Author/s:
Oliveira, EA;Kirley, M;Fonseca, JCB;Gama, K

Title:
Device Nimbus: An Intelligent Middleware for Smarter Services for Health and Fitness

Date:
2015-01-01

Citation:
Oliveira, E. A., Kirley, M., Fonseca, J. C. B. & Gama, K. (2015). Device Nimbus: An Intelligent Middleware for Smarter Services for Health and Fitness. International Journal of Distributed Sensor Networks, 2015 (8), https://doi.org/10.1155/2015/454626.

Persistent Link:
https://hdl.handle.net/11343/56503

Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 454626, 16 pages
http://dx.doi.org/10.1155/2015/454626

Research Article

Device Nimbus: An Intelligent Middleware for Smarter Services for Health and Fitness

Eduardo Araujo Oliveira,[1] Michael Kirley,[1] Jorge C. B. Fonseca,[2] and Kiev Gama[2]

[1]*Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia*
[2]*Computer Centre, Federal University of Pernambuco, Recife, PE, Brazil*

Correspondence should be addressed to Eduardo Araujo Oliveira; agogear@gmail.com

Received 16 December 2014; Accepted 18 March 2015

Academic Editor: Myung S. Yoo

The world is experiencing an unprecedented explosion in the number of smart devices and mobile apps that are available to users. In the health and fitness domains, many of the devices and technologies in the market place are restricted to proprietary platforms, typically working in isolation with fixed hardware settings. Consequently, an important challenge is to investigate techniques for combining and analyzing data encapsulated by these ubiquitous technologies. In this paper, we introduce an intelligent middleware—Device Nimbus—to meet this challenge. The middleware supports the integration of distributed and heterogeneous mobile sensor data, enabling both context and predictive analysis. We describe a minimum viable product of Device Nimbus and report the results of preliminary tests spanning multiple data sources focused on fitness apps, illustrating the efficacy of the middleware.

1. Introduction

In mirroring the growth and ubiquitous nature of smart and mobile devices, wearable health-monitoring systems (WHMS) have attracted a great deal of attention from both the research community and industry [1–3]. Fitness and/or wellness apps and devices are now widely accepted. Such apps, and in some cases even implantable devices, can be used to collect and manage information related to a users' health status. These biosensors are capable of measuring significant physiological parameters such as heart rate, blood pressure, body and skin temperature, oxygen saturation, respiration rate, and electrocardiogram [4]. Figure 1 provides a schematic representation of the current state of play in this domain, illustrating a number of invisible health and wearable devices.

Current ICT solutions targeting smart devices for health care have typically focused on collecting data from an individual sensor/gadget data (not from different sources), ending up as control systems for monitoring and helping users. Many of the nominated devices and technologies are not fully integrated; they have fixed hardware settings and typically they function in isolation.

The introduction of middleware, featuring context awareness and predicative analysis in a health care environment, offers many opportunities. However, the development of such a middleware poses many challenges. The middleware must enable the sharing and integration of information between various third party apps. This task would involve a large number of independent, autonomous, heterogeneous, and interacting subsystems, sensors/gadgets, and algorithms. Importantly, the middleware should provide support for data collection and be able to integrate with other fitness/wellness devices and apps in a mobile computing environment [5]. Context awareness in this domain may be thought of as the ability to define what is relevant to a given user at a specific point in time. The use of context allows the system to filter and disseminate useful information and adapt its services to the particular needs of the user, providing recommendations, alerts, and changes in interfaces (allowing greater flexibility and ease of use) [6]. This, in turn, would significantly advance predictive analysis and possibly lead to adaptive user behaviour.

This leads to the key research question addressed in this paper: *How can personalized healthcare support be provided*

Figure 1: Representative small wearable and/or invisible devices suitable for use with a health focus.

to users, based on data collected from heterogeneous mobile devices, sensors, and resources on the Web? This question can be broken down further. (i) What is the most effective way to integrate and organize multiple heterogeneous, autonomous subsystems, and sensors data? (ii) How can data mining techniques be used to provide personalized feedback and, in a health care focus, provide early warning signs for serious illnesses, based on users' particular lifestyles and needs?

In this paper, we introduce a software-based intelligent middleware, which we call Device Nimbus, to address these questions. Device Nimbus supports context and predictive analysis focusing on the integration of mobile sensor data for smarter services. The proposal encompasses three main components: *data collectors*, *data integration*, and *intelligent modules*. The resulting solution addresses the requirements of the target scenario by providing context awareness, adaptivity, flexibility, and extensibility to the proposed middleware. Device Nimbus is an innovative infrastructure that includes heterogeneity (e.g., different objects, sensors, protocols, and applications), dynamic environments (e.g., arrival and departure of devices, objects, and sensors), analysis (e.g., contents personalization, recommendations, and prediction), and an evolutionary framework (e.g., support for new protocols, devices, and sensors).

To demonstrate the efficacy of the proposed middleware, we have developed a minimum viable product (MVP) for Device Nimbus with limited functionality. A series of tests were designed and carried out to evaluate the performance of Device Nimbus using different data sources as a proof of concept in a fitness/wellness domain. The results clearly show that the Device Nimbus met the expectations of collecting, integrating, and analysing data.

This remainder of the paper is organized as follows. Section 2 provided a comprehensive review of existing software platforms for health and fitness related domains in the context of mobile devices and technologies. Section 3 introduces the Device Nimbus architecture in detail. Section 4 discusses the developed Device Nimbus MVP. Section 5 discusses preliminary tests used to demonstrate the efficacy of the middleware. Section 6 provides conclusions and future work opportunities.

2. Related Work

In this section, we provide a review of the related work focusing on an analysis of mobile health and fitness apps and services before describing recent middleware contributions. The section concludes with a discussion of work related to context aware middleware.

2.1. WHMS/Mobile Health and Fitness Apps. Several sensing technologies such as electrocardiogram (ECG), blood pressure (systolic and diastolic), respiration rate, oxygen saturation, heart rate, blood glucose, body movements, and electromyogram (EMG) have been developed and provided by different companies (see list below). These sensing technologies typically use different types of sensors, such as chest electrodes, arm cuff-based monitor, temperature probe, and pulse oximeter. Unfortunately, these technologies are typically closed solutions. They are not designed to support easy integration with alterative platforms and legacy systems (hospitals and clinics, e.g.). Generally, they are not extendible by third parties and do not predict or recommend actions based on users behaviours and needs.

LifeGuard [7] is a WHMS monitoring system for space and terrestrial applications. LifeGuard is capable of measuring two ECG leads, respiration rate, heart rate, oxygen saturation, body temperature, blood pressure, and body movement. LifeGuard was tested at extreme environments including tests with the ability of satellite transmission of collected data. The authors obtained great results indicating acceptable accuracy for the collected data and real-time transmission to remote servers.

Researchers in Harvard University have developed Codeblue [8], a medical sensor network platform for multipatient monitoring environments. Codeblue is based on the ZigBee compliant MicaZ and Telos motes. The solution was designed with biosensor boards for EMG, pulse oximetry, three-lead ECG, and body motion. The CodeBlue project also addressed issues of reliable communication between medical sensors, multiple receivers (PDAs carried by doctors and nurses), and various high data rates.

Wood and colleagues [9] presented the ALARM-NET, a residential monitoring network for pervasive and adaptive healthcare. The ALARM-NET works are based on monitoring environmental and physiological data of individuals, through three different sensors, in their residences for collecting, storage, and analysing data in a remote database. The sensors maintain connections for long-term data transmission and query periods with the remote database. In this way, patients can be monitored constantly.

MyHeart [10] is a smartphone-based wearable platform, capable of monitoring the user's ECG signal in real time via a wireless ECG sensor. MyHeart uses an artificial neural network—with machine learning strategies—to adapt the system to the individual user's physiological conditions, which can possibly result in more accurate classification of ECG patterns. LifeShirt [11], from VivoMetrics, is a washable lightweight vest that includes respiratory rate sensors, one-lead ECG for heart rate measurement, and an accelerometer for activity monitoring.

Ren et al. [12] present initial results from a portable system capable of measuring phonocardiography (PCG), electrocardiography, and body temperature. The developed system consists of a capacitor-type microphone inserted on a stethoscope for PCG detection, a three-lead ECG, a temperature sensor, a measuring board including a CPU, a Bluetooth transceiver, and an A/D module and a PDA with an external memory unit [4]. LiveNet, developed by The Media Laboratory of MIT, is a distributed mobile platform aiming at long-term health monitoring applications with real-time data processing and streaming and context classification [13]. LiveNet's main focus is in the interpretation of real-time contextual data from a physiological board with ECG, EMG, accelerometer, and galvanic skin conductance sensors.

