Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 637130, 12 pages
http://dx.doi.org/10.1155/2015/637130

Research Article

A Model-Driven Approach for Wearable Systems Developments

Angel Ruiz-Zafra,[1] Manuel Noguera,[1] Kawtar Benghazi,[1] and Sergio F. Ochoa[2]

[1]*Department of Languages and Informatics Systems, University of Granada, 18071 Granada, Spain*
[2]*Department of Computer Science, University of Chile, Santiago, Chile*

Correspondence should be addressed to Angel Ruiz-Zafra; angelr@ugr.es

Received 24 May 2015; Accepted 15 July 2015

Academic Editor: Raffaele Gravina

Copyright © 2015 Angel Ruiz-Zafra et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper proposes a model-driven approach for developing high-level software interfaces that allow developers to interact with wearable devices easily. These components hide the heterogeneity of the devices interfaces and provide developers with a simple and homogeneous way to interoperate with these digital peripherals. The use of this approach also allows reducing risks and development efforts.

1. Introduction

In recent years, the progress in several domains, such as electronics, communications, and computing has led to the creation of portable high-tech devices that provide several services to people and systems. In particular, wearable devices (or *wearables*) are launched on an almost weekly basis [1, 2]. From a simple wristwatch or GPS to the last state-of-the-art smart glasses, wearables can be found in a variety of application domains, ranging from healthcare and security to entertainment and business, or simply as service providers [3]. These devices (that usually embed sensing capabilities) are becoming increasingly popular, because of their potential for enhancing the quality of people's everyday life [4, 5].

There is a large and still growing variety of wearables since each vendor tries to differentiate their products from other vendors, in terms of services provided and costs. Consequently, they include proprietary software interfaces for these products, which cause clients and users to become highly dependent on the wearables of such a vendor.

Unfortunately, this lack of standard and open software interfaces (i.e., APIs (Application Programming Interfaces)) to interact with these devices limits the development of systems that really improve the quality of the people's life. For instance, if a developer has to implement a wearable system to monitor patients with Alzheimer's, the solution could involve at least a GPS, a computing unit, and a network interface. The GPS would determine the location of the monitored person. The computing unit would process such information to determine if the patient is moving according to his/her behavioral patterns. When this unit detects important changes in the person activity, it may assume that the patient is getting lost, and therefore it uses the network interface to notify other people (e.g., caregivers, relatives, or medical personnel).

The development of such a monitoring system would require that the software engineer would have to deal with the low-level software interfaces of the involved devices (the GPS and the computing unit), which probably vary depending on the device and vendor. Moreover, the interfaces of each device, as well as the details about data stream formats and transmission protocols, are usually provided through verbose datasheets, for which careful study and some training are also required so as to understand them.

This situation not only limits devices integration and interoperability of the solutions but also jeopardizes the development efforts of these systems, since it requires understanding and managing every device interface as well as writing specific source code for each device and operating system involved in the solution. Therefore, only highly experienced developers are able to address these challenges. Moreover, these developments require an extra effort (in terms of time and cost) and push developers to face unconventional risks derived from dealing with low-level abstractions used in

datasheets device specifications, thereby reducing the success rate of these projects.

This paper introduces a proposal to deal with the heterogeneity of software interfaces for wearable devices, by enabling the design of high-level software interfaces that ease the development of these systems. The proposal, called model-driven approach for wearable system development (MDWSD), reduces the complexity and effort of using wearables, regardless of their specific characteristics (e.g., purpose, API, and supported communication protocols).

MDWSD provides a user interface that allows vendors to create high-level interfaces for their devices. Software designers (or architects), who may want to provide simple and homogeneous abstractions about these devices' APIs, can do the same for reducing the effort and risks of wearable system developments. Thus, developers no longer have to deal with low-level complex interfaces for interacting with a wearable device.

The proposed approach considers the use of several components: (1) a metamodel that eases the definition of the device design features and its API; (2) a design tool to be used by architects to model the wearables; (3) a mechanism to generate the device model from the proposed metamodel; (4) a software component, called *coordinator*, which uses the instances of the metamodel (i.e., device models) to handle the devices; and (5) a process that shows how to articulate the use of the previously mentioned components.

The device models are represented in a custom dynamic language, named wearable markup language (WML), which allows designers to specify the elements and features that are mandatory to interact with the device. Because of this, each wearable has only one model, as an instance of the proposed metamodel. Furthermore, the design tool has been conceived to generate wearable device models easily, particularly for devices with sensing capabilities. The *coordinator* component has been designed to handle the device models (i.e., the instances of the metamodel) in an automatic and seamless way, thereby enabling interaction through an easy-to-use high-level software interface. The vendors' software designers are responsible for defining these high-level APIs using the design tool.

The rest of the paper is organized as follows: Section 2 describes related work; Section 3 presents a brief background on device development and integration; Section 4 presents the proposed model-driven development approach; Section 5 describes a case study in which the proposed approach was used; and Section 6 summarizes the conclusions and outlines the future work.

2. Related Work

Wearable computing has become an emerging and promising field because of the recent advantages in mobile computing and wireless communication technology. Several proposals based on the use of wearables are framed into the concept of wireless body area network (WBAN), where wearables from several natures support activities in various domains, such as healthcare or entertainment [6–8].

Most of these proposals have been designed to monitor people activities or health conditions. Provided that these solutions are focused on addressing a particular problem using specific devices, the seamless integration of several wearables in software applications is usually not addressed. Instead, the developers must create custom source code from scratch to interact with the wearables used in these projects.

Some other projects have considered the wearables integration an important challenge to address, and therefore they have proposed design solutions and infrastructures (e.g., middleware) to facilitate the software development process. For instance, in [9] the authors present an approach to improve the integration of Bluetooth based devices—in existing systems. In [10, 11] the researchers describe a middleware for supporting e-health environments, and in [12] a revision of several middleware is presented exposing their features, strengths, and weakness.

Most of middleware proposed to improve the integration of wearable devices is linked to particular elements; for example, specific communication protocols should be used [10] or it can be applied to a certain domain [11, 13]. Although they have shown to be useful, they provide a nonautomatic integration of devices and a limited support for device heterogeneity. This makes that developers have to write extra code to perform the devices integration process.

There are also proposals that intend to address the heterogeneity and lack of standardization for interacting with wearables, using a model-driven approach. For instance, in [14, 15] the authors propose an approach to enhance the data acquisition of wearables. Going a step further, the projects presented in [16, 17] explore the devices integration process, the lack of standardization, and the heterogeneity of wearable computing contexts. Based on that situation, the researchers propose model-driven approaches in order to address these challenges.

