sensors

Article

An API for Wearable Environments Development and Its Application to mHealth Field [‡]

Fabio Sartori [*,†]

Department of Informatics, Systems and Communication, University of Milano-Bicocca, 20126 Milano, Italy

* Correspondence: fabio.sartori@unimib.it; Tel.: +39-02-6448-7910

† Current address: viale Sarca 336/14, 20126 Milan, Italy.

‡ Presented at the Mobihoc '19: The Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Catania, Italy, 2–5 July 2019.

Received: 25 August 2020; Accepted: 19 October 2020; Published: 22 October 2020

Abstract: Wearable technologies are transforming research in traditional paradigms of software and knowledge engineering. Among them, expert systems have the opportunity to deal with knowledge bases dynamically varying according to real-time data collected by position sensors, movement sensors, etc. However, it is necessary to design and implement opportune architectural solutions to avoid expert systems are responsible for data acquisition and representation. These solutions should be able to collect and store data according to expert systems desiderata, building a homogeneous framework where data reliability and interoperability among data acquisition, data representation and data use levels are guaranteed. To this aim, the wearable environment notion has been introduced to treat all those information sources as components of a larger platform; a middleware has been designed and implemented, namely WEAR-IT, which allows considering each sensor as a source of information that can be dynamically tied to an expert system application running on a smartphone. As an application example, the mHealth domain is considered.

Keywords: Android Wear; Bluetooth Low Energy; application configuration; wearable/mobile sensor information; smart environments; Internet of Things; mHealth

1. Introduction

Wearable technology has the ability to provide applications with functionalities typically not present in mobile and laptop devices. Examples of these functionalities are monitoring of physiological functions, evaluation of environmental variables, etc. A number of smart applications can obtain benefits from employing wearable devices; in particular, being important sources of data for reasoning, they could be used in the design and implementation of a new generation of expert systems.

As reported in [1], developing expert systems and, more generally, intelligent systems, is a complex task, where both deep understanding of the problem domain and the definition of heterogeneous knowledge representation techniques and tools are needed. These techniques are usually based on different paradigms and platforms.

In this framework, Internet of Things (IoT) has offered new and important possibilities for the design and implementation of expert systems; currently, every device can be connected to the Internet and provide valuable information for decision-making [2]. In the well-known tripartite vision of IoT [3], expert systems are located in the semantic-oriented area, given that they have been always conceived as applications for reasoning about data; expert systems should be able to distribute knowledge bases, in order to gather data from sparse resources maximizing the benefits for their reasoning strategies.

In a previous paper [4], the wearable expert systems (WES) notion is introduced as an innovative paradigm to develop knowledge based systems able to modify deliberations dynamically, according to the temporal evolution of data acquired from wearable devices. In that paper, the design and implementation of WES from the knowledge engineering perspective is faced, focusing on the conceptual model necessary to develop a complete rule-based system from a bottom-up analysis of the relations among the data collected in the environment. In other words, data were assumed to be perfectly reliable and always available, since the final goal was to test the capability of the WES framework to develop a correct reasoning process depending on a variable set of observations.

In this paper, the goal is to show how the wearable expert systems can be connected to real-world wearable devices in order to build effective mobile applications. To this aim, it is important to notice that the data feeding the system are characterized by several parameters defining their suitability to applications. For this reason, the WES notion has been extended in order to give a complete view of modern mobile applications, defining the wider concept of *Wearable Environment*.

According to [5], the service-oriented vision considers IoT as "the inter-networking paradigm enabled by technology stack which provides a seamless connectivity between physical and virtual objects to facilitate the development of intelligent services and applications with self-configuring capabilities." A recent paper [6] on *sustainability of wearable devices* points out that the development of middleware technology is crucial for the design and implementation of services that support users through wearable devices. The main reason is the need for heterogeneous systems to co-exist into a unique conceptual framework.

In this context, Wearable Environment Acquisition and Representation InfrasTructure (WEAR-IT) is a collection of APIs to connect data gathered by wearables and WESs. The goal of WEAR-IT is to build an overall view of a knowledge domain, where data are acquired from sets of wearables and applications can choose the best data sources on the basis of their needs. This API layer allows the dynamic connection of data sources to distributed applications. An App, accessing the WEAR-IT set of APIs, can be hosted on a large number of smartphones and interrogated about the device capabilities and the willingness of the owner to contribute certain data by periodically publishing sensor measurements.

2. Materials and Methods

2.1. Related Work

Much work has been carried out in recent years on the definition, specification and deployment of IoT infrastructures (see, for instance, the following surveys [7,8]). However, the general problem of realizing open, application-neutral platforms did not attract much attention, and it is far from having found a satisfactory solution.

For instance, some proposals aimed at achieving interoperability at the protocol level privilege, based on interworking gateways, overlooking the convenience of a common programming environment [9]. The many works stemming from the Web of Things model (see, for instance, [10–12]) provide solutions to the information semantics problem, but they assume implicitly rather complex, resource-rich, endpoint implementations. Moreover, they are not well suited to situations where the applications may be either centralized or (fully or partially) distributed to the endpoints; in our view, providing a programming model supporting any application distribution strategy would be beneficial. Finally, the issues of searching for the needed sensors and enrolling and configuring them on-the-fly are almost absent from the literature.

As reported in [13], wearable sensors have become very popular in many applications, being useful in providing reliable and accurate information on people's behavior. Most of these applications are related to the medical [14–16] and sport and training [17–19] domains.

Ubiquitous healthcare systems [20] exploit many hardware and software components, e.g., wireless sensor networks [21] and wireless body area networks [22]. Moreover, they typically use mobile devices and wireless cloud services, with the aim to reach pervasive availability.

Indeed, the mHealth domain is one of the most interesting and promising fields of application of wearable technologies. Despite this, they are still underutilized in the healthcare domain [23]. In [24], four main causes are pointed out; in particular, the *lack of interoperability and standardization* in detecting and storing acquired data is considered very significant. In this field, great efforts have been made to assess the accuracy of wearable sensors in classifying *activities of daily living* (ADL). The utility of accelerometers to evaluate the performance of ADL by elderly people monitored at home has been demonstrated [25]. A prototype system using in-shoe pressure and acceleration sensor has been proposed [26] to classify activities such as *sitting*, *standing* and *walking*. Patients affected by degenerative disorders, such as Parkinson's disease, are an important target: in [27] an accelerometer-based device designed for step counting in patients with Parkinson's disease is presented. Wearable sensors were used in [28] to control the rehabilitation of patients at home after abdominal surgery.

The important facilitating role of wearable technologies for the promotion of a healthy lifestyle is stressed in [29]. Obese individuals support and the implementation of clinical interventions based on promoting an active and healthy lifestyle through wearable technologies have been proposed [30,31]. Monitoring physiological data can improve the diagnosis and treatment of chronic conditions; e.g., cardiovascular diseases can benefit from continuous monitoring of physiological parameters [32].

This paper presents an attempt to overcome the limitations of the architecture of the applications above, which is based on wireless sensor networks communicating with fixed stations where data are collected and managed. The approach is very similar to the principles of *wireless sensor networks with mobile elements (WSNME)* proposed in [33]. In WSNME, special support nodes are introduced in addition to sensor nodes and information sinks: these nodes act as intermediate data collectors or mobile gateways.

As reported in [34], mobility in WSNs is useful for several reasons, e.g. increased reliability and reduced cost of data transmission, connectivity benefits and energy efficiency. The main difference between WSNME and traditional applications such as body sensor networks is the introduction of this intermediate layer, where data from sensors are collected, making them available to interested users. In our model, interested users are wearable expert systems and the intermediate level is implemented by smartphones communicating with an IoT platform; this approach leads to the following definition of a wearable environment.

2.2. Definition of Wearable Environment

A *wearable environment* is a triple

$$WE = \{A, MD, S\}$$

where

- $A = \{a_1, a_2, \ldots, a_n\}$ is a set of *applications*, possibly interconnected;
- $MD = \{wd_1, wd_2, \ldots, wd_m\} \cup \{sp_1, sp_2, \ldots, sp_k\}$ is a set of *wearable devices* and *smartphones*, possibly interconnected; and
- $S = \{s_1, s_2, \ldots, s_t\}$ is a set of *sensors*.

Figure 1 shows how a wearable environment can be characterized in terms of the wearable devices involved. Nowadays, wearable devices can cover the entire surface of a human body: in particular, *smart-wristbands and smart-watches* are very important from the mHealth perspective, since they can record physiological parameters such as heart rate, calories burned during the day, etc; they are usually equipped with movement tracking functions that keep trace of distance walked and the amount of time active.

In our definition of a wearable environment, the interconnection among all these devices is mediated by a *smartphone*, which plays two roles, being both a data-generating device and a data collection hub for other wearables. For a large-scale application, the wearable environment would include several smartphones, each acting as the hub for a group of wearables: in the following, we describe the simplest case of a single smartphone, although the extension to a larger environment is possible.

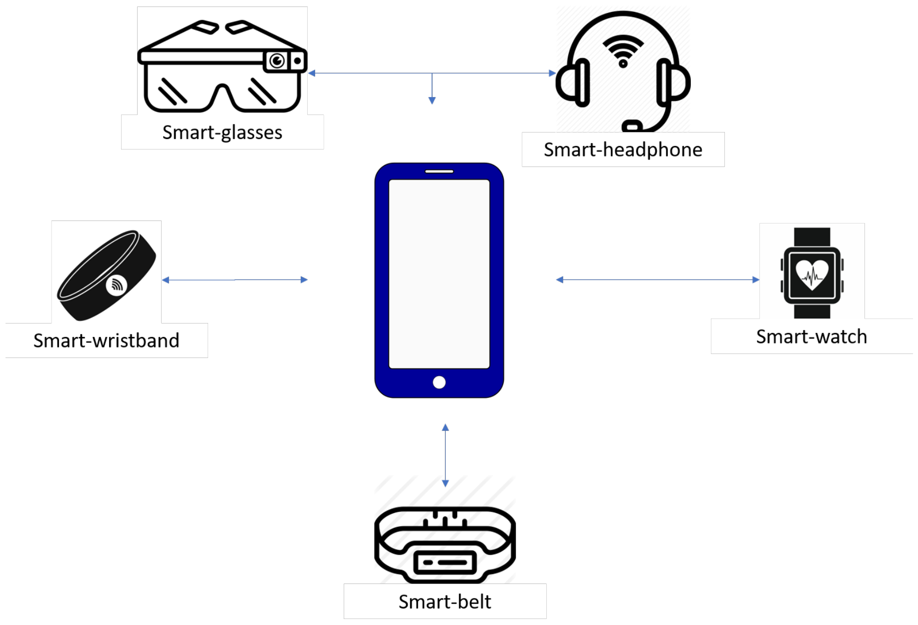


Figure 1. A sketch of the wearable devices involved in the definition of a wearable environment.

