded systems. He authored more than 100 scientific papers published in international journals and conferences. He has been Organizing Chair of some international symposia and workshops, technically co-sponsored by the IEEE Communication Society, focused on RFID technologies, and the Internet of Things.

Piercosimo Rametta received the Master’s degree in Computer Engineering with honors at the University of Salento, Lecce, Italy, in 2013. His thesis concerned the definition and implementation of a novel mash-up tool for Wireless Sensor Networks configuration. Since November 2013 he collaborates with IDA Lab IDentification Automation Laboratory at the Department of Innovation Engineering, University of Salento. His activity is focused on the definition and implementation of new mash-up tools for managing smart environments based on Wireless Sensor Networks and Internet of Things.

Luca Riccardi received the Master’s degree in Computer Science Engineering at the Polytechnic University of Bari, Bari, Italy, in 2017. He is currently a Ph.D. student in the Information Systems Laboratory (SisInfLab), Polytechnic University of Bari. His research interests include Formal Methods for Security in Internet of Things analysis of adaptable Service-Based Applications.

Ilaria Sergi received the Master’s degree in Automation Engineering from the University of Salento, Lecce, Italy, in 2012. Her thesis focused on the tracking of small laboratory animals, based on passive UHF RFID technology. Since 2012, she has been with the IDA Laboratory and Identification Automation Laboratory, Department of Innovation Engineering, University of Salento. Her activity is focused on the study of new indoor tracking systems and homecare solutions. She has authored several papers on international journals and conferences.