Recognizing the usefulness of the related proposals, they have limitations to address the challenge presented in this paper. Their main limitations are the following:

(i) The integration of new wearables in already deployed systems, as well as the data management of wearables to provide a proper dissemination, is not fully covered. This is because in some cases the developers should create new source code from scratch.

(ii) The use of these solutions requires understanding technical documentation, which is not usually written for regular developers.

In order to help address these limitations, this paper supports device development and integration, addressing problems such as the large variety of devices to integrate and the lack of the standardization about how to consume device services. The model-driven approach for wearable systems development (MDWSD) proposed in this paper intends to provide a solution for the integration and data management of wearable in an easy and quick way, providing a high-level interface to developers. Typically, the use of these APIs helps them reduce the development effort, risks, and complexity.

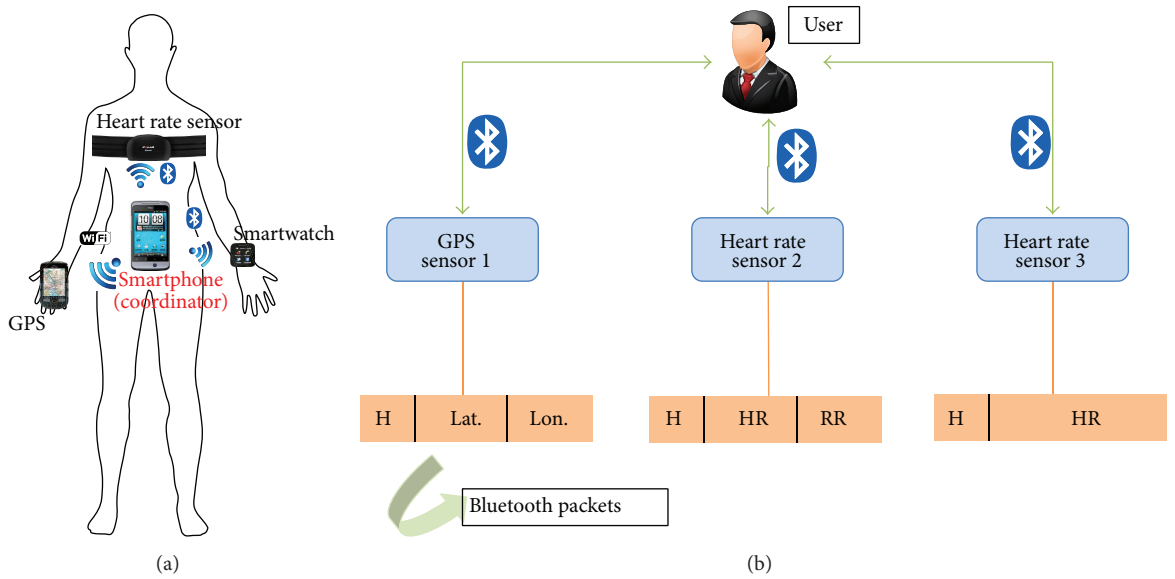


Figure 1: Smartphone as coordinator (a) and different packets of information (b).

3. Integration of Wearable Devices

Wearable devices are small independent digital components that have been inserted in clothing, apparel, or personal portable objects and that have been designed or created for a specific purpose [18]. These devices are able to provide one or more functionalities, for example, to inform the user position, body temperature, and heart rate.

Some devices, like smartphones or wristwatches, may use these functionalities to provide complex services, like determining the health condition of a mobile user in real-time. Particularly, smartphones have become reference devices in wearable computing. Thanks to their computing power, usability, and wireless communication capabilities, they are gaining acceptance to coordinate other wearable devices of mobile users.

Wearable devices can use several, wired and wireless, communication interfaces, such as USB, Bluetooth, and WiFi, to exchange information with other components and devices. Regardless of the communication technology to be used and whether the information is sent by request (packets) or stream, wearable devices are designed to serialize the information and provide the results as an array of bytes with a predefined or undetermined length, structure, and content.

A customized device design entails that the structure of the serialized information provided by one device may be different from other devices that play the same role, since they have been designed by other vendors according to other criteria or requirements. Consequently, other aspects (such as the packet length, the provided information, or even the order of the information presentation) may be also different.

The left-hand side of Figure 1 depicts a wearable computing system with a smartphone as a central management device (i.e., the smartphone acts as a coordinator device), which monitors various other devices (a GPS, a smartwatch, and a breast-band heart rate monitor). The information gathered from these sensing devices (e.g., the user position, heart rate, and RR interval, time between two beats) starts with a header (H) to identify the starting point of the information packet, due to the different format and length of the packets they provide (right-hand side of Figure 1).

The API (Application Programming Interface), designed by the architects from a particular vendor for interacting with one of the devices, provides a datasheet with all the device information (e.g., its technological features, size, and weight). The developers should use this datasheet to know how to interact with such a device, regardless of other concerns, like the communication protocols supported by the device. As a result, even two devices, designed by the same company, by the same engineers, and for the same purpose, could differ significantly in how to interact with them through their respective APIs.

This lack of standardization of the APIs jeopardizes the integration of wearables devices and leads software developers to several anomalous situations. For instance, if a vendor decides to release a new firmware to include additional communication capabilities in an already released device, the developers should write new source code and sometimes from scratch. Situations like this put in risk the development and operation of wearable computing solutions.

The dynamic nature of technology evolution should also be considered in the devices integration. For instance, we still have solutions using legacy wearable devices, which can only save information in an internal storage medium (e.g., MMC, SDCard, or Hard Drive). Although this reality represents a restriction in the design of wearable computing systems, it should be also considered for ensuring the devices integration.

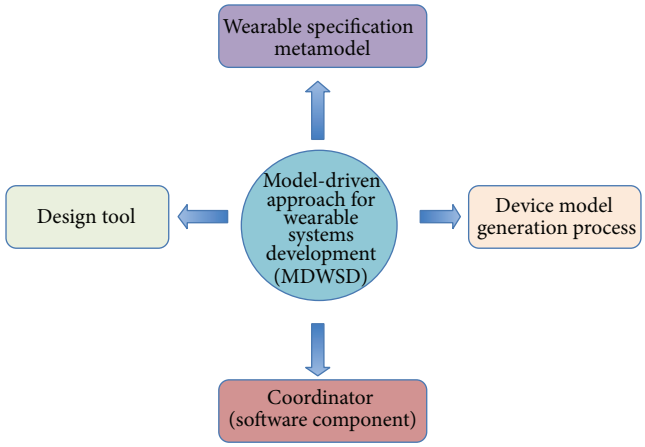


Figure 2: The proposed model-driven approach for wearable systems development.