2.3. *WES Level: Applications*

Applications at reasoning level of a wearable environment are WES, typically characterized as rule-based systems. They implement a specific decision making process exploiting the variability of the related knowledge bases over the time [35,36]. This is due to the exploitation of information sources able to gather data from the environment around; the observed system and its reference environment evolve through a series of macroscopic states, each characterized by a specific set of relevant rules.

The transition from one state to the next causes that some events can become less important than others; therefore, the inferences activated by such events should decrease their salience accordingly. To take care of such dynamic evolution of WES knowledge bases, *bayesian networks* [37] have been integrated into the WES specification, to identify, for each time frame, the set of rules fitting better with the current macroscopic state, on the basis of a specific strategy [38].

2.4. *API Level: A Bridge between Data and Applications*

Applications at reasoning level could adopt different paradigms to implement their decision making processes. Thus, they should be able to interact with data in the most profitable way accordingly. This issue implies rethinking the relationship between applications and wearable devices providing data, enabling the former to select, browse and query the preferred device according to their goals.

Figure 2 depicts this change of perspective. The traditional two-tier architecture of a wearable expert system, based on *acquisition* of data from sensors on a wearable device and *reasoning* exploiting them, is extended to a three-tier architecture by adding an explicit *data representation* level. The main benefit is the possibility to make the reasoning process independent of the specific representation of necessary data: while in the first case (on the left in Figure 2) the WES must implement an opportune strategy for representing information gathered by sensors, in the second one it must only notify the representation level about the characteristics of needed data and their format, being notified when data are available accordingly.

To reach this objective, a specific framework has been developed, namely *Wearable Environment Acquisition and Representation InfrasTructure* (WEAR-IT). A complete API makes the reasoning level able to communicate with the data one, which is described in the following section.

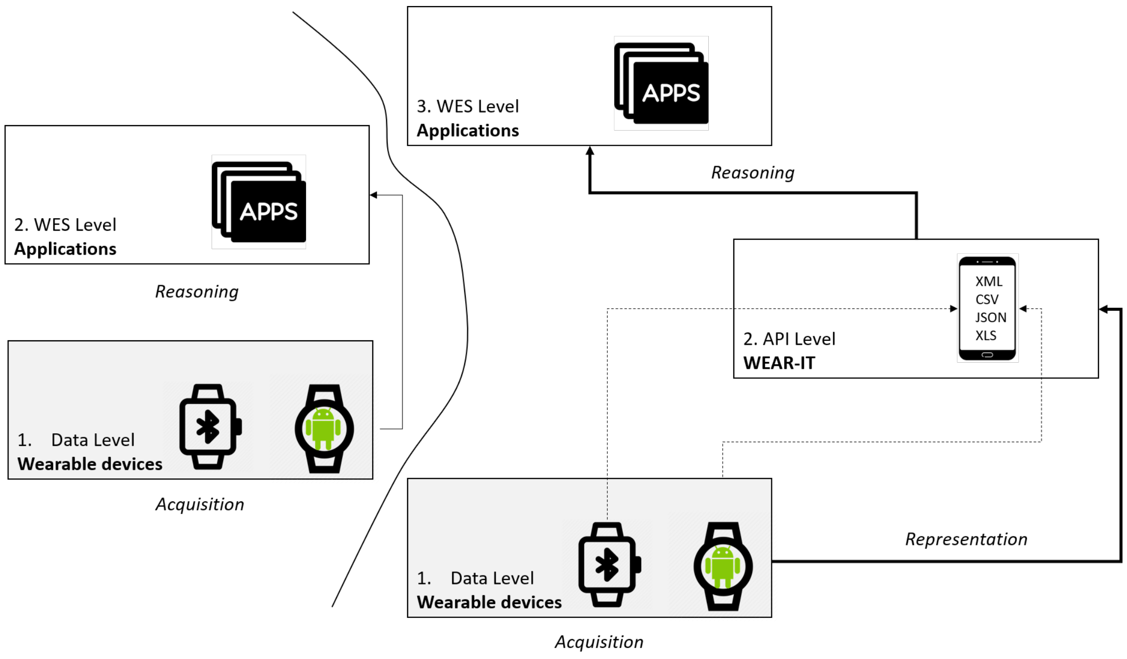


Figure 2. The API Level introduction: from a two-tier to a three-tier architecture.

3. The WEAR-IT API

3.1. Interpreting WEs as FSMs

Figure 3 presents a wearable environment as a Mealy [39] finite state machine (FSM):

$$WE = \{I, U, S, f, g\}$$

where the output state depends on the current state of the machine and the input received by it:

- $I = \{Scan, Connect, Sensors, Classify, Select, Play, Stop, Store\}$ is the *input alphabet*, made of primitives that can activate transitions of states.
- $U = \{\varnothing, Wearable\ Device, Sensor, True, False, Path\}$ is the *output alphabet*.
- $S = \{Reasoning, Representation, Acquisition\}$ is the finite set of possible states; in particular, the initial state $S_0 = Reasoning$;.
- $f : I \times S \longrightarrow S$: is the *transition function* $S(t+1) = f(I(t), S(t))$, mapping pairs of a state and an input symbol to the corresponding next state.
- $g : I \times S \longrightarrow U$: is the *output function* $U(t) = g(I(t), S(t))$, mapping pairs of a state and an input symbol to the corresponding output symbol.

This schema is useful to understand the role of the *representation* level in the wearable environment architecture, as well as the characterization of a wearable environment as a whole composed of heterogeneous entities.

When an application, i.e. a WES, is activated on the smartphone, the wearable environment enters the *reasoning* level. This is the initial state of the FSM: in fact, the information flow within a wearable environment is always started according to the needs of one or more applications. It is important to notice that the *reasoning* level is the final state too: the application could decide to execute its own decision making process and terminate without interacting with other levels, depending on its strategy.

Otherwise, it will move to the *representation* level to obtain necessary data to complete the body. This state is activated by a *scan* input, which specifies the application needs to know which wearable devices and sensors are available nearby to gather data. From now on, the wearable environment can maintain the representation level, move to *acquisition* level or come back to *reasoning* level. The *sensors* and *classify* inputs cause the level maintenance, being actions that provide lists of sensors around and their clustering according to the nature of detected data; the *connect*, *select*, *play* and *stop* inputs cause the transition to the *acquisition* state, being actions that allow establishing a connection with a specific wearable device, selecting one sensor on it and starting/stopping detection of data through it. Finally, coming back to the *reasoning* state is performed when the *store* input is given, since all the data necessary to the reasoning level to accomplish its actions are available.

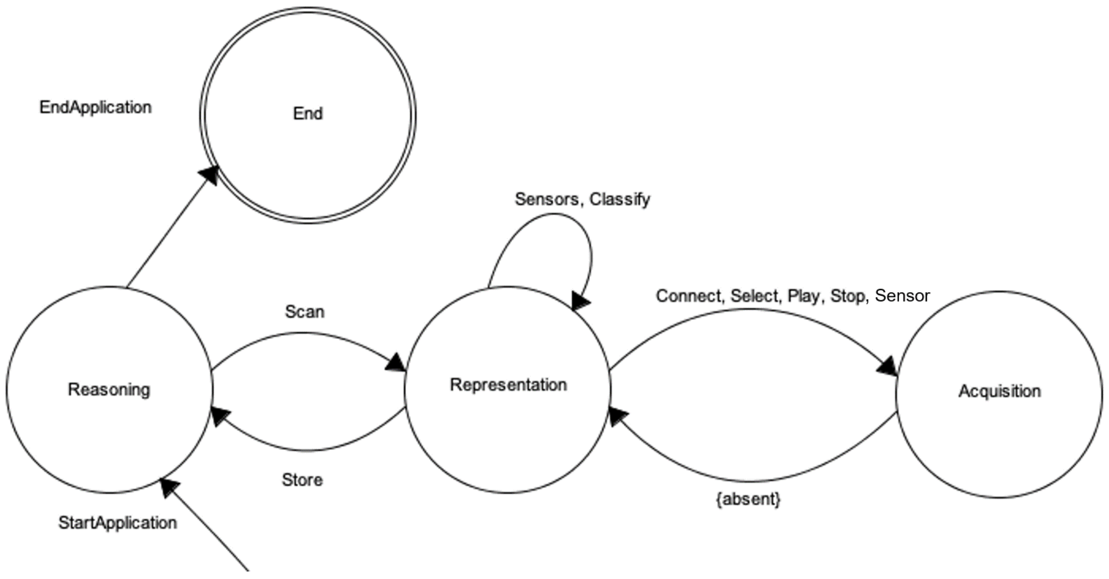


Figure 3. A Wearable Environment representation as a finite state machine.

Note that the transition from *acquisition* level to the *representation* one is automatic, at the end of the required primitive. Table 1 resumes the state transition table of a wearable environment.

Table 1. State transition table of a wearable environment.

Input / States	Scan	Connect	Sensors	Classify	Select	Play	Stop	Store	*{Absent}*
⟶Reasoning	Representation	∅	∅	∅	∅	∅	∅	∅	∅
Representation	∅	Acquisition	Representation	Representation	Acquisition	Acquisition	Acquisition	Reasoning	∅
Acquisition	∅	∅	∅	∅	∅	∅	∅	∅	Representation

Table 2 presents the output function g for each state. The *Scan* transition from *reasoning* to *representation* returns a list of *wearable devices*. During the persistence in the *representation* state many transitions occur. Towards the *acquisition* state, the *Connect* input returns the identification of a wearable devices among the previously scanned; the *Sensors* one allows obtaining a (possibly empty) *list of sensors* the previously connected wearable devices is equipped with; the $Classify$ transition arranges such sensors in a tree-like structure according to their nature; the *Select* input allows deciding which sensor to query; and the $Play/Stop$ actions start/terminate the acquisition of data from the selected sensor returning True/False values on the basis of sensor availability. Finally, the *Store* transition towards the *reasoning* state provides the *path* to the data storage. As introduced above, the transitions from *acquisition* state to the *representation* one are automatic at the end of the necessary operations. Since the goal of acquisition state is to finalize the interaction with selected wearable device or sensor, the result of such finalization is returned as a boolean value.