Recently, companies such as Apple, Google, and Samsung have already released smartwatches. The idea behind most of these watches is their capabilities as a comprehensive health and fitness watch. Digital application stores, like Google Play and Apple Store, have now thousands of health apps that get connected with different devices such as smartphones, smartwatches, and other mobile/wearable devices and have different goals.

Specifically in health domain, Apple's HealthKit system is being billed as a unifying force that ties the enormous amount of health-related apps on the App Store together. However, Apple's health is focused on data collection, not interpretation. Microsoft and Samsung, just to give more examples, have also their own devices and cloud platforms to collect and store data. Microsoft has a cloud-based service that helps users' live healthier by providing actionable insights based on data gathered from the fitness devices and apps that they use every day. Microsoft Health cloud monitors activities by tracking devices like the new Microsoft Band, smart watches, and mobile phones plus services like RunKeeper or MyFitnessPal. Using this fitness data and their Intelligence Engine in the cloud, Microsoft Health provides valuable and personal insights so users can reach their fitness goals [23].

SAMI (Samsung Architecture Multimedia Interactions) [24] is the Samsung common platform that will try to bring data from different devices together, especially in health domain. Samsung has partnered with IMEC and UCSF to fulfill a common mission to have sensors that will allow users to know more about their body and track their every move. To do this, Samsung started with a wearable device, called the SimBand [25]. SimBand is the Samsung concept design about health and wearable tech and into new territory, where blood flow, hydration level, skin temperature, and more are being tracked constantly. Samsung has also released Samsung Digital Health SDK [26], which helps developers to make better health applications for Samsung devices.

The combination of sensors, devices, and systems from different modalities and standards makes it necessary to develop hardware independent software solutions for efficient application development [27, 28]. In this context, there are numerous studies focusing on middleware/platform design [29–31]. Middleware can help health sensor/mobile networks to manage their inherent complexity and heterogeneity. The idea, according to [32], is to isolate commons behaviour that can be reused by several applications and to encapsulate it as system services. In this way, multiple sensors and applications can be supported easily.

2.2. Platforms and Middleware. Many of the existing ICT solutions for smart health device are proprietary solutions provided by large software and services vendors, such as IBM [33], Microsoft [34], Google [35], Samsung [36], and Apple [37]. Typically, these solutions are designed as a unified, distributed, and real-time control platform, adding cloud computing, sensing, simulation, analysis services, and applications. Integrating data from these devices is a significant challenge.

As a way of avoiding proprietary solutions, the Open Health Tools [38] was created as an open source community with a vision of enabling an ecosystem, where members of Health and IT professions collaborate. This collaboration is based on building interoperable systems (platform) that enable patients and their care providers to have access to vital and reliable information at the time and place it as needed. However, interoperability benefits are highly dispersed across many stakeholders, and early adopters are penalized by

negative network externalities and first-mover disadvantages, for example, faced barriers and challenges that have resulted in partial success, slow progress, and outright failure [39].

Nimbits [14] is a platform for connecting people, sensors, and software to the cloud and one another. Nimbits provides web services that can process time and geo stamped data that trigger calculations, statistics, alerts, and more. The platform has an Open Source Java Library that provides an easy way to develop JAVA, Web, and Android solutions that use a Nimbits Server as a backend platform. Etherios [40] is a machine-to-machine (M2M) platform-as-a-service that manage different connected devices from one user interface. It works with 3G, Wifi, Zigbee, and 4G networks. The platform is not free and offers its services based on different pricing packs.

ThingSpeak [15] is another open source IoT application and has an open API to store and retrieve data from device assets or things via LAN or using HTTP over the Internet. With this platform, location tracking applications, sensor logging applications, and social network of device assets with proper update of its status can be created. ThingSpeak allows numeric data processing like averaging, time-scaling, rounding, median, summing, and so forth to store and retrieve the numeric and alphanumeric data. ThingSpeak application features a JavaScript-based charts, read/write API key management, and a time-zone management.

The WSO2 [16] platform is a middleware platform that is 100% open source. Users can explore and experiment this platform with no licensing costs. Most importantly, users can download and play with the same version that runs in production at no cost. WSO2 not only uses an open-source license but also follows an open development process, which customers can observe and provide input into. The WSO2 platform is made up of over 25 products covering all major categories from integration and API management to identity and mobile. All products are built using a single code base; therefore, they work seamlessly with each other so less engineering resources are needed when integrating them.

Although other initiatives like Magic Broker [17, 18], SOFIA [19], and Xively [20] provide part of their infrastructure as free and open source software, they describe system architectures that are just centered on the smart environments concept, for creating interacting objects ecosystems (sensors, devices, appliances, and embedded systems). They are not concerned with simultaneously dealing with different vertical axes (e.g., health, weather, fitness, and feeding) and with closed or proprietary wearable solutions. In addition, all of those experimental efforts are custom-tailored solutions whose components or subsystems were not modelled as interchangeable parts, nor conceived to be integrated with other subsystems. In health domain, a big challenge is to collect, integrate, and analyse data from proprietary solutions (mobile devices from Nike, Garmin, Google, Apple, and others) and from lots of proprietary different sensors and track the same user based in lots of different input devices to provide them contextual feedbacks/suggestions, based on their particular needs.

2.3. Context-Aware and Predictive Health Middleware. Focusing on heterogeneity of data and context-aware solutions, Kang and colleagues [21] proposed a wearable context aware system for ubiquitous healthcare. Two types of wearable sensors provide the context for the system: a watch type sensor system and a chest belt type sensor system. The contextual system combines data from these sensors and operates with an OSGi context aware framework as a Java-based component service middleware on a Zigbee network.

MediAlly is a middleware for supporting energy-efficient, long-term remote health monitoring. The key to MediAlly's energy efficient operations lies in the adoption of an Activity Triggered Deep Monitoring (ATDM) paradigm, where data collection episodes are triggered only when the subject is determined to possess a specified context [22]. According to [22], MediAlly supports the on-demand collection of contextual provenance using a novel low-overhead provenance collection subsystem, configured using an application-defined context composition graph. The resulting provenance stream provides valuable insight while interpreting the "episodic" sensor data streams [22].

Gong et al. [41] propose an intelligent middleware with two-layer knowledge-based architecture, which can integrate heterogeneous health care applications dynamically. In the first data sources integration layer, standard-based health care ontology and global API metadata are achieved, which make authorized health care applications "understand" and operate heterogeneous data sources automatically and transparently. Based on this unified information, in the second application integration layer, configurable standard-based application workflow integration facilitates the communication and interoperation between the disparate health care applications.

The health middlewares listed above are (to some extent) able to identify patterns indicating or predicting a change in the health status of patients. However, they are not able to collect data from heterogeneous and proprietary wearable devices and, significantly, they are not open source or free.

Based on the requirement analysis described in Table 1, we position Device Nimbus' against the state-of-the-art. It will become evident, from Section 3, that the proposal covers a gap in current technology, supporting all the presented requirements, not previously contemplated by the analysed technology.

3. Device Nimbus Middleware

In this section, we describe the rationale behind, and design of, the proposed middleware. Device Nimbus provides a stepping stone towards solving many of the problems currently found in the health domain such as tracking users based on their use of different mobile devices/sensors, providing personalized feedback, and the possibility of connecting with clinics and hospitals.

3.1. Conceptual View. The proposed middleware, conceptually presented in Figure 2, provides a mechanism to integrate mobile devices and health sensors. From the functional viewpoint the proposed context-aware system can be represented as a layered middleware consisting of sensors, raw data retrieval, preprocessing, storage or management, and an application layer (Figure 3). This approach allows data to be

Table 1: State-of-the-art comparison of health systems/middleware requirements.

References versus requirements	Context	Scalability	Extensibility	Flexibility	Lightweight	Standard compliance	Resilience
LifeGuard [7]	Focus in hardware – not a platform. (none of the requirements)						
Codeblue [8]	—	X	X	X	—	X	X
ALARM-NET [9]	X	X	X	X	—	X	—
MyHeart [10]	—	—	—	—	—	—	X (data storage)
LifeShirt [11]	No details about the architecture/implementation/system provided in the paper.						
Ren et al. [12]	Not a platform – simple app to receive sensors data. (none of the requirements)						
LiveNet [13]	X	X	X	X	X	—	X
Nimbits [14]	—	X	—	X	—	X	—
ThingSpeak [15]	—	X	—	X	X	X	—
WSO2 [16]	—	X	—	X	X	X	? (not clear)
Magic Broker [17, 18]	X	X	—	X	X (as a state concept)	X	—
SOFIA [19]	X	X	—	X	—	X	X
Xively [20]	—	X	X	X	X	X	? (not clear)
Kang and colleagues [21]	X	—	X	X	? (not clear)	X	? (not clear)
MediAlly [22]	X	—	X	X	X	X	—

Figure 2: A conceptual intelligent middleware: Device Nimbus. This model is based on the integration of mobile sensor data for Smarter Services focused specifically on fitness sensors/device.