4. The Proposed Development Approach

In order to address the challenge of using heterogeneous wearable devices in a system, we propose the model-driven approach depicted in Figure 2, which encompasses (1) a metamodel to define the different wearable device features, (2) a design tool to specify the features and the manner to interact with a device, (3) a device model generation process to obtain a device model from the metamodel, and (4) a coordinator component which is the software component that interprets the device models (metamodel instances) to interact with the corresponding devices. In the following sections, we describe each contribution in further detail as well as the integration process.

4.1. Wearable Specification Metamodel. The metamodel proposed in this paper has been designed to be extendable and open, in order to enable the idea that new features and elements can be included in the future. This ensures that any wearable device can be modeled and eases the interaction with the devices from a software perspective. The metamodel is intended to support the design interface of any wearable device.

In the design process of the metamodel, some key concerns related to wearable devices have been identified. These concerns are represented or managed differently depending on the wearable device. The proposed metamodel encapsulates this heterogeneity and provides a single API to the developer, regardless of the device that he/she is manipulating. Thus, the metamodel helps ease the device integration. The key concerns considered in the metamodel are as follows:

(i) The communication protocol is indicated in the metamodel, but how to handle it depends on the platform. For instance, protocols like WiFi, Bluetooth, and ZigBee are IEEE-defined communication standards; however, how these are used is still platform-dependent. While BluetoothSocket and BluetoothDevice are the API proposed in the Android Platform to work with Bluetooth Protocol, CoreBluetooth is the one proposed by iOS. They both work with the same protocol, but not in the same way from a developer's point of view.

(ii) The wearable devices provide the information in a serialized array of bytes using two alternative ways: by request (synchronous packet) or by stream (asynchronous).

(iii) The information provided by a specific wearable device is typically variable in format and length (buffer size, structure, and content).

(iv) The dynamic information provided by a wearable device is specified in the same packet/stream. However, it can be in a predefined or undetermined position; that is, the position of the payload and other fields can vary within the buffer.

(v) The information provided by a wearable device starts with a customized value that we called *pivot* (*header* by the classical vendors and designers). This allows the packets that compose the information to be identified.

(vi) The information provided by a wearable device has a cyclic redundancy check (CRC) or other similar mechanism to check that the packet is valid.

(vii) The information provided by a wearable device is divided into groups of bytes. The length and position of these groups may be predefined or undetermined. The length of the buffer and the information position varies according to the service provided.

The metamodel is represented in a UML diagram in Figure 3.

Depending on how the information provided through a communication protocol is managed, various features related to data structure/dissemination have been considered and included in the metamodel, in order to represent an abstraction of a wearable device. The considered features are as follows:

(i) *Type of Operation*. It can be input (stream), output, or both (request).

(ii) *Pivot Element*. This element indicates where the package of information (buffer) starts.

(iii) *Payload Information*. This is the cargo of the data transmission, that is, the data representing the fundamental purpose of the transmission. For instance, the latitude and longitude in a package of information are sent by a GPS sensor.

(iv) *Fields*. This component includes the information from the device, such as values to represent the packet length or the starting point of the payload.

(v) *CRC or Check Method*. This represents the set of operations to validate incoming packets/information.

(vi) *Type of Buffer for Each Functionality/Service*. As mentioned before, it can be predefined or undetermined.

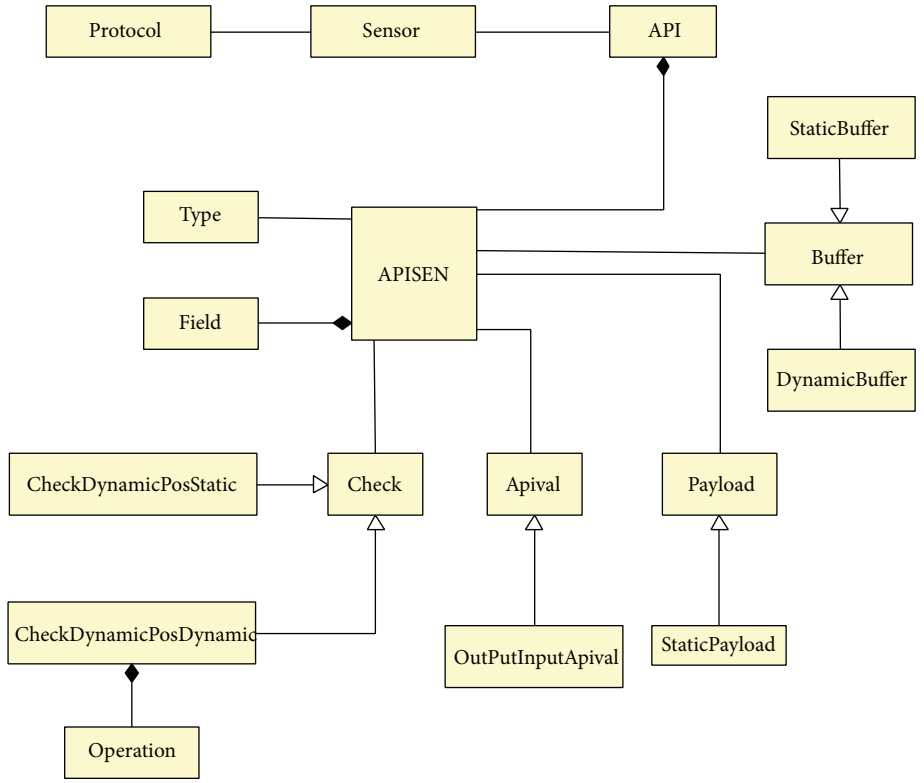


Figure 3: Wearable device metamodel.