Table 2. Output function for each state of a wearable environment.

Reasoning State	
$\mathbf{g(S \times I)}$	**U**
g (Reasoning, Scan)	List <Wearable Device >
Representation State	
$\mathbf{g(S \times I)}$	**U**
g (Representation, Connect)	Wearable device
g (Representation, Sensors)	∅, List <Sensor >
g (Representation, Classify)	Tree <Sensor>
g (Representation, Select)	Sensor
g (Representation, Play)	True/False
g (Representation, Stop)	True/False
g (Representation, Store)	Path
Acquisition State	
$\mathbf{g(S \times I)}$	**U**
g (Acquisition, Wearable device)	True/False
g (Acquisition, Sensor)	True/False
g (Acquisition, True/False)	List<SensorValue>

3.2. Operational Semantics

The last step of the wearable environment API formalization is the definition of an operational semantics to detail the interactions among the states. This is necessary to clearly define which kinds of services can be invoked by a state. Each primitive is represented as a *method* (see Table 3) characterized by:

- **Name**: the name of the method, to identify the service in a non ambiguous way.
- **Parameters**: the (possibly empty) list of values necessary to perform the required service.
- **Description**: a brief explanation of the method body.
- **Return value**: the value obtained by the service invoking entity at the end of the method. This value must be coherent with Table 2.

Table 3. Definition of a primitive structure.

Return Value	Method Name, Parameters, Description
Value	name Primitive(parameter 1, parameter 2, ... , parameter n) Description

3.2.1. Scan ()

This function searches all the wearable devices wd within a wearable environment that can be associated to a smartphone sp (see Table 4).

Table 4. *The Scan()* definition.

Return Value	Method Name, Parameters, Description
List <WearableDevice>	Scan () Returns the collection of wearable devices available in a wearable environment.

3.2.2. Connect(sp, wd)

This function (see Table 5) returns *true* if the pairing operation between the smartphone sp and one of the wearable devices wd previously listed by $Scan()$ succeeds, *false* otherwise. Pairing is made thanks to $Bluetooth^{TM}$ technology.

Table 5. The *Connect(sp, wd)* definition.

Return Value	Method Name, Parameters, Description
True/False	Connect (Smartphone sp, WearableDevice wd) Returns *True* if pairing sp and wd succeed; *False* otherwise.

3.2.3. Sensors(sp) and Sensors(wd)

These methods (see Table 6) enable the invoking application to obtain a complete list of sensors available on the smartphone sp it is running on, as well as a previously connected wearable device wd.

Table 6. *The Sensors(sp) and Sensors(wd) definition.*

Return Value	Method Name, Parameters, Description
List <Sensor>	Sensors (Smartphone sp) Returns the list of sensors on the smartphone on the smartphone *sp*.
List <Sensor>	Sensors (WearableDevice wd) Returns the list of sensors on the wearable device *wd*.

3.2.4. Classify(sp) and Classify(wd)

These primitives (see Table 7) group all sensors on the smartphone sp where the invoking application runs and/or on the wearable device wd, already paired with sp, according to a hierarchical structure. This structure, depicted in Figure 4, complies with $Android^{TM}$ categorization of sensors in *position sensors*, *movement sensors* and *environment sensors*.

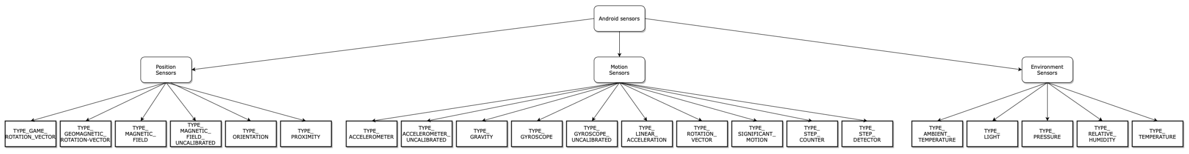


Figure 4. Hierarchical representation of sensors in $Android^{TM}$.

Table 7. The Classify(sp) and Classify(wd) definition.

Return Value	Method Name, Parameters, Description
Tree <Sensor>	Classify (*Smartphone sp*) Returns a tree-like cluster of sensors available on the smartphone *sp*.
Tree <Sensor>	Classify (*WearableDevice wd*) Returns a tree-like cluster of sensors available on the wearable device *wd*.

3.2.5. Sensor(sp) and Sensor(wd)

The collection of sensors $List < Sensor >$ returned by *Sensors* primitive depends on the device characteristics. It is extremely rare that a device is equipped with the totality of sensors available on the market. For example, *accelerometer*, *magnetometer* and *gyroscope* are common, while others can be derived from them (e.g., the rotation vector).

Since sensors could deprecate their performance due to accidents and/or aging, it is very important for an application to know if a specific sensor is working, even though it has been listed in the $Sensors(sp)$ and/or $Sensors(wd)$ result. $Sensor(sp, s)$ and $Sensor(wd, s)$, shown in Table 8, return *true* if the sensor s is really available for data gathering on a smartphone sp and/or a wearable device wd.

Table 8. *The Sensor(sp, s) and Sensor(wd, s) definition.*

Return Value	Method Name, Parameters, Description
True/False	Sensor (Smartphone sp, Sensor s) Returns True if the sensor s is available on the smartphone sp; False otherwise.
True/False	Sensor (WearableDevice wd, Sensor s) Returns True if the sensor s is available on the wearable device wd; False otherwise.

3.2.6. Select(sp, s) and Select(wd, s)

Once the wished sensor s has been identified and verified by the application, through the $Sensors(sp, s)/Sensors(wd, s)$ and $Sensor(sp, s)/Sensor(wd, s)$ primitives, respectively, the $Select()$ function in Table 9 allows activating s on sp or wd:

Table 9. The Select(sp, s) and Select(wd, s) definition.

Return Value	Method Name, Parameters, Description
Sensor	Select (Smartphone sp, Sensor s) Returns the ID code of the sensor s if it has been correctly activated on the smartphone sp; an empty string otherwise
Sensor	Select (WearableDevice wd, Sensor s) Returns the ID code of the sensor s if it has been correctly activated on the wearable device wd; an empty string otherwise.

The method returns information about the sensor to the invoking application. In particular, the *sensor identifier* (e.g., the UUID) is exploited by the reasoning level of the wearable environment for the next operations on it.

3.2.7. Play(sp) and play(wd)

The $Play()$ function is the most important part of the WEAR-IT API: given a sensor s, which has been previously identified, verified and selected on a smartphone sp and/or wearable device wd, it allows querying the sensor at different levels of granularity, in order to obtain detection of data from it, according to the reasoning level specification.

Table 10 shows the most extended interface of this function: the method can be properly overloaded according to the needs of the application. The parameters required are:

- **Frequency f**: This is the sampling frequency to collect data from the sensor s. The default setting is 5000 milliseconds.
- **Time period [t_0, t_n]**: This is the time interval during which the application wishes to collect data from sensor s at the given frequency f. Typically, t_0 is the instant when the primitive is invoked and $t_n > t_0$; if $t_0 = t_n$ only one value will be instantaneously returned, while $t_n < t_0$ will generate an error.

Possible variants are $Play(sp, s, n)/Play(wd, s, n)$, which allows obtaining n consecutive readings of sensor s value on a smartphone sp and/or a wearable device wd, sampled at the default frequency and $Play(sp, s)/Play(wd, s)$, which simply reads a value from sensor s when invoked.

Table 10. The Play(sp, s, t0, tn, f) and Play(wd, s, t0, tn, f) definition.

Return Value	Method Name, Parameters, Description
List <SensorValue>	Play (Smartphone sp, Sensor s, Time t_0, Time t_n, Frequency f) Returns a list of values gathered by sensor *s* on smartphone *sp* according to the parameters settings.
List <SensorValue>	Play (WearableDevice wd, Sensor s, Time t_0, Time t_n, Frequency f) Returns a list of values gathered by sensor *s* on wearable device *wd* according to the parameters settings.

3.2.8. Stop(sp, s) and Stop(wd, s)

This primitive interrupts the data collection from a sensor *s*, being it on a smartphone *sp* or a wearable device *wd*. Table 11 resumes it. This function can only be used if *Play*() has been previously invoked. The following cases can verify:

- If Play(*, s) has been invoked, with * = sp or * = wd, then *Stop(*, s)* is not mandatory.
- If Play(*, s, t_0, t_n, f) has been activated, with * = sp or * = wd and $t_0 < t_n$, then Stop(s) can be used to terminate the execution of Play(*, s, t_0, t_n, f) before the normal exit at the end of the $[t_0, t_n]$ time period.

Table 11. The Stop(sp, s) and Stop(wd, s) definition.

Return Value	Method Name, Parameters, Description
Void	Stop (Smartphone sp, Sensor s) Data detection on sensor *s* of smartphone *sp* is terminated.
Void	Stop (WearableDevice wd, Sensor s) Data detection on sensor *s* of wearable device *wd* is terminated.

3.2.9. Store()

Given a list of values, possibly composed of only one element, acquired by a previous *Play*() call, the *Store*() primitive allows archiving the data under different formats, such as JSON, XML, CSV, etc. As per the others, the Store() function aims to provide the applications at reasoning level of a wearable environment with a set of configurations to meet their needs. The default settings for parameters are the following ones:

- **File name**: a string composed by the *name of the sensor* and a *timestamp*, joined by "_".
- **Archiving directory**: the default directory of the application within the device where it runs.
- **Format**: the format of the file may depend on the type of data gathered by the sensor; the JSON (or XML) format is used to standardize it.
- **Access mode**: data are generally *appended* to the file, in order to preserve possible existing information.

Each variant of *Store*() returns the *path* of the file according to the operation result. In the case of errors, an error code is generated. In the case of success, the returned value could be an absolute path within the device (if default settings are active) or an *URI/URL* in the case of distributed resources. Table 12 summarizes the most extended interface of the *Store*() method.

Table 12. The Store() definition for smartphones and wearable devices.

Return Value	Method Name, Parameters, Description
Path	Store (Sensor s, Smartphone sp, Artifact a, List<SensorValue> sensorValues, Format f) Stores a set of data *sensorValues* gathered by sensor *s* within the *Artifact a*, according to the *format f*. In case of success, the *path* to the storage is returned; an error code is generated otherwise.
Path	Store (Sensor s, Wearable device wd, Artifact a, List<SensorValue> sensorValues, Format f) Stores a set of data*sensorValues* gathered by sensor *s* within the *Artifact a*, according to the *format f*. In case of success, the *path* to the storage is returned; an error code is generated otherwise.