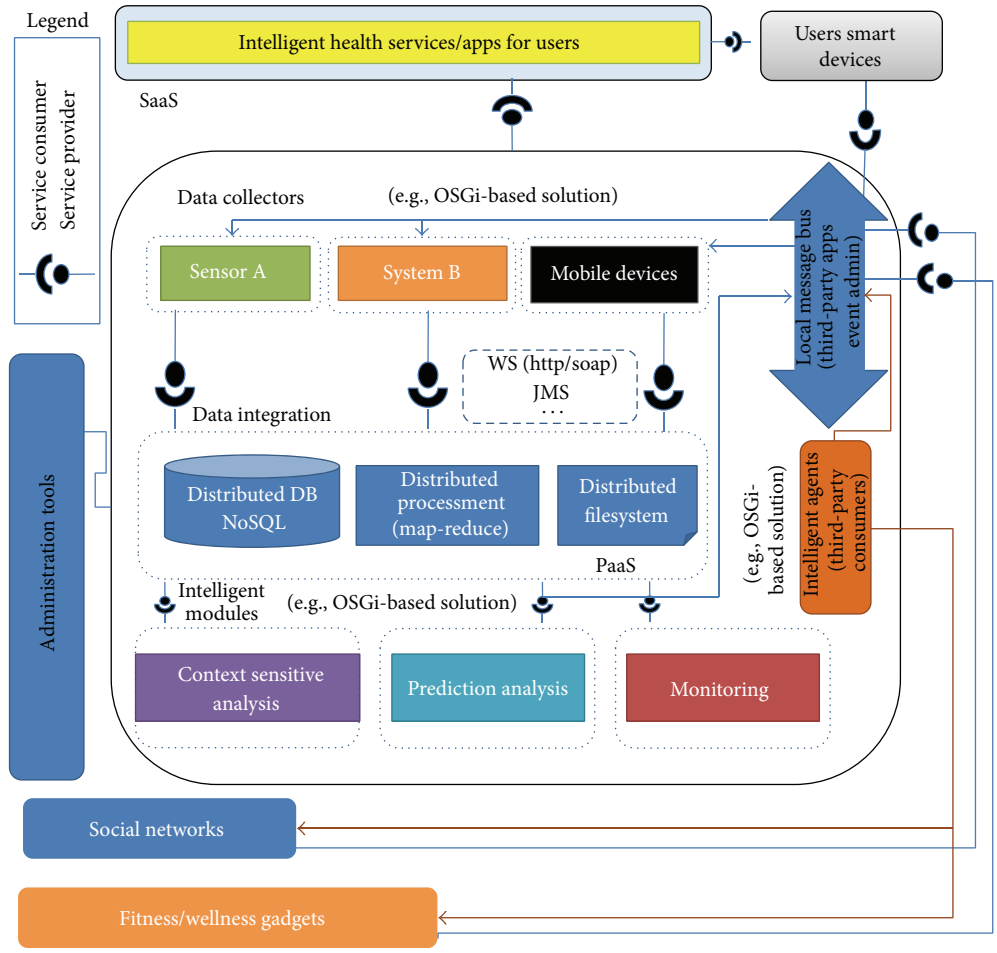

Figure 3: The high-level view of the Device Nimbus architecture.

integrated from smart devices in terms of context awareness: for text mining, sentiment analysis, node classification in the context of this application domain. Additionally, the middleware can be used as a repository/source of data for big players, such as the Samsung Architecture Multimodal Interactions (SAMI) cloud service. Individual users will be able to automatically convert units from smart devices and export and send data/reports to the physicians, health groups, hospitals, and even social networks.

3.2. Technical Specifications. Middleware typically operates in an environment, which includes heterogeneous computer architectures, operating systems, network protocols, devices, and databases [42]. Consequently, a key challenge is how to integrate such environments.

A service-oriented architecture (SOA) [43] brings loose coupling and flexibility to applications and allows the seamless integration of heterogeneous platforms and applications [44]. Dynamic platforms such as the OSGi service platform, which also applies service-oriented principles, provide the ability of dynamic evolution of code (i.e., component updates at application runtime).

OSGi specifications have been implemented in several certified platforms that have been adopted by a wide range of device vendors to empower their products with OSGi as a hosting platform for wired and wireless SOA oriented applications.

Examples of these platforms are: Equinox [45], Knopflerfish [46], Apache Felix [47] and Concierge OSGi [48]. OSGi-compliant free open source products such as ServiceMix [49] and JBoss Fuse [50], provide many patterns for facilitating the integration of new systems and components that need to exchange data. As a software architectural model for distributed computing, they represent a specialty variant of the more general client server model and promote agility and flexibility with regards to communication between applications. Its primary use is in Enterprise Application Integration (EAI) of heterogeneous and complex landscapes. For these reasons, SOC seemed as a good way to tackle the platform issues.

The proposed Device Nimbus middleware is service-oriented, OSGi-based. A high-level technical outline of the middleware, in terms of building blocks and their interactions, is illustrated in Figure 3. In the following subsections,

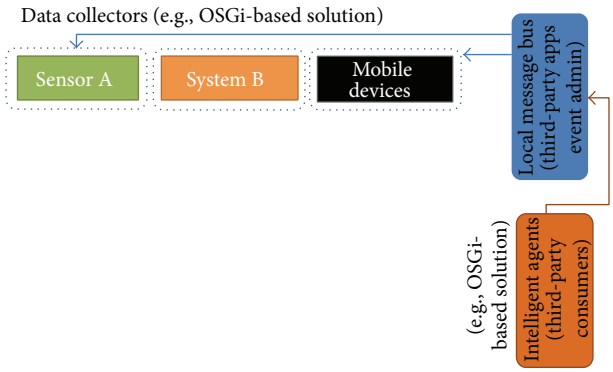

Figure 4: Data collectors: Device Nimbus architecture (extracted from Figure 3).

we describe each of the key blocks (com-ponents), data collectors, data integration, and intelligent modules, in detail.

Data Collectors. The data collectors are the part of the middleware that is responsible for collecting data from different devices and sensors. The objective of this layer is to allow for the easy integration of subsystems that collect data from the external world. This data collection can be achieved via sensors that provide data, through people that feed the system with data (e.g., through mobile devices), or through systems that are able to gather nonstructured data from the Web (e.g., social networks, web pages, and documents). A key aspect is that data from this layer may come from different domains (e.g., fitness, illness, and weather), which ultimately will allow the data integration layer to extract cutting-edge information for supporting more advance smart services.

Due to the dynamic nature of sensors/systems that may enter or leave the middleware in an unpredictable way, we decided to use a dynamic services platform in order to bring SOC to this layer. Both dynamicity and flexibility that allow the evolution of components and services at runtime, among other reasons, made the OSGi the platform of choice for constructing this layer as depicted in Figure 3 and in details in Figure 4. Built on top, an ESB is responsible for receiving data from different sensors and systems and delivering them into the middleware.

Efficient communication techniques can be applied in order to avoid system overheads, thus enabling easier scalability. For instance, depending on the number of system nodes and the chosen communication style the data traffic may be slow and affect the efficiency of the mediation chain. In such cases, with the help of the monitoring and adaptation layer, the system should be reconfigured in order to use a communication style that performs better upon the detected conditions. Quality-of-service (QoS) information will be fundamental for enabling the detection of such conditions.

By taking into account the current advances in ubiquitous computing, certain systems may play the part of a monitoring platform in an opportunistic nature (e.g., participant nodes are systems hosted in mobile devices whose connection may come and go). In order to easily accommodate such a changing architecture, the middleware must be flexible enough so it can be aware that certain nodes do not necessarily need to be fixed, since they are not broken and also must be able to cope with the unavailability of such nodes that are intermittent in the architecture. There must be a means for caching or resending data in case these nodes are not able to communicate with the middleware. Intermediary nodes in the architecture may also play an important role by filtering and aggregating data, as well as helping other opportunistic nodes (e.g., a mobile device user that prefers using WiFi hotspots when available instead of paying 3G access) that may want to participate as data collection nodes.