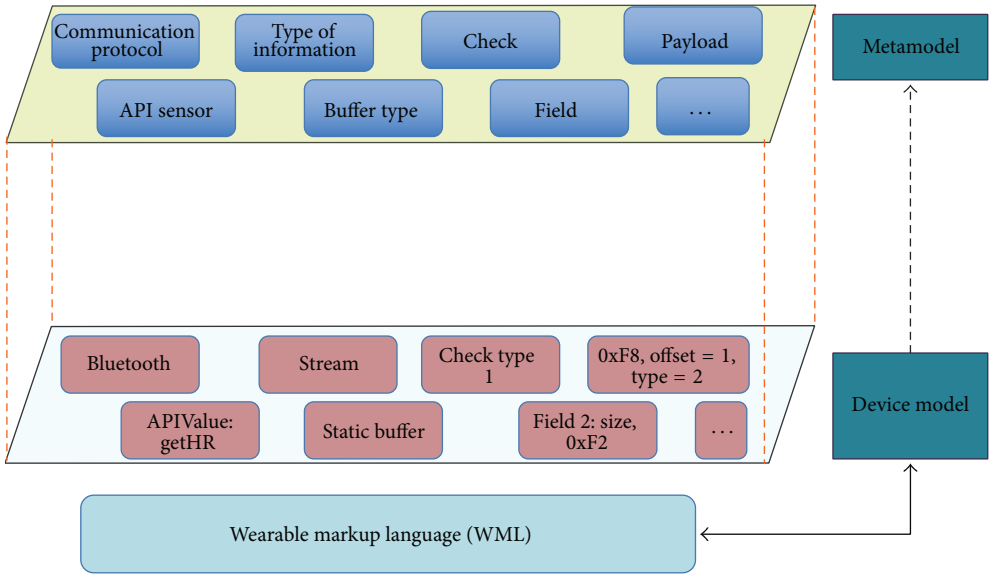


Figure 4: Example of an instance of the metamodel.

One of the design objectives of the metamodel was to make it customizable, so that it could meet present and future requirements of new wearables, as they are released. Because of this, it might be possible to create new customized models to cover forthcoming technologies and functions, by simply defining new concepts in the metamodel to support these new features and their implementation in the coordinator. The models generated from the metamodel (i.e., the instances) are represented in a custom markup language called wearable markup language (WML). These models are identified by a unique identifier called *Wearable Unique Identifier* (WUI) and they contain all the information required to interact with

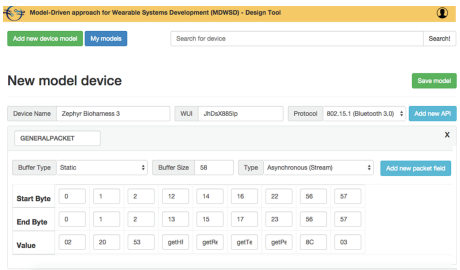


Figure 5: User interface of the design tool.

the device, such as the communication protocol used, values and functions provided by the wearable, and marshaling information (Figure 4).

4.2. Design Tool. Most wearables usually provide a datasheet or another kind of document where the developers can learn the device features and functionalities, in order to find out how the wearable works, that is, how to use the services of the device to get access to the sensed data. In some cases the datasheet is a huge file with all the device specifications on low-level implementation and data representation details, which hinders the use of the device for developers. Knowing how the device works is mandatory, but it usually requires expert knowledge.

In order to address this requirement, we have defined the role of *designer* and developed a design tool to support this task. The designer is a person with the required expert knowledge to identify the features and functionalities of the device. This person is also able to work with the device at low level and define a model that represents the device, thus avoiding the fact that developers have to deal with it. In order to help designers specify the devices models we have created a design tool. This tool is a web application conceived to be easy to use, enabling the design of any wearable easily and in a quick manner. Figure 5 shows a screenshot of the design tool.

The designer, taking the datasheet and other information concerning the device operation (usually provided by the device's vendor), uses the design tool to define the features of any wearable at a high level of abstraction. The designer also establishes how to interact with the device through its API, generating thus a device model. Such a model is represented in WML.

In this manner, just one user or team with the role of designer is required to produce the model of a wearable. One developer with the WUI of the device, together with its API (defined by the designer), can use the device in an easy way just using the coordinator.

4.3. Device Model Generation Process. The metamodel, shown in Figure 3 and represented in an XML file, contains the different features of any (so far) wearable device, for example, the communication protocol, how the information is sent, and the packet-checking method. In order to define a device model (i.e., an instance of the metamodel), some of these features will be used to represent the device features.

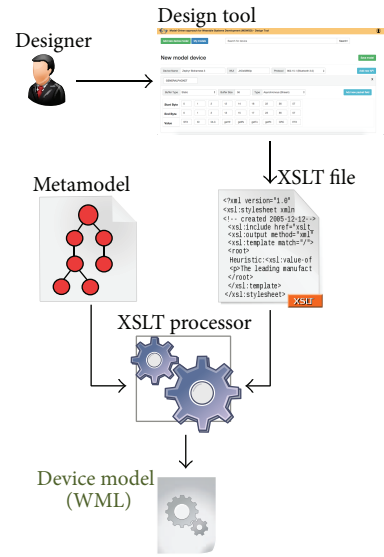


Figure 6: Device model generation procedure.

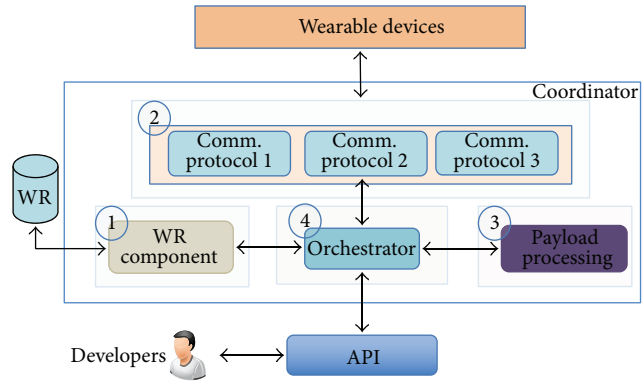


Figure 7: Architecture of the coordinator component.

Therefore, it is mandatory to count on a procedure to get instances of the metamodel. In this approach it was implemented as a transformation process from the metamodel to the device model, as shown in Figure 6. The process to obtain the different device models is based on XSLT (Extensible Stylesheet Language Transformation) approach [19].

The designer, using the design tool, models the device. Then, the design tool automatically generates a style sheet document (XML file) based on the XSLT that represents the design of the device, that is, its communication protocol, type of buffer, and API. After that, a service generates a custom XML file (in WML) that represents the device model, by using an XSLT Processor, using the style sheet document and the metamodel file.