3.3. An example

WEAR-IT has been developed under $Android^{TM}$ OS, since many wearable devices fully compatible with the $AndroidWear^{TM}$ interface are available at low cost on the market. Android OS provides a collection of primitives through which the sensors of a smartphone, or an Android Wear device, can be queried.

While developing WEAR-IT, other methods of interconnections among wearable devices have been considered, focusing on $BluetoothLowEnergy^{TM}$ technology. Other possibilities are the subject of future works (see Section 6).

Algorithm 1 shows a sketch of the usage of the WEAR-IT APIs. As shown in Figure 1, the smartphone sp is the only input necessary. Through the *scan* operation, the smartphone is able to determine the set of other available wearable devices, showing them to the user, which can be either a human being (if the graphical user interface is employed) or a software application. The user is then enabled to choose one specific wearable device *wd*, by means of the *connect* primitive; wd can then return the list of sensors it is equipped with and the user can select one of them exploiting the *select* operation. Finally, the *play* and *stop* functions are invoked between two distinct time instants t_1 and t_n, in order to detect raw data from the sensor for the desired period of time. As reported above, if $t_1 = t_n$, the $data_s$ variable contains a single value.

Algorithm 1 WEAR-IT API.

```
Require: $sp$
Ensure: $s, wd, data_s$
  $\{wd_i\} = scan(sp), i \in [1...k]$
  connect to a device $wd$ suitable for the application
  $connect(wd, sp)$
  identify the sensors in the wearable environment
  $\{s_j\} = classify(sensors(sp) \cup sensors(wd)), j \in [1...m]$
  select the needed sensor of type T
  $s = select(\{s_j\}, T)$
  $play(s)$
  measurement takes place
  $data_s = stop(s)$
  return $store(data_s, wd)$
```

An interesting extension of the algorithm above would be the remotization of WEAR-IT through the integration of IoT platforms. This would allow implementing dynamically the communication component of a smart city application. To provide a complete support to this kind of applications, it is possible to extend the API with the primitives needed to identify and select the endpoints suitable to the specific application needs. The primary extension is a $search(area, N)$ primitive which, in its simplest form, returns a list of up to N smartphones supporting WEAR-IT, presently located within the area boundaries. As an example, consider an application which requires a snapshot of the climate conditions (temperature, pressure and humidity) in a specific area. The implementation could follow the simple schema sketched in Algorithm 2.

Algorithm 2 Distributed data collection with WEAR-IT.

```
$\{sp_i\} = search(area, N)$
for all $i$ do
    $\{s_j\} = sensors(sp_i)$
    $st = select(\{s_j\}, TEMPERATURE)$
    if $st <> NULL$ then
        $t = play(st)$
    end if
    $sr = select(\{s_j\}, PRESSURE)$
    if $sr <> NULL$ then
        $p = play(sr)$
    end if
    $sh = select(\{s_j\}, HUMIDITY)$
    if $sh <> NULL$ then
        $h = play(sh)$
    end if
    if $t <> NULL$ then
        $store(sp_i.st, t)$
    end if
    if $p <> NULL$ then
        $store(sp_i.sr, p)$
    end if
    if $h <> NULL$ then
        $store(sp_i.sh, h)$
    end if
end for
```

Figure 5 shows how algorithm 1 works in practice, by means of an $Android^{TM}$ app developed to test the proposed API.

Here, it is supposed that

$$\text{WEAR-IT}_{wd} = \{sp \cup \{WearOS_i\} \cup \{BLE_j\}\}$$

with $j \in [1...n]$ and $j \in [1...m]$, being possible that $\{WearOS_i\} = \emptyset$ and $\{BLE_j\} = \emptyset$. At launch, the application allows scanning the wearable devices around the host smartphone and choose the device to connect with; this device belongs to one of the WEAR-IT$_{wd}$ subsets (Steps 1 and 2 in Figure 5). Then, the *classify* primitive is invoked (Step 3 in Figure 5). The wd sensors are clustered into four categories:

- *Sensor available* provides the list of all sensors mounted on the wearable device, alphabetically ordered. This function is useful to have a quick view about all the possible data an application at the reasoning level can exploit from the current device. The list of sensors can be exported to be used by applications.
- *Motion, environment and position sensors* group the sensors involved in the detection of the values necessary for the correct execution of an application, on the basis of its goals. The sensors (e.g., accelerometer) belonging to *motion* category can be exploited by applications interested in the

analysis of the user movement, e.g., recommender systems for training. The sensors (e.g., light) belonging to the *environment* category can be interesting for applications suggesting actions to take in response to changes in the wearable environment context, e.g., personalized entertainment. The sensors (e.g., orientation) belonging to *position* category can be queried by applications interested in the analysis of the user geographic position, e.g., systems for suggesting places to eat.

The *select* primitive usage is shown as Step 4 in Figure 5; accessing a category returned by *classify*, it is possible to watch the list of clustered sensors. Finally, raw data can be acquired and permanently stored by means of *play*, *stop* and *store* functions (Steps 5 and 6 in Figure 5). The final configuration of the wearable environment obtained at the end of the algorithm in also shown.

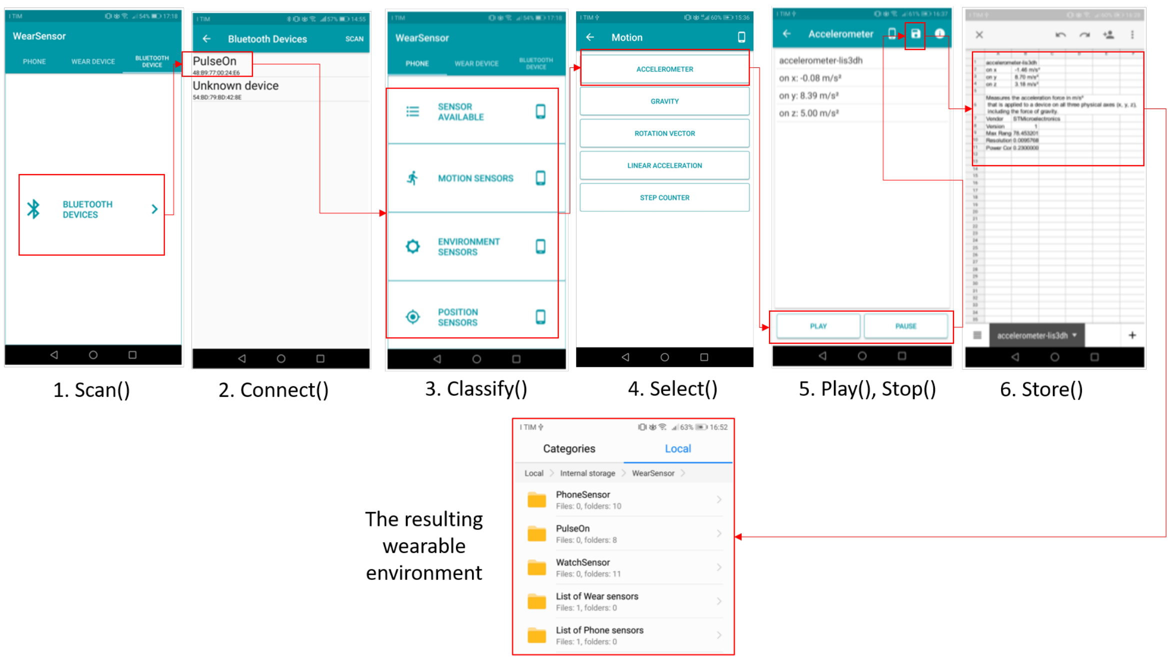


Figure 5. An application of Algorithm 1 for the generation of a wearable environment.

4. Case Study

Figure 6 shows a concrete implementation of the wearable environment conceptual model (see Figure 3 into a computational model, where an mHealth app, namely *MoveUp*, is considered at reasoning level. The diagram presents all the interactions among the different components of the wearable environment as well as interactions with external entities, such as the *storage*, allowed by the primitives previously described.

Looking at this diagram, the wearable environment is a platform where *software components*, i.e. wearable expert systems), *hardware components*, such as smartphone, wearable devices and their sensors, and *external resources*, like storage, are homogeneously managed. Thanks to the API provided by WEAR-IT, the MoveUp wearable expert system can access data from sensors on wearable devices in a transparent way.

The *Domain Dependence* label in Figure 6 illustrates the main technologies and operating systems currently available: wearable expert systems obtain data in a format depending on the desired sensor and wearable device. This format varies according to the domain: sensors available in Android Wear are different from sensors available on a BLE device. The WEAR-IT API makes the WES obtain data from the sensor without being aware of the sensor features. Solid arrows in Figure 6 show the current development of WEAR-IT API from the domain dependence perspective; dashed arrows specify under-development or possible future implementations.

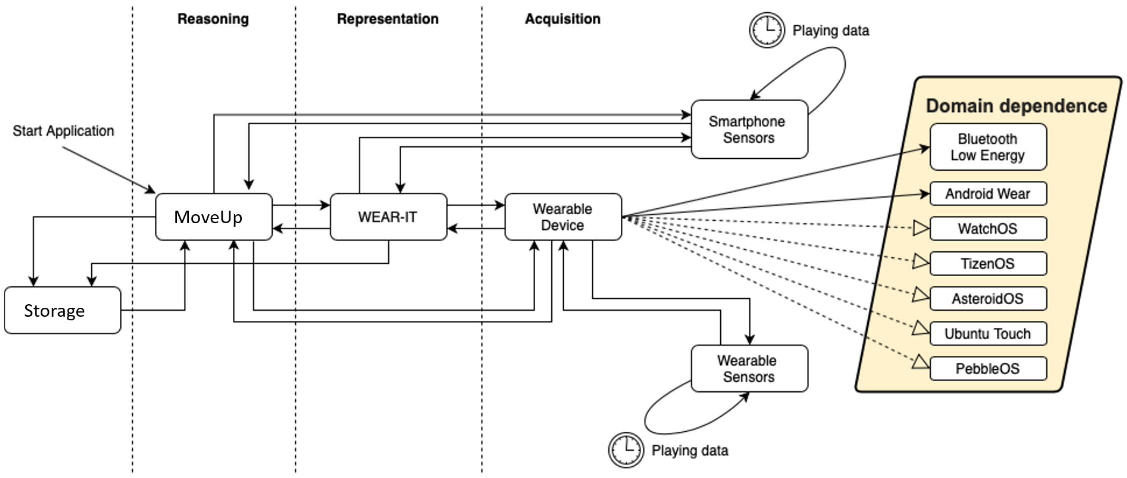


Figure 6. The wearable environment FSM model implemented into a case study.