Architectures that support runtime evolution (referred to as dynamic adaptation) also allow behaviour and functionality to evolve over time. This is typically achieved through (a) components (or subsystems) that can be added, updated, or removed without interrupting the overall system execution or (b) by reconfiguring system attributes that indicate the active or enabled behaviours of the system. In the first case, the possibility of inserting modules during execution depends on units that can be easily pluggable in the architecture. A common programming interface provided by the middleware will allow third party systems to provide the required facets so the systems can be easily inserted in the architecture of the running system. In the second case, in order to support the reconfiguration of attributes during execution, the system needs to supply a runtime model representing the system state and also provide interfaces or actuators that allow changing such model attributes. Changes to that model will reflect the actual system behaviour and state.

Data Integration. The second key block in the design is the part of the middleware that is responsible for persistence and data integration. The data collected from the data collection systems and persisted in an environment that relies, for example, on a cloud computing infrastructure for guaranteeing the provisioning of the necessary storage space. Moreover, the middleware is able to handle several communication protocols thanks to bridge mechanisms we provide: Java message service (JMS) and web services (HTTP/SOAP). Bridges to other protocols, such as XMPP, could be easily added even during execution. The ESB and the intelligent agent of the middleware manage the input data from the different devices in a consolidated NOSQL database (Figures 3 and 5).

A significant challenge when developing smart applications, as well as interoperability of distributed systems, is the design of techniques for the integration of distributed data on the Web and from sensors. Processing and analysing acquired data, associated with concepts of pervasive and ubiquitous computing, among others, supports smart applications and context-sensitive systems development. The proposed data integration module is developed for big data processing, classification, and organization to support the development of applications on top. Additionally, a service layer providing access to the processed data allows for the construction of

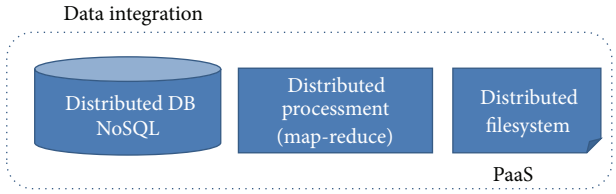

Figure 5: Data integration modules: Device Nimbus architecture (extracted from Figure 3).

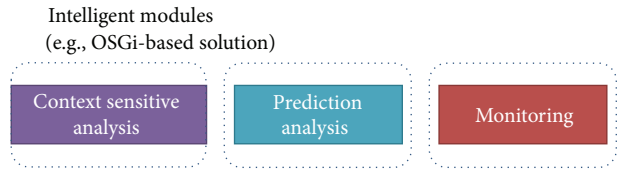

Figure 6: Intelligent modules: Device Nimbus architecture (extracted from Figure 3).

smart applications and services that reuse functionality of the platform.

Smart applications built on top of the designed service layer typically have to consider processed information dynamically to support users' activities. Thus, the monitoring process should occur continuously, comprehensively, and transparently. The quality of smart services is directly related to the quality of the information collected.

The data integration module can be developed with third party components and engines for processing the data and inferring information and knowledge from it. Mechanisms for data analysis and data mining must be used in this module. Any anomaly detection and failure prediction techniques will be enhanced by exploiting the monitored data and the supplementary operational data about the infrastructure produced through adaptive monitoring and run-time stimulation. This will rely on the identification of variables and patterns of the infrastructure's operational profile by employing data mining and machine learning techniques, including variable (feature) selection methods and regression analysis.

For failure prediction, instead of following the conventional techniques that just observe the operational state of the infrastructure, the proposed approach will heavily rely on the additional data produced by the controlled run-time stimulation.

Maintenance strategies can be selected, tuned, and applied at run time based on the results from monitoring and subsequent failure predictions and on their impact on the resilience of the system. The maintenance actions to be performed should be determined based on one or more variables correlated to the degradation of the system state. Maintenance actions of the same type, preplanned and with a given frequency, will be instantiated from preventive maintenance policies.

Intelligent Module. The third block in the proposed design is the part of the middleware that is responsible for data analysis. This component combines context aware capabilities and predictive analysis, using state-of the-art machine learning models. Due to the dynamic nature of artificial intelligent learning modules that may enter or leave the middleware (additional analysis new modules), the intelligent module was designed to be integrated with OSGi. The middleware considered dynamic services platform in order to bring SOC to this layer (Figures 3 and 6).

The intelligent module will be built on top of data integration layer. Device Nimbus middleware provides also an interface to help in configuring, monitoring, and managing the middleware and to get connected with third-party apps, as described below.

Administration Tools. The final components of the model that must be addressed are administration tools. The proposed middleware provides an array of administration tools that allow users configuring, monitoring, and managing of the subsystems (Figure 3). This basically includes (i) a system management console for visualizing the nodes that participate in the system's architecture (which may vary over time) at runtime and eventually reconfiguring system parameters on them and (ii) a tool for visualizing and configuring the system monitoring and adaptation policies. The goal is to provide a sort of administration view (i.e., a "control panel") for the people that will be in charge of the system administration. In order to be able to easily connect to the other systems, the developed tools will implement specific middleware connectors and probes for communicating with the systems that are being managed. The integrated systems must provide implementations of the connectors and probes according to proposed middleware's API.

The local message bus is based on OSGi's event admin and is used for sending information reader data. The event admin is an OSGi standard service that provides a mediator for the publisher-subscriber pattern.

Intelligent Healthcare Services. User interaction with the middleware is via intelligent health services/apps. Applications based on service-oriented computing will benefit from the middleware, which will provide many services on top of the data that is preprocessed (Figure 3). Examples of such applications are a command and control centre that visualizes the data and analytic information about what is being collected from the data collection systems and an application that shows trends/predictions about health domain. Developers will be able to use this middleware when developing new apps, which can collect and analyse personal metrics/data in a variety of predetermined ways.

Network communication, scalability, reliability, coordination, and heterogeneity have been common requirements of traditional middleware such as remote procedure calls (RPC), message oriented middleware (MOM), distributed computing environment (DCE), transaction processing monitors (TPMON), and object oriented middleware (ORB) [park2005middleware]. Therefore, some requirements different from traditional middleware have to be considered for the middleware supporting applications and

services in the ubiquitous environment. In terms of new requirements, the Device Nimbus is designed in a way that guarantees the following (Table 1).

(i) Context should include device characteristics, user's activities/behaviour/routine, and services.

(ii) Scalability: the same middleware can be used in different scenarios: using either a single type of data collection or many others from third parties, all integrated through the middleware.

(iii) Extensibility: on top of the middleware, it will be possible to easily add new smart health services that aggregate on top of the gathered data, as well as to plug data consumers (OSGi-based). Both approaches allow generating relevant information on top of the integrated data that was collected by the integrated systems.

(iv) Flexibility: all middleware layers will be easily configurable through OSGi platform that will be accessed through management consoles. Properties, such as the routing, conversion, and storage of data, will be capable of being configured at runtime. Using the same approach, the monitoring subsystems, although autonomous, will be easily reconfigurable as well, allowing for the change of policies and strategies for analysis and prediction as well as maintenance action.

(v) Lightweight: this project will guarantee a minimum range of functionalities used by most applications.

(vi) Standards compliance: this project will utilize open standards for interfaces definition, network communication, and data representation, also allowing for the extensibility of the middleware by facilitating the integration of additional subsystems and services.

(vii) Resilience: the monitoring and maintenance mechanisms that will be orthogonal to all subsystems and components will provide the ability to predict and tolerate failures (including malicious operations), providing actions for adjusting the system's behaviour if deviations are detected.

In summary, Device Nimbus is designed in order to achieve three major measurable objectives: (i) definition and implementation of components for the data collection, (ii) definition and implementation of components for the data integration, and (iii) definition and implementation of a layer for processing and analysing the acquired data.