4.4. Coordinator. The metamodel proposed in this approach has been designed to generate device models to avoid facing the complexity of managing low-level detail information when coding for wearable-based systems. Although these models contain the necessary information to interact with

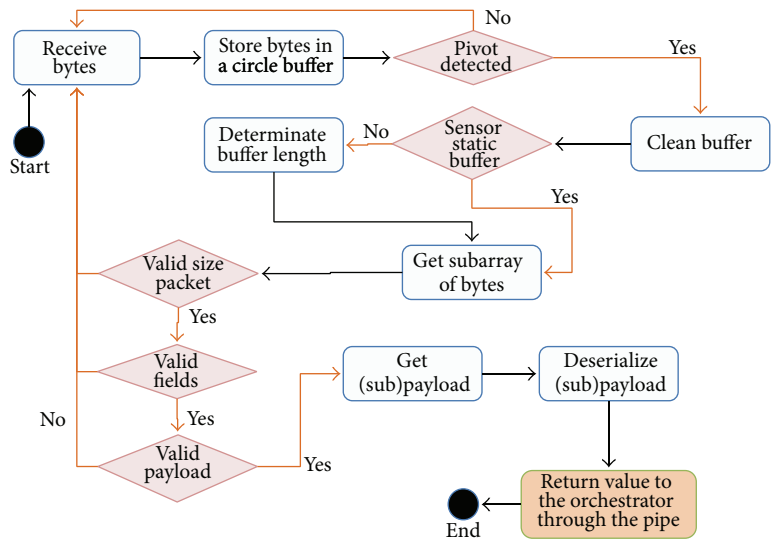


Figure 8: Payload processing.

the devices, it is also necessary that the software allows applications to use the wearable through this model.

The *coordinator* component is responsible for the models integration, and it acts as a bridge between the device models and the developers, by automating the integration (detection, synchronization) and easing the use of the devices for the developers.

Figure 7 shows a diagram that represents the coordinator design and its different elements. The coordinator is composed of four main components, each one has a specific purpose.

(1) Wearable Repository (WR). This component is responsible for interacting with the wearable models repository hosted on an external server (or in the cloud), where all the device models are stored. The WR component (labeled as 1 in Figure 7) uses a service specifying the WUI to obtain each device model.

(2) Communication Protocols. In order to handle different wearable devices, it is necessary to interact with them through different communication protocols, such as Bluetooth, WiFi, ZigBee, or Serial. Therefore, a software component for each specific protocol (labeled as 2) should be implemented to access the information provided by the device. This information is supplied in the form of arrays of bytes. All the wearable devices with the same communication protocol are handled through the same component.

(3) Payload Processing. Once a wearable device has sent the information, it is necessary to process it to ensure that is correct as well as obtain the payload. The payload processing component is responsible for transforming the payload represented in arrays of bytes into understandable information using the specifications of the device model. Moreover, the component should ensure the Quality of Service (QoS). The payload processing component (labeled as 3) is depicted in the flowchart presented in Figure 8.

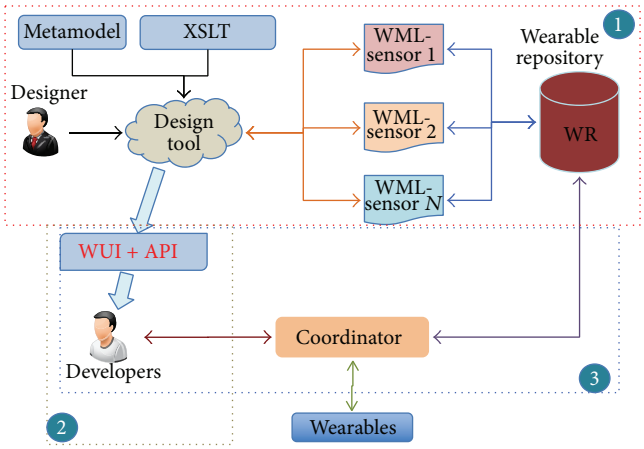


Figure 9: Process to use a new device based on the MDWSD approach.

(4) Orchestrator. This component is responsible for consuming the services provided by other components and orchestrating them to provide the proper information to developers through its API. The API can be also used by developers to interact with the coordinator, in order to define the device to be used, the value of the variables that should be retrieved from the device, turning on/off a device, or enabling/disabling notifications.

In order to handle the asynchronous interaction with devices, the coordinator uses a custom software component called *pipe*. A pipe is an event-driven software element responsible for transmitting the information from the devices to the developers (e.g., heart rate, time between two beats, use position, and contextual information) through the coordinator module. In this manner, when a developer uses this module to handle a device, it returns a pipe (one pipe per device) to the developer. When the device sends

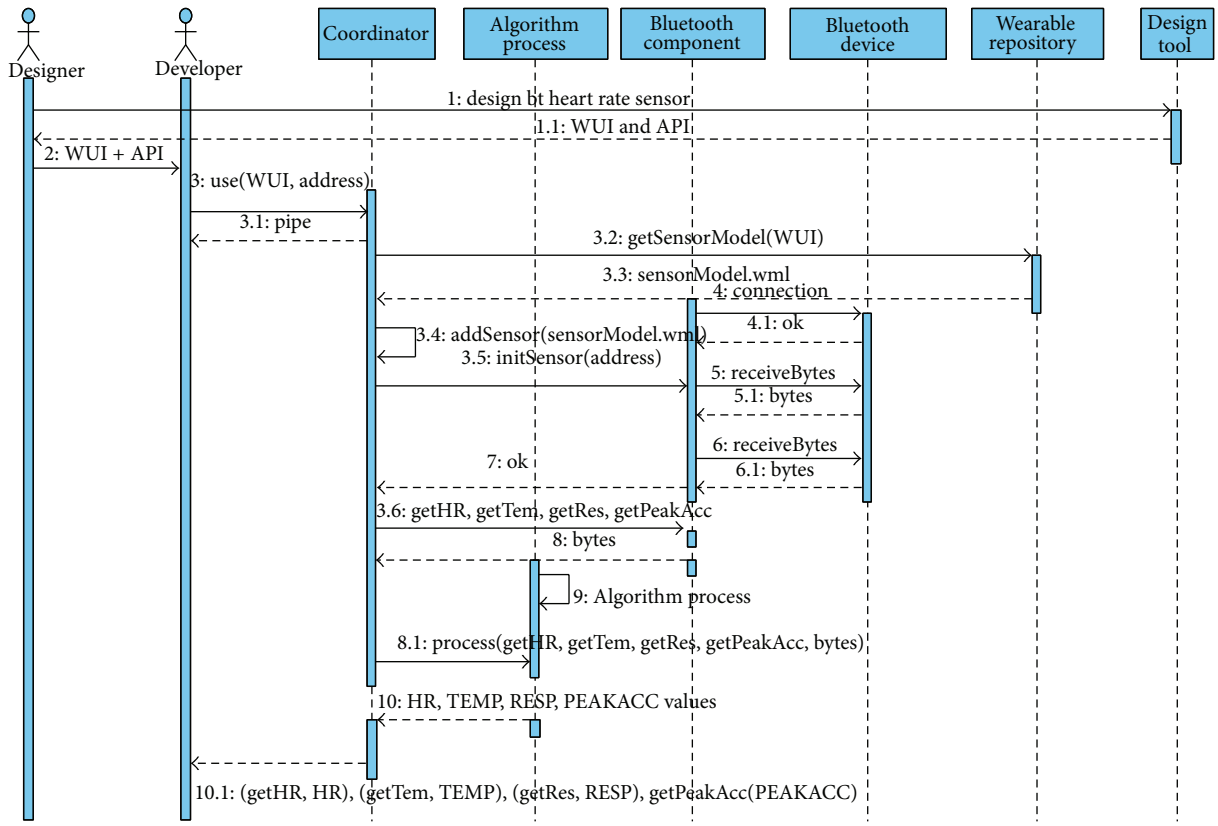


Figure 10: Sequence diagram of the process to use a new multifunctional device.

the information, which is processed by the coordinator using the process of the Figure 8, an event/callback is sent to the developer notifying him/her about relevant information related to a specific API (defined in the device model).