Figure 7 presents a typical situation where wearable environments are exploited. Each user has some applications installed on his/her smartphone. Moreover, the smartphone is equipped with the WEAR-IT framework. The applications interact with WEAR-IT to acquire data from wearable devices distributed around the environment. Storage of data is on the cloud, to maximize the sharing of data among different users and/or applications, if involved in the same overall project. To this aim, WEAR-IT can communicate with opportune IoT platforms such as KAA (https://www.kaaproject.org/).

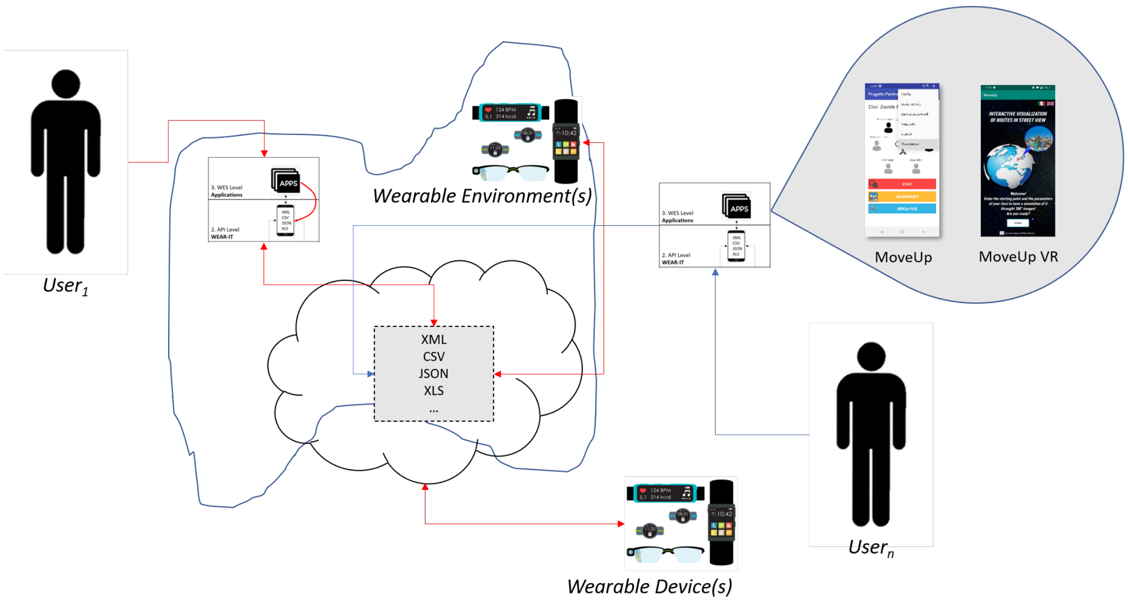


Figure 7. Information flow in the proposed scenario.

The *MoveUp* app supports personalized training programs in people at risk from the cardiovascular disease point of view. Physical activity (PA) is a very important factor to prevent chronic diseases; although guidelines have been produced about the amount of PA to accomplish, and they are continuously updated, much of the population continues to follow a sedentary lifestyle [40]. Traditional behavioral interventions have produced scarce results from the PA promotion point of view [41]; on the contrary, the availability of new, wearable technologies has allowed developing new applications to support people in modifying their behavior with respect to PA.

In the case study, the user exploits an application suggesting him/her how much physical activity to accomplish during the week; the evaluation is based both on quantitative, physical (e.g., "how much time do you train during a week?") and qualitative, psychological variables (e.g., "how well did you feel during the training session?"). From the physical point of view, the *metabolic equivalent of task* (MET) variable is used to estimate the amount and intensity of PA accomplished. The approach is based on guidelines provided by WHO [42] and it has been validated during an experiment involving 60 people, as presented in [43]. Here, the focus is not on the application itself, but on how the WEAR-IT introduction can be exploited to increase efficiency in the design and implementation of mHealth applications, as well as thinking at their development as an iterative process where new modules can be added that use the same wearable devices and data.

To develop the MoveUp reasoning strategy, the relationships between MET and *heart rate* (HR) has been used, given by the following formula [44]:

$$MET = 4 * Time^{MPA} + 8 * Time^{IPA} \tag{1}$$

where $Time^{MPA}$ and $Time^{IPA}$ are the periods of time the subject is involved in *moderate* and *intense* physical activity, respectively, measured in minutes.

Thus, the wearable environment in Figure 7 is configured to detect HR values from one connected wearable device starting at a given *time instant* and *frequency* and for a given *period of time*, as suggested by the application. Let us suppose that the user must accomplish 120 MET of physical activity: according to WHO guidelines (see https://www.who.int/dietphysicalactivity/factsheet_recommendations/en/), 120 METs correspond to 30 minutes of MPA or 15 min of IPA. A PA session is defined *moderate* if the registered HR values are in the range $[\frac{6*MHR}{10}, \frac{7*MHR}{10}]$, with $MHR = 220 - age$ is the subject's *maximum heart rate*, depending on his/her *age*. A PA session is defined *intense* if the registered HR values are in the range $(\frac{7*MHR}{10}, \frac{8*MHR}{10}]$.

This means that HR values should be sampled from a HR sensor over a 30-min time horizon. Algorithm 3 shows a sketch of Algorithm 1 related to the HR detection in MoveUp. The wearable device used in the example is a $PulseOn^{TM}$ (see https://pulseon.com/) smartwatch.

Figure 8 shows how WEAR-IT is involved in the development of the related wearable expert system. Starting from raw data acquisition from wearable devices (e.g., the heart rate), the MoveUp module provides suggestions about the amount of physical activity to accomplish week by week (further details about the conceptual model in [45]), thanks to WEAR-IT API use.

Algorithm 3 WEAR-IT API use in MoveUp.

```
Require: $sp$
Ensure: $s, wd, data_s$
  Comment: scan the wearable environment
  $\{PulseOn^{TM}, ...\} = scan(sp), i \in [1...k]$
  Comment: connect to $PulseOn^{TM}$ suitable for the application
  $connect(PulseOn^{TM}, sp)$
  Comment: identify the sensors in the $PulseOn^{TM}$ wearable device
  $\{s^j_{PulseOn^{TM}}\} = classify(sensors(PulseOn^{TM})), j \in [1...m]$
  Comment: select the needed sensor of type HEART_RATE
  $s = select(\{s^j_{PulseOn^{TM}}\}, HEART\_RATE)$
  Comment: measurement takes place for 30 minutes at default frequency; stop is not mandatory
  $play(s, t_0, t_0 + 30min)$
  Comment: data gathered by HR sensor are stored in XLS format
  return $store(s, PulseOn^{TM}, Path_{data_s}, data_s, XLS)$
```

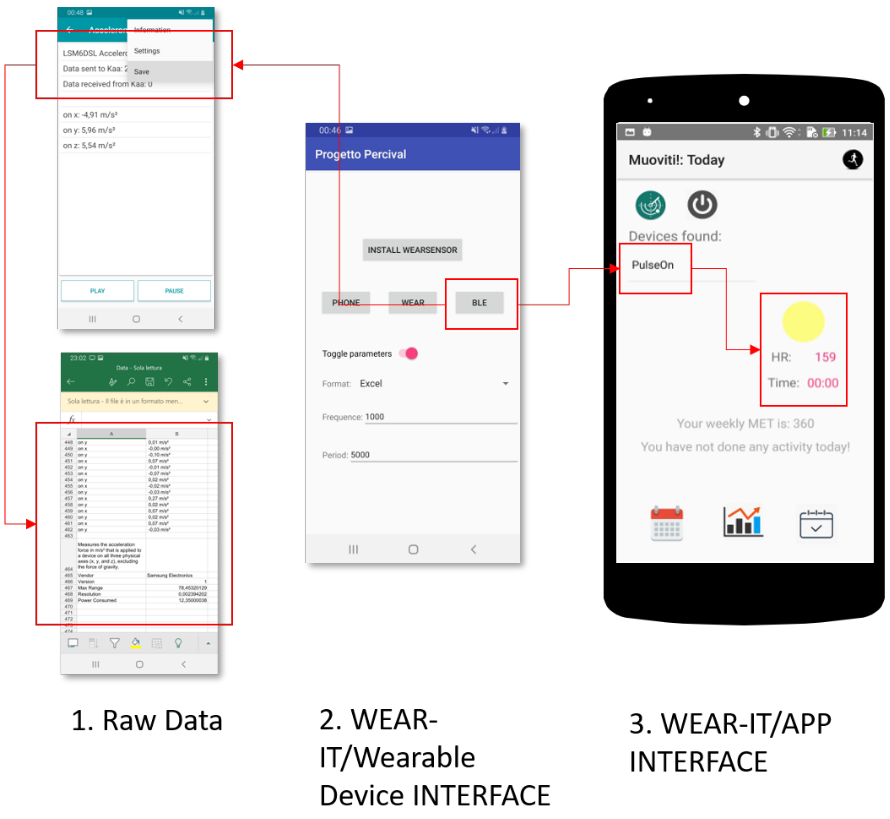


Figure 8. The three stages of app development in the case study.