4. Minimum Viable Product

In order to test the proposed middleware, a MVP was developed and will be detailed in this section. A MVP is the version of a new product that allows a team to collect the maximum amount of validated learning about users and/or customers with the least effort. MVP is the core component of Lean Startup methodology that is based on the build-measure-learn feedback loop (Figure 7). The first step is to determine the problem that needs to be solved. Subsequently,

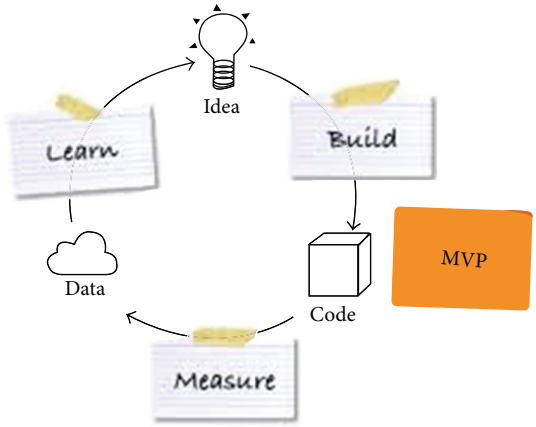

Figure 7: Minimum viable product feedback loop [51].

a MVP is developed to begin the process of learning as quickly as possible. It is important to note that, in the current status of Device Nimbus, it is working well enough to be used by the development group or in a controlled test environment, but additional work is required before uncontrolled testing can be carried out [51].

In the remainder of this section, we describe the necessary steps in the MVP by aligning with each of the key blocks/components in Device Nimbus.

Data Collectors. As part of the data collectors layer development (Figures 3 and 4), the Apache Camel [52] routine engine (with a lightweight embedded ESB) was integrated into Device Nimbus and is being used to receive data from five different physical Arduino sensors (PIR, noise, temperature, humidity, and luminosity) and from Twitter and Facebook. Apache Camel is a messaging technology glue with routing and was chosen because of its simplicity. It joins together messaging start and end points allowing for messages to be transferred from different sources to different destinations. Following the proposed architecture (Figure 3), ServiceMix Kernel was converted into an Apache Camel route container. Apache ServiceMix Kernel is a small OSGi based runtime, which provides a lightweight container onto which various bundles can be deployed.

Apache Camel has special components [53], ready to be used, to receive data from Twitter, Facebook, and many other web environments. Available as of Camel 2.10, the Twitter component enables the most useful features of the Twitter API by encapsulating Twitter4J. It allows direct, polling, or event-driven consumption of timelines, users, trends, and direct messages. Also, it supports producing messages as status updates or direct messages.

As an example, to search for all statuses with the keyword "nikeplus," Apache Camel creates routes as below:

from ("twitter://search?type=direct&**keywords**=**nikeplus**&consumerKey=[s]&consumerSecret=[s]&accessToken=[s]&accessTokenSecret=[s]").
to("db:socialnetwork").

Available as of Camel 2.12, the Facebook component provides access to all of the Facebook APIs accessible using Facebook4J. It allows for messages to retrieve, add, and delete posts, likes, comments, photos, albums, videos, photos, checkins, locations, links, and so forth. It also supports APIs that allow polling for posts, users, checkins, groups, locations, and so forth.

Based on #hashtags tracking, Device Nimbus can collect data from any health app that shares contents through #hashtags on Twitter or Facebook. This strategy helps Device Nimbus collecting data from users of proprietary wearable solutions such as Nike Plus, Run Keeper, and Garmin Connect watches, which shares contents with #hashtags in Social Networks. These proprietary data (user decides whether to personal data) also presents details about running distances, running spots, and time. Device Nimbus can collect data from users in different environments (active behaviour).

To support additional sensors and systems in Device Nimbus, Apache Camel must be updated with new and different components [53]. As a strategy to collect data from web environments, the proposed middleware was also designed based on an intelligent agent architecture. The intelligent agent was developed and added as part of the middleware to collect data from users, based on their provided logins in web environments (e.g., Twitter, Facebook, Skype, Gtalk, and others). The intelligent agent was designed to track and monitor #hashtags and to work as a chatterbot. Through natural language processing (NLP), the agent can interact with users in different environments. In the same way that sensors and systems can communicate with Device Nimbus to provide data (through the ESB), users can provide data (logins in social networks, devices that they want to be tracked and monitored, and other) to the middleware through simple and natural chats with the intelligent agent.

To better explain how the intelligent agent works in the middleware, we present an example scenario that describes the interaction between one user and the Device Nimbus. In this scenario, we assume that the middleware will be able to collect data from heterogeneous and distributed sensors (S), considering context elements (CE) to answer questions (Q):

(i) {**S**} = {Humidity, Temperature, NFC, Luminosity, PIR, Facebook, Twitter},

(ii) {**C**} = {{New posts in Facebook and Twitter, from the middleware users', using nikeplus or runkeeper apps}, {Interactions between users' and the intelligent agent of the middleware through Facebook or Twitter—NLP}, {Big climatic changes}, {Holidays and special dates}},

(iii) {**Q**} = {{Identify individuals participating in physical activities in a specific location, based on data collected from Twitter or Facebook (#hashtag) posts}; {Identify the relationship between individuals participating in physical activities in a specific location and environmental data (temperature/humidity/date/time)}; {Identify whether the same participant visited different locations using Facebook or Twitter data}; {Identify the main fitness locations in the city (city mapping)}; {Identify the main fitness locations in the city and what are the most empty/crowd date/time}; {Identify the main fitness locations in the city, what are the most empty/crowd date/time and the relationship between these locations and environmental data (temperature/humidity)}; {Identify what individuals has to say about the fitness locations of the city by tracking Twitter and Facebook hashtags}}.

To answer the questions (Q), the intelligent agent must be able to merge inputs from the distributed and heterogeneous sensors (S), considering the different CE and routine of each user. There are people that like exercising in the rain, while others love exercising on sunny days, for example. To identify the popular fitness locations in the city, the intelligent agent must be able to track the users. By merging their fitness location, day of the week, time of the day, and frequency that they run/exercise, the intelligent will be able to provide rich and personalised feedback to each user, based on their needs.

To ensure the quality of data collected from Twitter and Facebook, the intelligent agent and the Apache Camel are both looking for the same #hashtags and users (logins were provided as input). To collect data from Twitter, the intelligent agent is looking at data from Twitter Sample Streaming Data [54]. To get connected with Facebook, Facebook4J is being used [53].

A single instance of an intelligent agent, which is provided by Device Nimbus middleware, can be available in alternative environments (such as a contact on Skype and GTalk or as an user in Twitter or Facebook). Despite the fact that the intelligent agent tracks special #hashtags from Device Nimbus users and appears in many different environments, the middleware provides a single agent to them all. In other words, the user can chat about their health or routine across different environments with the same bot. If a user starts communicating with the intelligent agent in Gtalk, asking him about good spots to run: "where can I run in Melbourne?" they will get an answer. In parallel, the intelligent agent will be monitoring many different users on Twitter and Facebok and will be able to identify where most of them are running in the city, days of the week that people mostly run, the time of the day they run, and so forth. To provide high quality answers, the data collected from environmental sensors and other data sources will also be considered. By providing an interface of Device Nimbus intelligent agent as a chatterbot, more data can be collected and analysed from different users in different environments.

To be tracked (monitored) by Device Nimbus, each user must use special commands such as "#addEnvironment Twitter oliveiraeduardo" to set a new login in the middleware (user is in Facebook adding a Twitter login, e.g., teaching the bot about other logins distributed in the Web). The advantage of sharing logins with Device Nimbus is related to the ability of the middleware to provide health support and assistance to the user. It is important to note that only the owner of each login recorded in the middleware has access to their personal information and feedback.

Natural language processing is used by the intelligent agent of the middleware to communicate with the users in the various integrated Web environments. We have used

the ProgramD [55] library and Drools [56] inference engine (rule-based reasoning) to design and develop the intelligent agent fitness knowledge base. Drools is responsible for integrating users distributed data and for considering context while users are interacting with the intelligent agent of Device Nimbus. The knowledge-api, drools-core, drools-compiler, and drools-decisiontables modules are working with OSGi. ProgramD is a fully functional artificial intelligence markup language (AIML) bot engine that is implemented with Java. It supports multiple bots, and it is easy to configure and runs in a GUI application and also under a J2EE environment. AIML is an XML dialect for creating natural language software agents. When the AIML markup language is loaded for the bot, users can chat with the bot.