4.5. Using a Wearable Device. The approach proposed in this paper allows addressing the use of any device, regardless of its technical features and the lack of standardized access to the device information. This approach has been conceived to improve the use of new wearable devices in wearable computing contexts and facilitate and support part of the development tasks, thereby reducing the implementation effort (time and cost) and risk. In order to achieve this goal, the metamodel and the coordinator presented before are used in this process. Particularly, the use of a new wearable device supported by this approach consists of three stages, and it is represented in Figure 9.

4.5.1. Design. The designer uses the design tool to come up with a device model API, compliant to the device features or datasheet. During this stage, the designer uses different device documentation to model the device interaction and defines a custom API to interact with the device in order to obtain the data provided/sensed by the device. The design tool uses the metamodel and device model generation process to produce a device model, which is stored in a wearable model repository (WR).

4.5.2. API Release. Once the design is finished, the design tool provides the API defined by the designer and also a unique identifier for the device model (i.e., the WUI). Thus, the developers are then able to access this API.

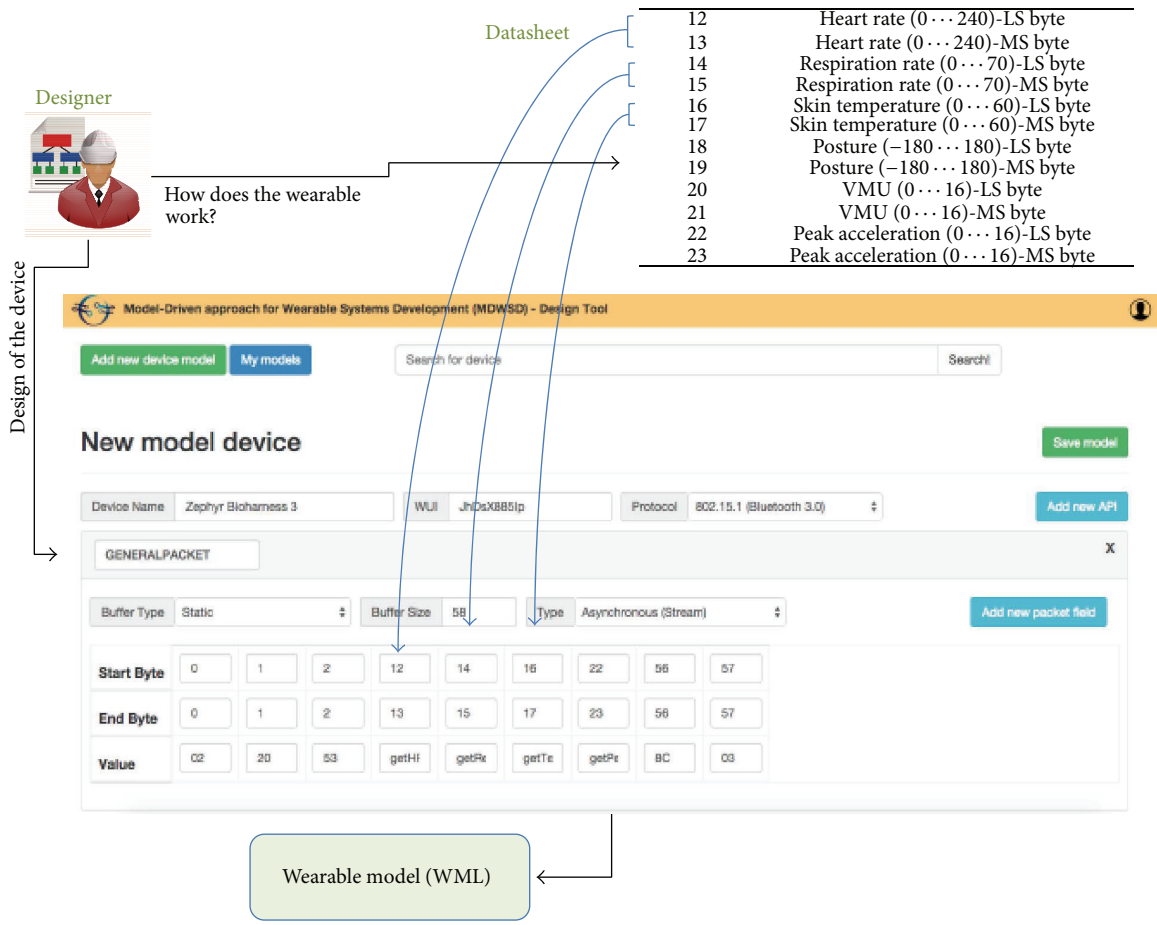
4.5.3. Interaction with the Device. With the API, the WUI, and using the coordinator, the developers are able to easily handle the device through the API and thus obtain the information in an easy-to-understand format, without the need to know about the device's internal workings or low-level details.

This approach entails the following advantages:

(i) The developer can use any wearable device with no knowledge of communication protocols or the device workings.

(ii) The possibility of defining a unique customized API for each device standardizes the access to the information, because all the users/developers utilize the device in the same way.

(iii) If there is a bug in the device model, the designer can redefine it. If this redesign does not involve modifying the API, the software coordinator uses this new model in a transparent way for developers. Device models are platform-independent and they can be used by any coordinator, independently of the



12	Heart rate (0 · · · 240)-LS byte
13	Heart rate (0 · · · 240)-MS byte
14	Respiration rate (0 · · · 70)-LS byte
15	Respiration rate (0 · · · 70)-MS byte
16	Skin temperature (0 · · · 60)-LS byte
17	Skin temperature (0 · · · 60)-MS byte
18	Posture (−180 · · · 180)-LS byte
19	Posture (−180 · · · 180)-MS byte
20	VMU (0 · · · 16)-LS byte
21	VMU (0 · · · 16)-MS byte
22	Peak acceleration (0 · · · 16)-LS byte
23	Peak acceleration (0 · · · 16)-MS byte

Figure 11: Device model design process.

particular platform in which they are used (e.g., iOS, Android, and Windows).