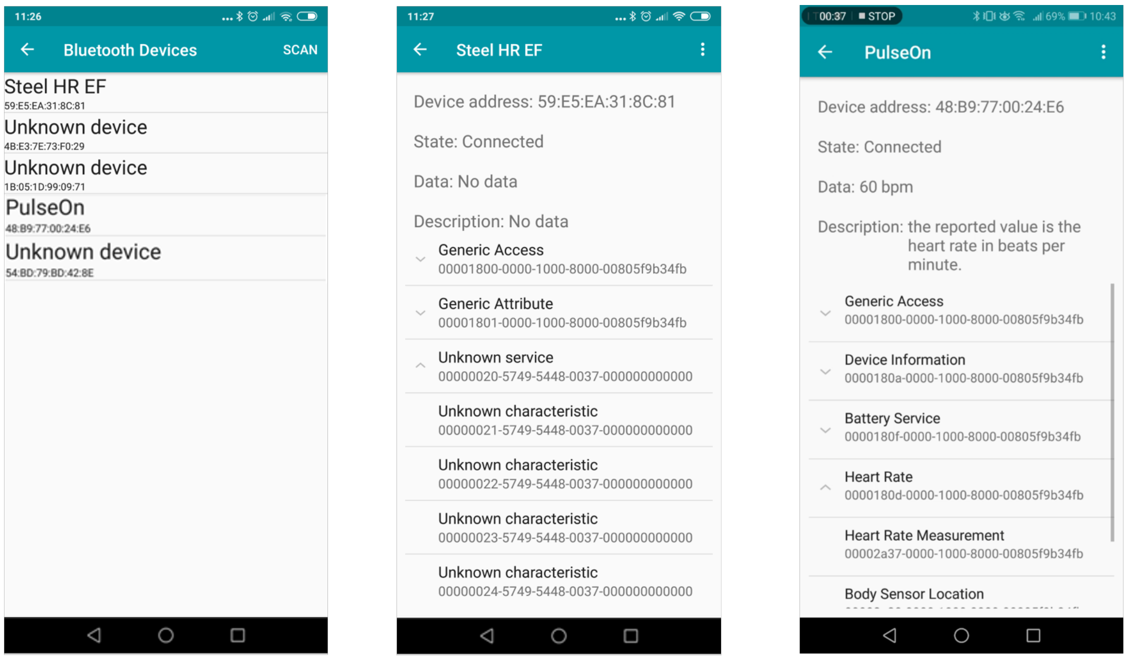
5. Discussion

In this paper, a conceptual and computational approach to the design of IoT-based intelligent systems is presented. The main idea is the definition of a wearable environment where intelligent systems can interact with sensors in order to maximize the performance of their reasoning strategy. To do this, a *middleware* layer has been inserted between the application and data layers, to provide the former with an opportune API to gather data from the latter. The main benefits emerging from this choice, represented in the diagram of Figure 6, are *interoperability* among applications at reasoning level and *reliability* of data made available to applications. In the following, these two aspects are further explained.

5.1. Interoperability at Reasoning Level

It is important to notice that such API has been designed and implemented taking care of the goal of a wearable environment: to define a mean to allow an application at reasoning level to perform its decision making process being sure to find and obtain necessary data.

Figure 9 refers to the case study above: two distinct BLE devices, namely $PulseOn^{TM}$ (see https://pulseon.com/) and $NokiaSteelHR^{TM}$ (see https://www.withings.com/uk/en/steel-hr) smartwatches, are included in the wearable environment definition. Both of them have a HR sensor, thus the MoveUp application could choose the one or the other indifferently, from its performance point of view. Unfortunately, if the $NokiaSteelHR^{TM}$ were selected to query its sensors, the WEAR-IT framework would not be able to discover available data sources. This means that BLE configuration of the device is not standard, thus no data can be delivered to a WES.



Steel HR EF
59:E5:EA:31:8C:81
Unknown device
4B:E3:7E:73:F0:29
Unknown device
1B:05:1D:99:09:71
PulseOn
48:B9:77:00:24:E6
Unknown device
54:BD:79:BD:42:8E

Device address: 59:E5:EA:31:8C:81
State: Connected
Data: No data
Description: No data
Generic Access
00001800-0000-1000-8000-00805f9b34fb
Generic Attribute
00001801-0000-1000-8000-00805f9b34fb
Unknown service
00000020-5749-5448-0037-000000000000
Unknown characteristic
00000021-5749-5448-0037-000000000000
Unknown characteristic
00000022-5749-5448-0037-000000000000
Unknown characteristic
00000023-5749-5448-0037-000000000000
Unknown characteristic
00000024-5749-5448-0037-000000000000

Device address: 48:B9:77:00:24:E6
State: Connected
Data: 60 bpm
Description: the reported value is the heart rate in beats per minute.
Generic Access
00001800-0000-1000-8000-00805f9b34fb
Device Information
0000180a-0000-1000-8000-00805f9b34fb
Battery Service
0000180f-0000-1000-8000-00805f9b34fb
Heart Rate
0000180d-0000-1000-8000-00805f9b34fb
Heart Rate Measurement
00002a37-0000-1000-8000-00805f9b34fb
Body Sensor Location

1.Wearable Environment BLE devices 2. Steel HR sensors: not available 3. PulseOn sensors: available

Figure 9. BLE devices comparison in WEAR-IT.

This is a problem common to many wearable devices, for which opportune APIs are (maybe partially) not available and they can only be interrogated through their official apps (e.g., *HealthMate*^{TM} (see https://www.withings.com/nl/en/health-mate)); the possibility to connect a proprietary app such as Health Mate to a middleware such as WEAR-IT, in order to overcome the problem described so far could open new possibilities of developing innovative application in mobile domains; one possible solution, according to the wearable environment definition, would be to extend the APIs set of WEAR-IT to allow an application a_i "playing" another application a_j:

$$API_{a_{i_{a_j}}} = \{play(a_j), stop(a_j), store(a_j)\}$$

An example of such interoperability is the development of MoveUp VR.The users of the MoveUp module are people characterized by sedentary lifestyle and great difficulties to perform regular physical activities due to many reasons. They result to be very frail psychologically and they could be discouraged by well-known barriers [46].

To support the users to plan their training session, an interactive function for calculating the best route to run across has been designed and implemented. Figure 10 shows the information flow concerning this goal. First, the user chooses the starting point on the map: this point could be concrete (e.g., the address of the hotel where the person stays during a business trip) or desirable (e.g., a beautiful point of interest). Then, an input form is activated where the following data must be inserted:

- *Duration of the PA session*: This value can be calculated from the MET amount suggested by MoveUp (remind that 120 MET are equivalent to 30 min of moderate PA). The user is free to setup the value according to his/her wishes.
- *Speed*: This value is set by default at 6 km/h, but the user can modify it according to his/her conditions.
- *Direction*: This value is set by default at 0 degrees, but the user can modify it exploiting the *rotation vector sensor* (if available) provided by WEAR-IT; for example, he/she could point towards a POI he/she sees on the map, in order to force the application to consider it in the calculation.

At the end of the calculation, the proposed route is highlighted on the map and the user can simulate to run it through the *StreetView*^{TM} activation. Doing so, he/she will be able to decide if the route neighborhood fits with his/her expectations or modifying it otherwise. In this case, the MET value exploited in the route calculus is asked to the MoveUp application; the MET value is not gathered from a sensor, being derived from HR thanks to the MoveUp decision's model. Much work must be done to extend the WEAR-IT API to enable an application to interface with another one to obtain data, and this is being addressed by ongoing work.

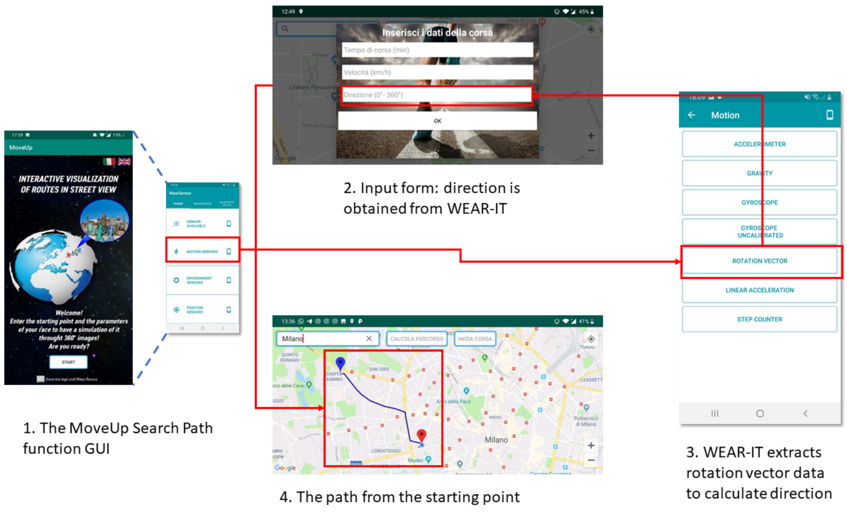

1. The MoveUp Search Path function GUI
2. Input form: direction is obtained from WEAR-IT
3. WEAR-IT extracts rotation vector data to calculate direction
4. The path from the starting point
INTERACTIVE VISUALIZATION OF ROUTES IN STREET VIEW
Inserisci i dati della corsa
Tempo di corsa (min)
Velocità (km/h)
Direzione (0°- 360°)
OK
Motion
ACCELEROMETER
GRAVITY
GYROSCOPE
GYROSCOPE UNCALIBRATED
ROTATION VECTOR
LINEAR ACCELERATION
STEP COUNTER
Milano
CALCOLA PERCORSO
INIZIA CORSA

Figure 10. The information flow to calculate the path for running in MoveUp.

5.2. Data Reliability

Expert systems, and wearable expert systems too, are generally able to work if some data are missing. The availability of faulty data is more critical, since the decision making process could be erroneously conducted. The probability that measures from wearable devices are incorrect is high, especially when data gathered concern medical parameters such as HR [47]. One of the most important goals of the wearable environment described is allowing applications at reasoning level to deal with reliable data. To this aim, some experiments on the wearable environment storage have been conducted.

Given an IoT device to measure *temperature*, *humidity*, *luminosity* and *battery voltage*, data gathered have been archived in the KAA platform for a period of 36 days. The resulting table contained 2,313,682 records. Table 13 summarizes the number of anomalies detected on them.

Table 13. Analysis of anomalies on data measurements.

Values	**Parameter**			
	Temperature	**Humidity**	**Luminosity**	**Voltage**
Null	901 (0.03%)	902 (0.03%)	93880 (4%)	526 (0.02%)
Outliers	383443 (16.6%)	299084 (12.9%)	0	8 (0.03%)

Figure 12 shows the distribution of values for the temperature and humidity variables. It is possible to notice how the temperature curve is uniformly distributed between 10 and 35 °C; these values are much more than others, constituting the set of correct values for the temperature detection. Moreover, there are some peaks of anomaly values that are very frequent. i.e., 0.51 and 121.49 °C. About humidity, the distribution is Gaussian between 24% and 60% and uniform between 0% and 24%; moreover, there are very few measurements in the (60%, 100%] range, with a significant peak of anomaly at 114.45% (about 4000 units).