To create custom responses in Device Nimbus, each AIML content uses special metadata: a feature not supported by Programd library. In order to make the AIML more powerful, we have updated the Programd library and adopted the use of embedded metadata to AIML, to identify the contents requested by each user and his doubts, as shown in the example:

```
<category>
  <pattern> * PLACES * RUN * MELBOURNE
</pattern>
  <template>
   <random>
<li>#content=sports;subcontent=running;interest
=spot# Royal Botanic Gardens and St Kilda beach.
Different options are Royal Park and Princess Park.
Enjoy;)</li>
<li>#content=sports;subcontent=running;interest
=spot# What about Alexandra Ave, Royal Botanic
Gardens or Albert Park? If you prefer beaches, you
must try St Kilda. Different options are Royal Park
and Princess Park. Enjoy;)</li>
   </random>
  </template>
</category>
```

The metadata is used in the middleware for recommendation and personalization of contents based on users' profiles and needs. To identify the inputs provided by users (questions and interactions) with the tags defined in AIML, we use an interpreter that uses pattern matching techniques. The input text from users should match with the text that is inside the <pattern> AIML tag to provide an output. AIML also allows for the use of regular expressions in its tags, to improve the matches' scenarios. The new programd interpreter, updated for the middleware, was developed in Java and will be distributed under the same license used by programd. Now, even with the AIML static knowledge base (when → then), Device Nimbus is able to provide dynamic and personalized answers because of the metadata support.

The advantage of providing a single intelligent agent in the middleware lies in the fact that, with only one agent, Device Nimbus can also have a single integrated database. If a user interacts with the intelligent agent through Facebook, the agent will know, referring to the historical database of the user that they have already communicated with then through Twitter and Skype, and that s/he has demonstrated interest in running spots. At the same time, the intelligent agent is able to integrate these data with data that comes from #hashtags or other different wearable devices, modeling every user based on their routine and unique needs.

Data Integration. As part of the Data Integration layer development (Figures 3 and 5), the NOSQL can receive data from the intelligent agent and the Apache Camel. Two different databases are being used: (i) general database, which has data collected from different sensors/devices, received from Apache Camel or through the intelligent agent; (ii) contextual database, with analysed data from different users. The intelligent modules are responsible for updating the contextual database, based on analysis on general database. Apache Camel and the intelligent agent of Device Nimbus are responsible for updating the general database. The idea of a contextual database is to improve performance in providing feedback to users. Device Nimbus cloud infrastructure is running at the National eResearch Collaboration Tools and Resources project (NeCTAR) (https://www.nectar.org.au/). Device Nimbus is running in the following NeCTAR instance: Ubuntu 13.10 (Saucy) OS, 4 GB RAM | 1 VCPU | 10.0 GB Disk machine. To deploy the middleware we also used Java 1.7, Apache Tomcat v8.0.14, NoSQL v3.2.5, Apache Camel v2.14.0, JBoss Drools v6. The adopted code repository was the Bitbucket.

Intelligent Modules. As part of the intelligent modules development (Figures 3 and 6), a rules-based Knowledge was developed with JBoss Drools [56]. This knowledge based is integrated with the proposed middleware, as part of the intelligent module.

The intelligent module is able to identify patterns between data collected from different sources and integrated in the general database. Data collected from Twitter (#nikeplus) can be combined with environmental data (temperature and humidity) to discover new information about users (frequency of running, patterns between running activities and the weather, and others). Based on this analysis, Device Nimbus is able to understand about the routine of a user or of a set of users. Based on this understanding, Device Nimbus can provide personalized health feedback to users.

The contextual intelligent module is responsible for monitoring users through updating the contextual database based on users profile, behaviour, and routines. With JBoss Drools, rules are responsible for analysing new data on general database and for analysing and updating the contextual database. The feedback provided to users is triggered from the contextual database.

During this minimum viable product development, the resilience of the overall infrastructure including the data collection subsystems and design of the core infrastructure with extensible features and integrated visual environments for data management and monitoring was not developed.

As the next step, a case-based knowledge and predictive analysis technique will be added as part of the intelligent module.

5. Demonstration of Device Nimbus Using Fitness and Wellbeing Apps

In this section, we report the results from a series of tests carried out to explore the efficacy of the MVP implementation of Device Nimbus. Given constraints, we have nominated fitness and wellbeing apps as our primary source of data from the wider health domain (in future work, experiments will be conducted using some of the health apps listed in Section 2.1 instead of the fitness and wellbeing apps; such experiments will require collaboration with medical professionals, clinics, and hospitals). We construct a number of alternative scenarios where individuals engaged in physical activities interact with the middleware when using fitness apps and social media. We are particularly interested in describing how information gleaned from the collected data could be used to provide both timely and useful feedback for the individuals concerned.

Each of the components of the proposed middleware (as listed in Sections 3 and 4) was tested. Figure 8 provides a high-level overview of the scenarios. The specific test details for each component are listed below.

(i) Data collection:

  (a) test data collection from Twitter, Gtalk, and Skype;
  (b) test data collection from environmental sensors (temperature, humidity, and luminosity).

(ii) Data integration:

  (a) test the ability to the middleware to integrate the heterogeneous data into the NoSQL database;
  (b) test the ability of the intelligent agent of the middleware to integrate the users' data—tracked in Twitter or through interactions with him—into the NoSQL database.

(iii) Intelligent modules:

  (a) test the ability to the middleware to analyse the integrated data and physical activities.

Data collection was limited to five individual participants, engaged in physical activities. Data was collected using Nikeplus and RunKeeper fitness apps, via environmental/meteorological data, and via an analysis of interactions with the intelligent agent of the middleware using Gtalk, Skype instant messenger, and Twitter. Fitness activities that were shared on Twitter are written to an individual participant's time-dated content report. It is important to note that Device Nimbus tracking requires participants to agree and accept an invitation, after which their details are added to the NoSQL general database and their activities are constantly monitored.

For security and privacy reasons, the collected, integrated, and analysed data is just shared with each user that requests to be tracked, by chatting with the intelligent agent. The intelligent agent never collects data from users without their permission and never shares information between different users.

Environmental data was collected via a small meteorological station, developed as part of this experiment. By collecting environmental data, the middleware can be used to identify the correlation between rates of physical activities and specific weather conditions. In the preliminary MVP tests, we limit environmental data collection to humidity, temperature, and luminosity. Further testing will be carried out utilising the NFC, PIR, and noise sensors.

The test data was collected over a three-week period. The environmental data was collected every 30 mins from 07 h to 22 h. The data collected from the physical sensors (recorded on a SD card) was added to the Device Nimbus database on a daily basis.

The systems integrated with the Device Nimbus—Gtalk, Twitter, and Skype—were monitored closely. In the preliminary MVP test, Facebook data was removed to simplify the analysis. Twitter hashtags (#nikeplus and #runkeeper) from the participants were monitored 24 hours a day during the test period. The database was automatically updated by Apache Camel every time that a new #hashtag was found. All interactions between the intelligent agent and participants in Twitter, Gtalk, and Skype environments were recorded.

The participants hashtags shared in Twitter were tracked and added to the NoSQL general database by the ESB. Interactions between the middleware intelligent agent and the participants were added into the NoSQL general database by the agent. A rule-based knowledge system and JBoss Drools inference engine was then used to answer the set of questions (Q) presented in Section 4.

Results and Discussion. In the first phase of our analysis, we document the interactions between the participants and Device Nimbus's intelligent agent. Here, our aim was to determine whether the intelligent agent was able to (a) identify the same participant in different instant messengers and Twitter messages based on their different logins and (b) provide personalized responses based on individual's needs and log analysis.

Over one hundred interactions were recorded between individual participants and the intelligent agent. An analysis of the logs showed that the intelligent agent was able to integrate different logins from the same user. Significantly, it was able to identify the same individual in different environments and to personalize contents based on historical and contextual data (through AIML metadata contents). One clear observation was that the conversations extracted from the fitness domain must be improved to increase the participants' experience. In many cases, the intelligent agent was not able to answer general questions about the domain. A summary of the interactions on Twitter, Gtalk, and Skype with the intelligent agent suggests that (i) more than 60% of the questions were about general content; (ii) 30% of the questions were about fitness exercises and locations; (iii)