(iv) Each wearable device model is designed only once but used many times.

(v) The metamodel and coordinator are designed to be open and extendable. The metamodel is extendable to ensure that it adapts to new technologies or requirements. Changes in the metamodel entail changes in the new wearable model, and usually in the coordinator. For example, although a new communication protocol could be supported in the model, it should be also implemented in the coordinator.

This approach ensures interaction and use. Firstly, any developer with no expert knowledge can quickly use any device by simply using the API. Secondly, the coordinator should be able to detect new devices, access the repository to find its model, and incorporate it automatically.

5. Case Study

In order to illustrate the process of using wearable devices in a software application, Figure 10 shows a sequence diagram that represents the use of a sensor, from the design to the usage stage. The sample device is a Bluetooth multifunctional sensor (heart rate, temperature, accelerometer, etc.) with the specified API *getHR* to obtain the heart rate value, *getResp* to get the respiration rate, *getTemp* for the body temperature, and *getPeakAcc* to determine the linear acceleration.

The different steps to use a wearable, from scratch, are as follows:

(1) The designer/expert reads about how the sensor works using the datasheet or other information provided by the vendor.

(2) The designer, knowing how the sensor works, uses the design tool to create the sensor model according to its features and defines a high-level API for such a device. Figure 11 represents a designer using the design tool to model the wearable from the datasheet where, for example, *getHR* is the tag defined to represent the heart value information inside of the payload or *getTemp* is to represent the body temperature.

(3) Once the designer concludes, the design tool generates the sensor model using the device model generation process. Algorithm 1 represents an excerpt of WML file with the sensor model considered in this case study.

```
<?xml version="1.0" encoding="UTF-8"?>
<sensor>
  <id>JhDsX885Ip</id>
  <protocol>802.15.1</protocol>
  <name>Zephyr Bioharness 3</name>
  <api>
    <apisen>
      <id>GENERALPACKET</id>
      <type>0</type>
      <maxbuffersize>58</maxbuffersize>
      <buffertype>1</buffertype>
      <pivot>0x02</pivot>
      <payload>
        <start>3</start>
      </payload>
      <fields>
        <field>
          <id>idop</id>
          <value>0x20</value>
          <start>1</start>
          <offset>0</offset>
        </field>
        <field>
          <id>length</id>
          <value>0x53</value>
          <start>2</start>
          <offset>0</offset>
        </field>
      </fields>
      <apival>
        <id>getHR</id>
        <start>12</start>
        <end>13</end>
        <datatype>0</datatype>
      </apival>
      <apival>
        <id>getResp</id>
        <start>14</start>
        <end>15</end>
        <datatype>0</datatype>
      </apival>
      <apival>
        <id>getTemp</id>
        <start>16</start>
        <end>17</end>
        <datatype>0</datatype>
      </apival>
      <apival>
        <id>getPeakAcc</id>
        <start>22</start>
        <end>23</end>
        <datatype>0</datatype>
      </apival>
    </apisen>
  </api>
</sensor>
```

Algorithm 1: Device model represented in WML.

(4) The design tool also releases the WUI (*JhDsX885Ip*) and the API (*getHR, getResp, getTemp,* and *getPeakAcc*) to the developers. These tags/identifiers correspond to those required for retrieving the heart rate, respiration rate, temperature, and linear acceleration through the coordinator.

(5) The developer adds the coordinator (as a software component, e.g., a .jar file in Android) into the project.

(6) The developer uses the coordinator in conjunction with the WUI and the sensor API to interact with the sensor, as shown in Algorithm 2. Particularly, the developer can use the WUI to request the WML file through the coordinator. The coordinator, in turn, uses the WR component to obtain the sensor model and manages this model as an internal software element to handle the sensor through a pipe (class *zPipe* in Algorithm 2).

In the application side, once the wearable device has been detected, connected, and synchronized, the coordinator starts interacting with the sensor by means of the components and processes the information according to the payload processing component.

When the coordinator has a correct and understandable value, according to the check error criteria specified in the sensor model, it launches an event in the corresponding pipe with the sensor information in a human-understandable format (e.g., *getHr* returns 80 beats per minute and *getTemp* informs 37.4 Celsius). The coordinator returns this information to the developer, who handles it in the proper way; for example, he/she shows it on the display, processes it, or stores it in a database.

6. Conclusions and Future Work

Wearable devices are constantly being launched in many different domains, such as healthcare, entertainment, wellness, and sport training. The architecture, operation modes, data streams, and representation formats of these devices are designed in a different way by vendors' engineers. This happens because of the absence of standards and reference models in the field.

This situation results in heterogeneous possibilities of interacting with these devices. Considering that designers are responsible for defining the manner in which sensor information is accessed through a custom and well-defined API, the heterogeneity and lack of API standardization hamper the use and integration of these devices. Therefore, developers must write source code mostly from scratch and deal with low-level details about the wearable technical features and interfaces, increasing the development effort and risk.

In order to help address these challenges, this proposal presents a model-driven approach that abstracts the developers from these challenges, by providing them device models and abstract APIs through which they can interact with the wearable devices in a simple way.

```
zCom.init();
zCom.setContext(getApplicationContext());
String WUI="jhDsX885Ip";
String MAC="00:22:D0:02:4C:49";
zPipe pipe=new zPipe(){
    public void handleMessage(Message msg){
        super.handleMessage(msg);
        if(msg.getData().containskey("getHR")){
            Object hr=msg.getData().get("getHR");
            //manage heart rate value
        }
        else if(msg.getData().containskey("getTemp")){
            object temp=msg.getData().get("getTemp");
            //manage skin temperature
        }
        else if(msg.getData().containskey("getResp")){
            Object resp=msg.getData().get("getResp");
            //manage respiration rate
        }
        else if(msg.getData().containskey("getPeakAcc")){
            Object peak=msg.getData().get("getPeakAcc");
            //manage linear acceleration
        }
    }
};
zCom.addDevice(WUI, MAC, pipe);
```

Algorithm 2: Use of coordinator in Android platform.

The proposed approach, named model-driven wearable systems development (MDWSD), comprises five main components: a *wearable specification metamodel*, a *design tool*, a *device model generation process*, a *coordinator component,* and a *model-driven wearable integration process*.