Figure 11 presents the distribution of values for the luminosity and voltage parameters. The luminosity distribution is linear on the whole interval, except for peaks at the limits (0 and 1600 lux respectively) and some anomalies at 41, 43 and 48 lux values. Finally, the battery voltage distribution is Gaussian, without any significant peak or anomaly.

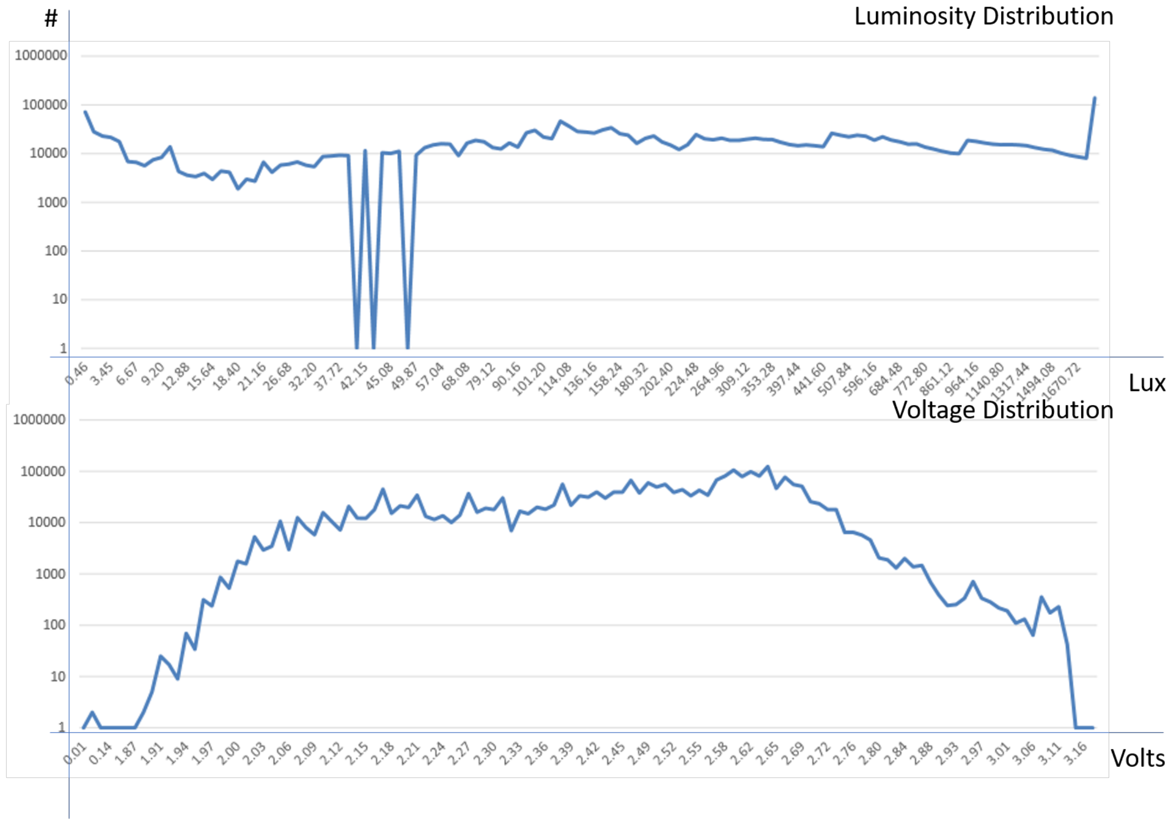


Figure 11. Distribution of luminosity and battery voltage values archived in the KAA storage.

The analysis of curves of parameters values allowed deriving opportune sets of acceptable values for sensors measurements, as shown in Table 14. The temperature range was extended to [−15 °C, 50 °C] to take care of winter and summer periods (the experiment was conducted łlast spring).

Table 14. Acceptable values for temperature, humidity, luminosity and voltage.

Parameter	**Measurement**	
	Minimum Value	**Maximum Value**
Temperature	−15 °C	50 °C
Humidity	0% h	100% h
Luminosity	0 Lux	2500 Lux
Voltage	1, 5 Volts	3, 5 Volts

The last step of the intervention is the nullification of out of bounds values. The consequent loss of information due to the deletion of data from the storage is compensated by the lower probability of taking wrong decisions as a consequence of sensors failures. Figure 13 shows the decrease of standard deviation of variables at the end of this intervention. According to the knowledge domain, out of bound values could be substituted by reasonable, standard values: for example, in the MoveUp case study, a possible strategy to improve the data quality could be replacing extremely high outlier values of HR with user's MHR.

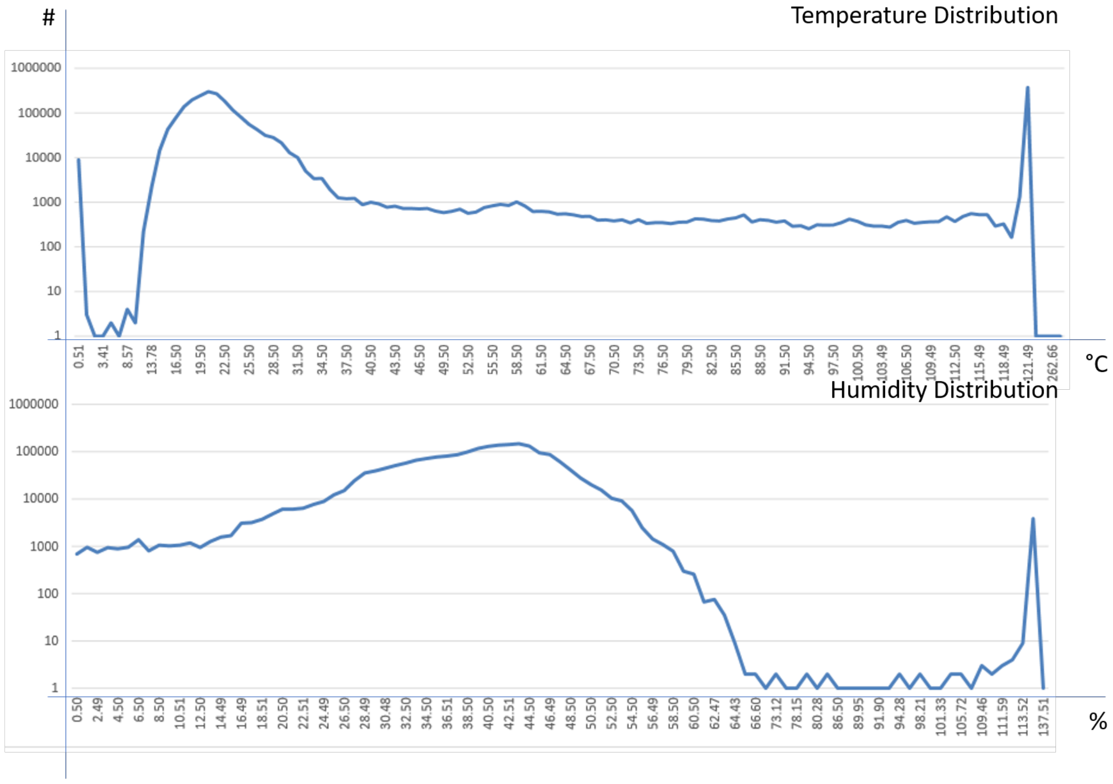


Figure 12. Distribution of temperature and humidity values archived in the KAA storage.

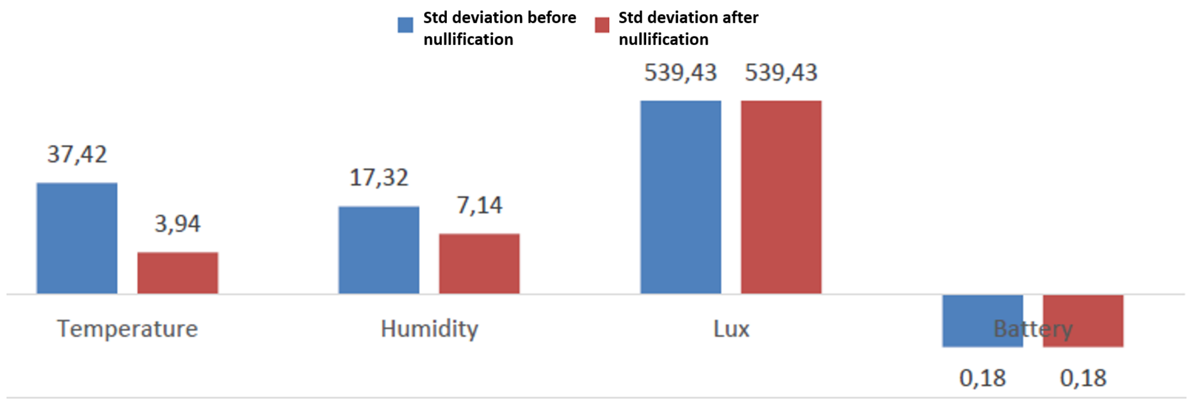


Figure 13. Standard deviation of temperature, humidity, luminosity and voltage values before and after nullification.

6. Conclusions

In this paper, we propose the *wearable environment* notion and architecture as a means to cast into a unique conceptual and computational framework data acquisition from sensors, data representation through wearable devices and their use by means of wearable expert systems. The heart of this notion is the data acquisition and representation layer, where the WEAR-IT platform has been designed and implemented to build an efficient, reliable and scalable bridge between raw data and applications.

To our knowledge, only three applications provide services comparable to WEAR-IT: *SensorCap, Sensor Toolbox and LightBlue Explorer* (see https://play.google.com/store/apps/details?id=br.ufmg.dcc.ssig.sensorcap; https://play.google.com/store/apps/details?id=com.galaxy.sensortoolbox; and https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer). With respect to them, only WEAR-IT can be integrated (see Figure 8) into a wearable environment at the moment, while the others can be used as external services to collect and store data that can be then elaborated by the application.

Indeed, data reliability is one of the most important factor for WES development. Consequently, given that, in a wearable environment, WEAR-IT acts as an intermediate data collector, it is important to ensure that the *quality* of data stored is high enough (see Section 5.2). This issue is currently addressing our research [48].

Finally, a future extension of the present work concerns the definition of an extended wearable environment infrastructure, where many users, each one equipped with a *personal wearable environment*, could be asked to join an *integrated*, *geographically distributed* and *scalable* wearable environment. In this environment, an application could query end-points and acquire data from them according to their position and reachability from mobile nodes [33].