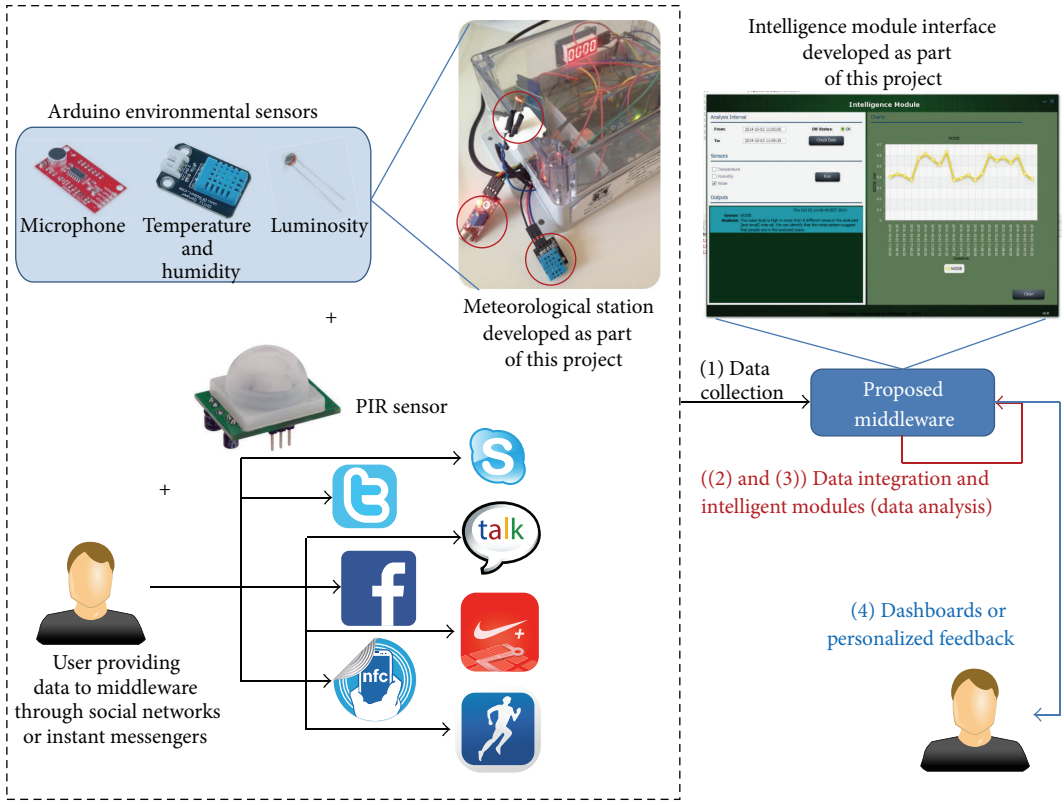

Figure 8: Device Nimbus inputs, components, and general flow.

10% of the questions focused on support and help while interacting with the intelligent agent.

At this stage of the MVP implementation, the intelligent agent was not able provide personalized dashboards (information) to the different participants. The intelligent agent was able to interact via natural language processing and provide tips about fitness locations, to receive tracking requests from the participants and to integrate different logins from a participant in different web environments.

In the second phase of analysis, a detailed review of Device Nimbus' logs and individual users' fitness reports (location, date, and time of their contents shared in Twitter) were analysed. We observed that the Twitter sample streaming data used by Device Nimbus was not sufficient to track users' #hashtags on Twitter with reliability, when compared with the number of running shared times reported by users (37 times). Device Nimbus was only able to track 8 of these shared tweets. This may be attributed in part to the fact that Device Nimbus was using the sample streaming channel of Twitter [54], which led to significant loss of data on this public channel. This stream provided millions of messages to Device Nimbus. These messages were filtered, one by one, by JBoss Drools. Only messages from known participants and with special #nikeplus and #runkeeper hashtags were recorded in general database. An alternative approach to improve data collection from Twitter will be to use Firehose, a more complete streaming channel (special permission should be requested) or to request that users send a direct message to the intelligent agent. Direct messages are easy to be tracked. Collecting Twitter data with Apache Camel did not mitigate this problem.

In the final analysis phase, our goal was to verify if Device Nimbus, supported by the intelligent modules, was able to analyse the integrated environmental and Twitter data in order to answer the set of Questions (Q) presented in Section 4.

The tracked Twitter messages in Device Nimbus were always read in the same format: **#user#USER NAME#message content#CONTENT#HASHTAG#location#LOCATION NAME#timezone#TIMEZONE**.

By analysing a simple example of a real tracked message, **#user#oliveiraeduardo#text#2:07pm, class registration progress: 0%. #nikeplus#location#Royal Botanical Gardens, Melbourne#timezone#null**, we can see that it is easy to extract information such as (i) individuals participating in physical activities, (ii) the location of the physical activity and (iii) the time of the training and the date, by considering the day of the post. Once tracking participants' hashtags, the intelligent agent of Device Nimbus was able to split and analyse Twitter posts in order to answer questions Q1, Q3, Q4, and Q5 (Table 2).

For every tracked participant post in Twitter, the intelligent agent was responsible for updating the contextual database of Device Nimbus, consolidating the locations where the participants were exercising with their activities date and time. Given the small number of participants in the

Table 2: Questions to be answered in the experiment by analysing environmental data together with Twitter.

Questions versus Sensors	Humidity	Temperature	Luminosity	Twitter	Able to answer?
Q1: identify individuals participating in physical activities in a specific location, based on data collected from #hashtag posts				X	Yes
Q2: identify the relationship between individuals participating in physical activities in a specific location and environmental data	X	X	X	X	Yes
Q3: identify whether the same participant visited different locations, based on data collected from #hashtag posts				X	Yes
Q4: identify the main fitness locations in the city (city mapping)				X	Yes
Q5: identify the main fitness locations in the city and what are the most empty/crowd date/time				X	Yes
Q6: identify the main fitness locations in the city, what are the most empty/crowd date/time and the relationship between these locations and environmental data	X	X	X	X	Yes
Q7: identify what individuals have to say about the fitness locations of the city, based on data collected from #hashtag posts				X	No

MVP tests, it is not possible to answer Q4 on the large scale (such as across the city or suburban area). However, we have assumed that the question was answered because the intelligent agent could identify if the different participants ran, in the three weeks period of experiment, in the same location. The same strategy was adopted to answer the question Q5. In future work, it will be interesting to increase the scale of the tests to include more participants and locations.

By merging of the environmental data and the Twitter data, the intelligent module was able to identify correlations between individual's physical activities and the weather, in order to answer questions Q2 and Q6. To support the data analysis understanding, an interface was developed as part of this experiment (Figure 8). The system interface uses graphical representations for the collected and integrated environmental and Twitter data in both the general and contextual database, in order to support the data patterns recognition. To analyse the integrated data, 16 different rules were created and added to the intelligent agent knowledge in the JBoss Drools, as part of the intelligent module.

As an example of the interface output, depending on a time interval filter, parameterized in the interface system, the temperature, humidity, days of the week, time, and Twitter posts data were combined and analysed by the intelligent module. With this simple experiment, the Device Nimbus was able to identify that (i) when the weather was too hot or cold, or it was raining, all participants avoided outdoor spaces; (ii) 4 participants exercised mainly in the morning and in the night; (iii) the most common fitness/exercising days were on Monday, Tuesday, and Thursday; (iv) 2 participants ran every Saturday and every Sunday during the experiment period; (v) the most shared running location in Melbourne was the Royal Botanical Gardens.

An important outcome from this small suite of tests is the fact that data harvested from multiple sources, when combined and processed, paints a very clear picture describing the physical activities occurring in specific locations. Given the status of the MVP of Device Nimbus, the administrative tools have not been fully implemented. However, ongoing development is under way.

6. Conclusions and Further Works

Large global technology providers are consistently investing in ICT solutions for health and fitness systems. Unfortunately, many of these solutions are closed, preventing the development of solutions provided by local market platforms.

In this paper, we have proposed a novel intelligent middleware of services, Device Nimbus. An important design feature of the middleware is the inclusion of an environment for the integration and aggregation of services, applications, and health/fitness data. The rationale, on which our middleware design is based, is that it should be flexible and extensible. Third party services will be able to build systems and apps on top of the Device Nimbus thus forming an "ecosystem" around the middleware.

Device Nimbus' middleware encapsulated key blocks (or components) for data collection from heterogeneous sensors and devices, data retrieval, preprocessing, storage or management, and an application layer. The intelligent agent module allows for prediction capabilities and context awareness, allowing for the dissemination of useful and timely information for users in dynamic environments.

A MVP of Device Nimbus was deployed to investigate the effectiveness of the overall middleware design. A series of small experiments were carried out, focusing on the use of fitness apps, and social media in a physical activity scenario. The results obtained from the experiments, focused on mobile fitness apps, suggest that the intelligent middleware can be used to effectively monitor users' routines. The full implementation of Device Nimbus work is currently under way in a collaboration project between The University of Melbourne, Australia and the Federal University of Pernambuco, Brazil.

In future work, we will carry out more detailed experiments using a range of health wearable devices. By collecting

physiological inputs such as heart rate, blood pressure, body and skin temperature, oxygen saturation, respiration rate, electrocardiogram, and others, Device Nimbus will be able to provide users' with personalized preventive care, providing early warning signs for serious illnesses.