The metamodel allows developers to define the specification of a wearable device, regardless of its vendor-specific features. It also permits the specification of a custom API to access sensor information. This metamodel has been designed to be extendable in order to cover the requirements and features of future-released devices.

The design tool supports the definition of wearable models, which represent their features and how to interact with them. The device model generation process is used by the design toolto produce instances of the metamodel (i.e., each sensor model). The coordinator is the software component that handles sensor models in order to facilitate the integration of new devices and allow the use of these sensors in a seamless and transparent manner. Finally, the model-driven wearable integration process is the component that allows the MDWSD approach components to be coordinated through a single process.

This paper also presents a case study to illustrate the development of a software application that gets several functionalities from a commercial wearable device, by using the MDWSD approach. As a future work, we plan to make this proposal deployable as an open software solution, so that it can be used by any developer community.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work has been partially funded by the Facultad de Educación, Economía y Tecnología de Ceuta under the "Contrato-Programa" of research for the period 2013–2015. This work has also been supported by Fondecyt (Chile), Grant no. 1150252. The authors would also like to acknowledge input from COST Action AAPELE (IC1303).

References

[1] S. Mann, "Smart clothing: the shift to wearable computing," *Communications of the ACM*, vol. 39, no. 8, pp. 23–24, 1996.

[2] D. Roggen, D. G. Perez, M. Fukumoto, and K. van Laerhoven, "ISWC 2013—wearables are here to stay," *IEEE Pervasive Computing*, vol. 13, no. 1, pp. 14–18, 2014.

[3] L. Gatzoulis and I. Iakovidis, "Wearable and portable eHealth systems," *IEEE Engineering in Medicine and Biology Magazine*, vol. 26, no. 5, pp. 51–56, 2007.

[4] T. Kleinberger, M. Becker, E. Ras, A. Holzinger, and P. Müller, "Ambient intelligence in assisted living: enable elderly people to handle future interfaces," in *Universal Access in Human-Computer Interaction. Ambient Interaction*, vol. 4555 of *Lecture*

Notes in Computer Science, pp. 103–112, Springer, Berlin, Germany, 2007.

[5] A. J. Jara, M. A. Zamora, and A. F. G. Skarmeta, "An Internet of things—based personal device for diabetes therapy management in ambient assisted living (AAL)," *Personal and Ubiquitous Computing*, vol. 15, no. 4, pp. 431–440, 2011.

[6] Y.-D. Lee and W.-Y. Chung, "Wireless sensor network based wearable smart shirt for ubiquitous health and activity monitoring," *Sensors and Actuators B: Chemical*, vol. 140, no. 2, pp. 390–395, 2009.

[7] C. Chen, A. Knoll, H. E. Wichmann, and A. Horsch, "A review of three-layer wireless body sensor network systems in healthcare for continuous monitoring," *Journal of Modern Internet of Things*, vol. 2, no. 3, 2013.

[8] Q. Zhang, Y. Su, and P. Yu, "Assisting an elderly with early dementia using wireless sensors data in smarter safer home," in *Service Science and Knowledge Innovation*, pp. 398–404, Springer, Berlin, Germany, 2014.

[9] T.-W. Jo, Y.-D. You, H. Choi, and H.-S. Kim, "A bluetooth-UPnP bridge for the wearable computing environment," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1200–1205, 2008.

[10] D. Carr, M. J. O'Grady, G. M. O'Hare, and R. Collier, "SIXTH: a middleware for supporting ubiquitous sensing in personal health monitoring," in *Wireless Mobile Communication and Healthcare*, pp. 421–428, Springer, Berlin, Germany, 2013.

[11] P. Castillejo, J.-F. Martinez, J. Rodriguez-Molina, and A. Cuerva, "Integration of wearable devices in a wireless sensor network for an E-health application," *IEEE Wireless Communications*, vol. 20, no. 4, pp. 38–49, 2013.

[12] S. Hadim and N. Mohamed, "Middleware: middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, p. 1, 2006.

[13] C. Rodriguez-Dominguez, T. Ruiz-Lopez, J. Luis Garrido, M. Noguera, and K. Benghazi, "A model-driven approach to service composition on the basis of the specification of BPMN choreographies," *Computer Systems Science and Engineering*, vol. 30, no. 1, pp. 69–77, 2015.

[14] B. Akbal-Delibas, P. Boonma, and J. Suzuki, "Extensible and precise modeling for wireless sensor networks," in *Information Systems: Modeling, Development, and Integration*, pp. 551–562, Springer, Berlin, Germany, 2009.

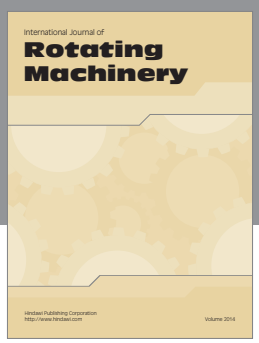
[15] F. Losilla, C. Vicente-Chicote, B. Álvarez, A. Iborra, and P. Sánchez, "Wireless sensor network application development: an architecture-centric MDE approach," in *Software Architecture*, pp. 179–194, Springer, Berlin, Germany, 2007.

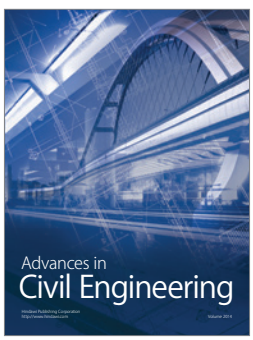
[16] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi, "SPINE: a domain-specific framework for rapid prototyping of WBSN applications," *Software: Practice and Experience*, vol. 41, no. 3, pp. 237–265, 2011.

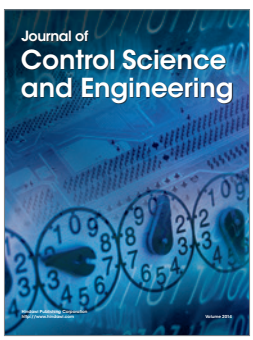
[17] G. Fortino, R. Giannantonio, R. Gravina, P. Kuryloski, and R. Jafari, "Enabling effective programming and flexible management of efficient body sensor network applications," *IEEE Transactions on Human-Machine Systems*, vol. 43, no. 1, pp. 115–133, 2013.

[18] S. Mann, "Wearable computing: a first step toward personal imaging," *Computer*, vol. 30, no. 2, pp. 25–32, 1997.

[19] J. Clark, P. Danielsen, B. Labs et al., *XSL Transformations (XSLT)*, World Wide Web Consortium (W3C), 1999, http://www.w3.org/TR/xslt.







Hindawi
Submit your manuscripts at
http://www.hindawi.com