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Program Interface
BLE	Bluetooth Low Energy
GUI	Graphical User Interface
HR	Heart Rate
IoT	Internet of Things
IPA	Intense Physical Activity
MET	Metabolic Equivalent of Task
mHealth	Mobile Health
MHR	Maximum Heart Rate
MPa	Moderate Physical Activity
NCD	Non-communicable Diseases
NTP	Network Time Protocol
OS	Operating System
PA	Physical Activity
SE	Self Efficacy
VR	Virtual Reality
WES	Wearable Expert System
UUID	Universal Unique IDentifier
WBAN	Wireless Body Area Network
WSN	Wireless Sensor Network
WSNME	Wireless Sensor Networks with Mobile Elements
WHO	World Health Organization

References

1. Jovanović, J.; Gašević, D. Achieving knowledge interoperability: An XML/XSLT approach. *Expert Syst. Appl.* **2005**, *29*, 535–553.
2. Garcia-de Prado, A.; Ortiz, G.; Boubeta-Puig, J. COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things. *Expert Syst. Appl.* **2017**, *85*, 231–248.
3. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
4. Sartori, F.; Melen, R. Wearable expert system development: definitions, models and challenges for the future. *Program* **2017**, *51*, 235–258.
5. Čolaković, A.; Hadžialić, M. Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. *Comput. Netw.* **2018**, *144*, 17–39.
6. Lee, J.; Kim, D.; Ryoo, H.Y.; Shin, B.S. Sustainable wearables: Wearable technology for enhancing the quality of human life. *Sustainability* **2016**, *8*, 466.

7. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376.
8. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Int. Things J.* **2017**, *4*, 1125–1142.
9. Kim, J.; Yun, J.; Choi, S.C.; Seed, D.N.; Lu, G.; Bauer, M.; Al-Hezmi, A.; Campowsky, K.; Song, J. Standard-based IoT platforms interworking: implementation, experiences, and lessons learned. *IEEE Commun. Mag.* **2016**, *54*, 48–54.
10. Zeng, D.; Guo, S.; Cheng, Z. The web of things: A survey. *JCM* **2011**, *6*, 424–438.
11. Wong, B.P.; Kerkez, B. Real-time environmental sensor data: An application to water quality using web services. *Environ. Model. Softw.* **2016**, *84*, 505–517.
12. Al Rasyid, M.U.H.; Sayfudin, A.; Basofi, A.; Sudarsono, A. Development of semantic sensor web for monitoring environment conditions. In Proceedings of the 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA), Lombok, Indonesia, 28–30 June 2016; pp. 607–612.
13. Mukhopadhyay, S.C. Wearable sensors for human activity monitoring: A review. *IEEE Sens. J.* **2015**, *15*, 1321–1330.
14. Edwards, J. Wireless sensors relay medical insight to patients and caregivers [special reports]. *IEEE Signal Process. Mag.* **2012**, *29*, 8–12.
15. Shaltis, P.A.; Reisner, A.T.; Asada, H.H. Cuffless blood pressure monitoring using hydrostatic pressure changes. *IEEE Trans. Biomed. Eng.* **2008**, *55*, 1775–1777.
16. Poh, M.Z.; Kim, K.; Goessling, A.; Swenson, N.; Picard, R. Cardiovascular monitoring using earphones and a mobile device. *IEEE Pervasive Comput.* **2012**, *11*, 18–26.
17. Salvo, P.; Di Francesco, F.; Costanzo, D.; Ferrari, C.; Trivella, M.G.; De Rossi, D. A wearable sensor for measuring sweat rate. *IEEE Sens. J.* **2010**, *10*, 1557–1558.
18. Strohrmann, C.; Harms, H.; Kappeler-Setz, C.; Troster, G. Monitoring kinematic changes with fatigue in running using body-worn sensors. *IEEE Trans. Inf. Technol. Biomed.* **2012**, *16*, 983–990.
19. Ermes, M.; Pärkkä, J.; Mäntyjärvi, J.; Korhonen, I. Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions. *IEEE Trans. Inf. Technol. Biomed.* **2008**, *12*, 20–26.
20. Rodgers, M.M.; Pai, V.M.; Conroy, R.S. Recent advances in wearable sensors for health monitoring. *IEEE Sens. J.* **2015**, *15*, 3119–3126.
21. Alemdar, H.; Ersoy, C. Wireless sensor networks for healthcare: A survey. *Comput. Netw.* **2010**, *54*, 2688–2710.
22. Movassaghi, S.; Abolhasan, M.; Lipman, J.; Smith, D.; Jamalipour, A. Wireless body area networks: A survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1658–1686.
23. Appelboom, G.; Camacho, E.; Abraham, M.E.; Bruce, S.S.; Dumont, E.L.; Zacharia, B.E.; Connolly, E.S. Smart wearable body sensors for patient self-assessment and monitoring. *Arch. Public Health* **2014**, *72*, 28.
24. Casselman, J.; Onopa, N.; Khansa, L. Wearable healthcare: Lessons from the past and a peek into the future. *Telemat. Inform.* **2017**, *34*, 1011–1023.
25. Mathie, M.J.; Coster, A.C.; Lovell, N.H.; Celler, B.G.; Lord, S.R.; Tiedemann, A. A pilot study of long-term monitoring of human movements in the home using accelerometry. *J. Telemed. Telecare* **2004**, *10*, 144–151.
26. Sazonov, E.S.; Fulk, G.; Sazonova, N.; Schuckers, S. Automatic recognition of postures and activities in stroke patients. In Proceedings of the 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Minneapolis, MI, USA, 2–6 September 2009; pp. 2200–2203.
27. Giansanti, D.; Maccioni, G.; Morelli, S. An experience of health technology assessment in new models of care for subjects with Parkinson's disease by means of a new wearable device. *Telemed. Health* **2008**, *14*, 467–472.
28. Aziz, O.; Atallah, L.; Lo, B.; ElHelw, M.; Wang, L.; Yang, G.Z.; Darzi, A. A pervasive body sensor network for measuring postoperative recovery at home. *Surg. Innov.* **2007**, *14*, 83–90.
29. Patel, M.S.; Asch, D.A.; Volpp, K.G. Wearable devices as facilitators, not drivers, of health behavior change. *JAMA* **2015**, *313*, 459–460.
30. Amft, O.; Tröster, G. Recognition of dietary activity events using on-body sensors. *Artif. Intell. Med.* **2008**, *42*, 121–136.
31. Benedetti, M.G.; Di Gioia, A.; Conti, L.; Berti, L.; Degli Esposti, L.; Tarrini, G.; Melchionda, N.; Giannini, S. Physical activity monitoring in obese people in the real life environment. *J. Neuroeng. Rehabil.* **2009**, *6*, 47.

32. Sciacqua, A.; Valentini, M.; Gualtieri, A.; Perticone, F.; Faini, A.; Zacharioudakis, G.; Karatzanis, I.; Chiarugi, F.; Assimakopoulou, C.; Meriggi, P.; et al. Validation of a flexible and innovative platform for the home monitoring of heart failure patients: preliminary results. In Proceedings of the 2009 36th Annual Computers in Cardiology Conference (CinC), Park City, UT, USA, 13–16 September 2009; pp. 97–100.
33. Di Francesco, M.; Das, S.K.; Anastasi, G. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sens. Netw.* **2011**, *8*, 1–31.
34. Anastasi, G.; Conti, M.; Di Francesco, M.; Passarella, A. Energy conservation in wireless sensor networks: A survey. *Ad. Hoc. Netw.* **2009**, *7*, 537–568.
35. Sartori, F.; Melen, R. Time Evolving Expert Systems Design and Implementation: The KAFKA Approach. In Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015), Lisbon, Portugal, 12–14 November 2015; pp. 84–95.
36. Melen, R.; Sartori, F.; Grazioli, L. Modeling and understanding time-evolving scenarios. In Proceedings of the 19th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, FL, USA, 12–15 July 2015; pp. 267–272.
37. Korb, K.B.; Nicholson, A.E. *Bayesian Artificial Intelligence*; CRC Press: Boca Raton, FL, USA, 2010.
38. Sartori, F.; Melen, R.; Lombardi, M.; Maggiotto, D. Virtual round table knights for the treatment of chronic diseases. *J. Reliab. Intell. Environ.* **2019**, *5*, 131–143.
39. Mealy, G.H. A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* **1955**, *34*, 1045–1079.
40. Hallal, P.C.; Bauman, A.E.; Heath, G.W.; Kohl 3rd, H.W.; Lee, I.M.; Pratt, M. Physical activity: More of the same is not enough. *Lancet* **2012**, *380*, 190.
41. Conn, V.S.; Hafdahl, A.R.; Mehr, D.R. Interventions to increase physical activity among healthy adults: meta-analysis of outcomes. *Am. J. Public Health* **2011**, *101*, 751–758.
42. WHO. *Global Recommendations on Physical Activity for Health*; World Health Organization: Geneva, Switzerland, 2010.
43. Baretta, D.; Sartori, F.; Greco, A.; D'Addario, M.; Melen, R.; Steca, P. Improving physical activity mhealth interventions: development of a computational model of self-efficacy theory to define adaptive goals for exercise promotion. *Adv. Hum. Comput. Interact.* **2019**, *2019*, doi:10.1155/2019/3068748.
44. Armstrong, T.; Bonita, R. Capacity building for an integrated noncommunicable disease risk factor surveillance system in developing countries. *Ethn. Dis.* **2002**, *13*, S13–8.
45. Baretta, D.; Sartori, F.; Greco, A.; Melen, R.; Stella, F.; Bollini, L.; D'addario, M.; Steca, P. Wearable devices and AI techniques integration to promote physical activity. In Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct, Florence, Italy, 6–9 September 2016; pp. 1105–1108.
46. Manaf, H. Barriers to participation in physical activity and exercise among middle-aged and elderly individuals. *Singap. Med. J.* **2013**, *54*, 581–586.
47. Kumar, M.S.; Dhulipala, V.S.; Baskar, S. Fuzzy unordered rule induction algorithm based classification for reliable communication using wearable computing devices in healthcare. *J. Ambient. Intell. Humaniz. Comput.* **2020**, 1–12, doi:10.1007/s12652-020-02219-0.
48. Sartori, F.; Melen, R.; Giudici, F. IoT Data Validation Using Spatial and Temporal Correlations. In Proceedings of the Research Conference on Metadata and Semantics Research, Rome, Italy, 28–31 October 2019; pp. 77–89.