Conflict of Interests

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

Acknowledgment

Eduardo Araujo Oliveira would like to thank the National Council for Scientific and Technological Development (CNPq), Brazil, for supporting his postdoc position (scholarship provided under report n° BEX 9213/13-9).

References

[1] 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.

[2] A. Lymberis and A. Dittmar, "Advanced wearable health systems and applications—research and development efforts in the European union," *IEEE Engineering in Medicine and Biology Magazine*, vol. 26, no. 3, pp. 29–33, 2007.

[3] G. Troster, "The agenda of wearable healthcare," in *IMIA Yearbook of Medical Informatics*, pp. 125–138, Schattauer, Stuttgart, Germany, 2005.

[4] A. Pantelopoulos and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 40, no. 1, pp. 1–12, 2010.

[5] R. Schmohl and U. Baumgarten, "A generalized context-aware architecture in heterogeneous mobile computing environments," in *Proceedings of the 4th International Conference on Wireless and Mobile Communications (ICWMC '08)*, pp. 118–124, IEEE, Athens, Greece, August 2008.

[6] E. A. Oliveira, V. C. O. Aureliano, R. S. de França, and P. C. A. R. Tedesco, "Digital and collaborative technologies for smarter distance education," *Revista de Sistemas de Informação da FSMA*, no. 14, pp. 39–47, 2014.

[7] C. W. Mundt, K. N. Montgomery, U. E. Udoh et al., "A multiparameter wearable physiologic monitoring system for space and terrestrial applications," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 3, pp. 382–391, 2005.

[8] V. Shnayder, B. R. Chen, K. Lorincz, T. R. F. Fulford-Jones, and M. Welsh, "Sensor networks for medical care," Tech. Rep. TR-08-05, Division of Engineering and Applied Science, Harvard University, Cambridge, Mass, USA, 2005.

[9] A. Wood, G. Virone, T. Doan et al., "ALARM-NET: wireless sensor networks for assisted-living and residential monitoring," Tech. Rep. CS-2006-11, Department of Computer Science, University of Virginia, 2006.

[10] J. Luprano, J. Sola, S. Dasen, J. M. Koller, and O. Chetelat, "Combination of body sensor networks and on-body signal processing algorithms: the practical case of my heart project," in *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, pp. 76–79, April 2006.

[11] K. J. HeilmanandS and S. W. Porges, "Accuracy of the Life shirt (Vivometrics) in the detection of cardiac rhythms," *Biological Psychology*, vol. 3, pp. 300–305, 2007.

[12] J. Ren, C. Chien, and C. C. Tai, "A new wireless-type physiological signal measuring system using a PDA and the bluetooth technology," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT '06)*, pp. 3026–3031, Mumbai, India, December 2006.

[13] M. Sung, C. Marci, and A. Pentland, "Wearable feedback systems for rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 2, article 17, 2005.

[14] Nimbits, http://www.nimbits.com/.

[15] ThingSpeak, http://www.thingspeak.com/.

[16] WSO2, http://www.wso2.com/.

[17] M. Blackstock, N. Kaviani, R. Lea, and A. Friday, "MAGIC broker 2: an open and extensible platform for the internet of things," in *Proceedings of the 2nd International Internet of Things Conference (IOT '10)*, pp. 1–8, December 2010.

[18] A. Erbad, M. Blackstock, A. Friday, R. Lea, and J. Al-Muhtadi, "MAGIC broker: a middleware toolkit for interactive public displays," in *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '08)*, pp. 509–514, IEEE Computer Society, Hong Kong, March 2008.

[19] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci, "Smart City: an Event Driven Architecture for monitoring public spaces with heterogeneous sensors," in *Proceedings of the 4th International Conference on Sensor Technologies and Applications (SENSORCOMM '10)*, pp. 281–286, July 2010.

[20] Xively, https://xively.com/.

[21] D. O. Kang, H. J. Lee, E. J. Ko, K. Kang, and J. Lee, "A wearable context aware system for ubiquitous healthcare," in *Proceedings of the 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS '06)*, pp. 5192–5195, New York, NY, USA, August 2006.

[22] A. R. Chowdhury, B. Falchuk, and A. Misra, "MediAlly: a provenance-aware remote health monitoring middleware," in *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications (PerCom '10)*, pp. 125–134, IEEE, April 2010.

[23] Microsoft Health, http://www.microsoft.com/microsoft-health/en-us.

[24] Samsung SAMI, http://www.samsung.com/us/globalinnovation/media/.

[25] SimBand, http://www.samsung.com/us/globalinnovation/innovation_areas/.

[26] Samsung Digital Health SDK, http://developer.samsung.com/health.

[27] A. Triantafyllidis, V. Koutkias, I. Chouvarda, and N. Maglaveras, "An open and reconfigurable wireless sensor network for pervasive health monitoring," in *Proceedings of the 2nd International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth '08)*, pp. 112–115, IEEE, Tampere, Finland, February 2008.

[28] T. S. López and D. Kim, "Wireless sensor networks and RFID integration for context aware services," Artigo publicado no Web Site do Auto-ID Labs, 2008, http://cocoa.ethz.ch/media/documents/2014/06/archive/withhold_AUTOIDLABS-WP-SW-NET-026.pdf.

[29] A. B. Waluyo, S. Ying, I. Pek, and J. K. Wu, "Middleware for wireless medical body area network," in *Proceedings of the IEEE*

Biomedical Circuits and Systems Conference (BiOCAS '07), pp. 183–186, IEEE, November 2007.

[30] H.-F. Lu and J.-L. Chen, "Design of middleware for telehomecare systems," *Wireless Communications and Mobile Computing*, vol. 9, no. 12, pp. 1553–1564, 2009.

[31] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, "50 ways to build your application: a survey of middleware and systems for wireless sensor networks," in *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '07)*, pp. 466–473, IEEE, Patras, Greece, September 2007.

[32] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2688–2710, 2010.

[33] IBM, http://researcher.watson.ibm.com/researcher/view_group.php?id=4477.

[34] Microsoft, http://www.microsoft.com/health/en-au/solutions/Pages/smart-devices.aspx.

[35] Google, http://googleblog.blogspot.com.au/2014/03/sharing-whats-up-our-sleeve-android.html.

[36] Samsung, http://www.samsung.com/uk/business/solutions-services/mobile-solutions/healthcare/samsung-m-emr.

[37] Apple, http://techcrunch.com/2014/06/03/apples-health-offerings-focus-on-data-collection-not-interpretation/.

[38] Open Health Tools, http://www.openhealthtools.org/.

[39] D. Manzaroli, L. Roffia, T. S. Cinotti et al., "Smart-M3 and OSGi: the interoperability platform," in *Proceedings of the 15th IEEE Symposium on Computers and Communications (ISCC '10)*, pp. 1053–1058, June 2010.

[40] Etherios, http://www.idigi.com/.

[41] P. Gong, D. Feng, and Y. S. Lim, "An intelligent middleware for dynamic integration of heterogeneous health care applications," in *Proceedings of the 11th International Multimedia Modelling Conference (MMM '05)*, pp. 198–205, January 2005.

[42] N.-S. Park, K.-W. Lee, and H. Kim, "A middleware for supporting context-aware services in mobile and ubiquitous environment," in *Proceedings of the International Conference on Mobile Business (ICMB '05)*, pp. 694–697, IEEE, July 2005.

[43] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Proceedings of the IEEE CS 4th International Conference on Web Information Systems Engineering (WISE '03)*, pp. 3–12, December 2003.

[44] K. Gama, L. Touseau, and D. Donsez, "Combining heterogeneous service technologies for building an Internet of Things middleware," *Computer Communications*, vol. 35, no. 4, pp. 405–417, 2012.

[45] Equinox, http://www.eclipse.org/equinox/.

[46] Knopflerfish Open Source OSGi Sevice Platfurm, http://www.knopflerfish.org/.

[47] Apache Felix, http://felix.apache.org/.

[48] Concierge OSGi, http://concierge.sourceforge.net/.

[49] ServiceMix, http://servicemix.apache.org/.

[50] JBoss Fuse, http://www.jboss.org/products/fuse/overview/.

[51] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*, Random House, 2011.

[52] Apache Camel, http://camel.apache.org/.

[53] Apache Camel Components, http://camel.apache.org/components.html.

[54] Twitter Sample Streaming API, https://dev.twitter.com/streaming/public.

[55] ProgramD, https://code.google.com/p/ainotebook/wiki/programd_aiml.

[56] JBoss Drools, http://www.drools.org/.